2013

Bài giảng Hệ Thống Nhúng

cuu duong than cong . com

Đậu Trọng Hiển

Trương Ngọc Sơn

SPKT

Chương 1

GIỚI THIỆU HỆ THỐNG NHÚNG

1.Hệ thống nhúng (Embedded System):

Hiện nay hệ thống nhúng đã và đang từng bước phát triển ở Việt nam, nó thay cho các hệ thống vi xử lý trước đây. Hệ thống nhúng được ứng dụng rộng rãi trong ngành điện tử, máy tính và viễn thông như các hệ thống điện thoại, các máy đo, các hệ thống điều khiển tự động trong công nghiệp, thương mại và ngân hàng. Tuy nhiên chúng ta vẫn chưa có một định nghĩa cụ thể về hệ thống nhúng. Thông qua quá trình vận hành, xây dựng và phát triển hệ thống nhúng chúng ta có thể hiểu hệ thống nhúng như sau:

Hệ thống nhúng là một ứng dụng bao gồm ít nhất một thiết bị lập trình được như vi xử lý, vi điều khiển hay các vi mạch xử lý số. Nó là một hệ thống dựa trên vi xử lý để thực hiện một chức năng hay một dãy chức năng cụ thể nào đó.

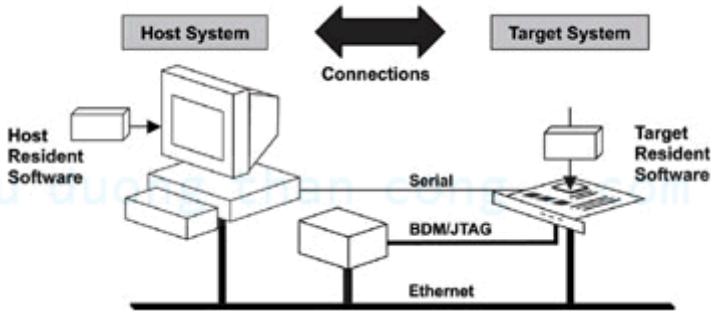
Hệ thống nhúng là một ứng dụng được tích hợp cả phần cứng và phần mềm nhằm phục vụ các bài toán chuyên dụng trong lĩnh vực công nghiệp, y tế, quân sự.

Một máy tính PC là một thiết bị có nhiều chức năng và người sử dụng có thể thay đổi các chức năng thông qua việc thêm, xóa phần mềm ứng dụng, trong khi đó hệ thống nhúng được thiết kế để phục vụ một số chức năng cụ thể, xác định. Chính vì thế hệ thống nhúng được các nhà phát triển tối ưu hóa nó nhằm giảm thiểu kích thước và chi phí sản xuất.

Các thiết bị cầm tay PDA cũng có đặc điểm giống hệ thống nhúng nhưng chúng không phải là hệ thống nhúng vì chúng có nhiều chức năng.

Để thay đổi chức năng của hệ thống nhúng thông thường người ta dựa và các công cụ phát triển và công việc này do các chuyên gia phát triển hệ thống nhúng thực thiện. Quá trình xây dựng lại chức năng hệ thống nhúng giống như quá trình thay đổi chức năng hệ điều hành, thông thường người ta thay đổi, sửa chữa, thêm, xóa các trình điều khiển, hoạt động của hệ thống sau đó tiến hành biên dịch lại cho hệ thống nhúng.

Một hệ thống nhúng được kết nối với một hệ thống máy chủ để phát triển hệ thống, quá trình phát triển hệ thống hay lập trình cho hệ thống nhúng được thực hiện trên một máy tính có các công cụ hỗ trợ. Các kết quả của các quá trình biên dịch như: các tập tin ảnh của hệ điều hành, các tập tin thực thi… được nạp xuống hệ thống nhúng thông qua các kết nối như: *serial, usb, ethernet..*



Hình 1-1 Kết nối trong quá trình phát triển hệ thống nhúng

Một hệ thống nhúng thông thường có các thành phần sau:

- **Vi xử lý:** thông thường là các vi xử lý 32 bit, các vi xử lý đóng vai trò bộ xử lý trung tâm trong hệ thống nhúng, ngày nay với sự phát triển của ngành công nghiệp điện tử, nhiều hãng sản xuất vi xử lý cho ra đời các chip vi xử lý 32 bit với nhiều tính năng tích hợp phục vụ trong hệ thống nhúng như Renesas với các chip họ SH, AMCC với PowerPC, Cirrus Logic với ARM7, ARM9, Atmel…

CuuDuongThanCong.com https://fb.com/tailieudientucntt

- **Bộ nhớ:** bao gồm bộ nhớ RAM, EEPROM hay Flash ROM.

- Các ngoại vi bao gồm các giao tiếp IO như USB, Ethernet, PCI… Tùy vào mục đích, yêu cầu của mỗi hệ thống khác nhau mà thiết kế các ngoại vi khác nhau, trong đó có một số ngoại vi chung như *ethernet, usb, serial*. Các ngoại vi này vừa là các giao tiếp của hệ thống trong các ứng dụng vừa làm nhiệm vụ truyền dữ liệu trong quá trình nạp phần mềm cho hệ thống hay gỡ rối hệ thống.

Phần mềm trong hệ thống nhúng: Phần mềm là chương trình điều khiển hoạt động của hệ thống nhúng, trong một số hệ thống nhúng phần mềm còn được gọi là hệ điều hành nhúng. Nó giống như một hệ điều hành chạy trên máy tính nhưng chúng được các nhà phát triển tối ưu sao cho có thể vận hành hiệu quả trên hệ thống có bộ nhớ và tốc độ xử lý giới hạn.

Một số hệ điều hành chạy trên hệ thống nhúng là *Linux, QNX, Windows CE*...

Trong các hệ thống nhúng sử dụng hệ điều hành Linux, bộ phần mềm gồm các phần như sau:

- **Bootloader, uboot, redboot:** phần mềm khởi động hệ thống
- **Kernel:** Nhân của hệ điều hành
- **File system:** Hệ thống tập tin

Một số hệ thống nhúng quanh ta như các thiết bị nghe nhìn, các thiết bị trong khoa học kỹ thuật, các thiết bị phục vụ trong đời sống tinh thần…

Hình 1-2 Một số hệ thống nhúng

2.Hệ thống thời gian thực (Real-time operating system_ RTOS):

Trong các bài toán điều khiển chúng ta hay bắt gặp các thuật ngữ " Thời gian thực".

Thời gian thực không phải là thời gian phản ánh một cách trung thực chính xác thời gian hay yêu cầu thời gian hệ thống phải trùng với thời gian thực tế.

Hệ thống thời gian thực được hiểu là các hoạt động của hệ thống phải thỏa mãn về tính tiền định. Tính tiền định là hành vi của hệ thống phải được thực hiện đúng trong một khung thời gian cho trước hoàn toàn xác định, khung thời gian này được quyết định bởi đặc điểm và yêu cầu của hệ thống.

Thực tế cho thấy rằng hầu hết các hệ thống nhúng là các hệ thống thời gian thực và ngược lại hầu hết các hệ thống thời gian thực là các hệ thống nhúng.

3.Đặc điểm của hệ thống nhúng:

Hệ thống nhúng có một số đặc điểm sau:

- Độ tin cậy cao.

CuuDuongThanCong.com https://fb.com/tailieudientucntt

- Có khả năng bảo trì và nâng cấp.
- Hiệu quả về thời gian thực hiện.
- Kích thước, khối lượng nhỏ.

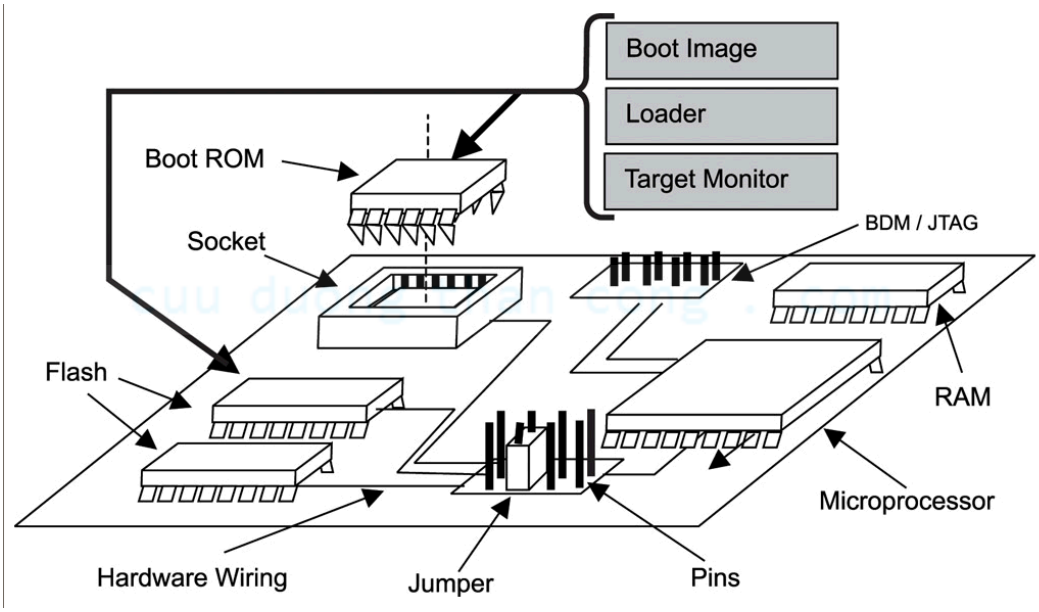
4.Các khái niệm sử dụng trong hệ thống nhúng:

4.1 Quá trình khởi động hệ thống:

- Image thực thi chương trình được biên dịch cho hệ thống nhúng có thể được truyền từ công cụ phát triển hệ thống nhúng (Host) vào hệ thống nhúng (Target) quá trình này được gọi là "Loading the Image".

- **Image có thể được load và hệ thống nhúng thông qua các cách như sau:**

- Load Image vào bộ nhớ EEPROM hay Flash.
- Download Image trực tiếp lên bộ nhớ SRAM của hệ thống nhúng thông qua cổng nối tiếp RS232 hay cổng mạng (ethernet) quá trình này đòi hỏi một số trình ứng dụng chạy trên Host và Target như Embedded Monitor, Embedded Loader, Target debug…
- Download Image thông qua JTAG.



Hình 1-3 Hệ thống nhúng cơ bản

- **Embedded Loader:** là một chương trình được nạp vào hệ thống nhúng đầu tiên, Embedded Loader được hiểu giống như BIOS của máy tính. Embedded loader có dung lượng nhỏ nên thông thường được nạp vào ROM, trên các hệ thống vi xử lý nhỏ, Loader được nạp vào một vùng riêng trên vi xử lý.

Chương trình Embedded Loader có nhiệm vụ kết nối với Host trong quá trình truyền file ảnh (Image) xuống hệ thống nhúng.

Để có thể truyền dữ liệu giữa Host và Target, giao thức truyền được xây dựng sao cho các tiện ích chạy trên Host và Embedded Loader trên Target điều hoạt động theo các thông số của giao thức này.

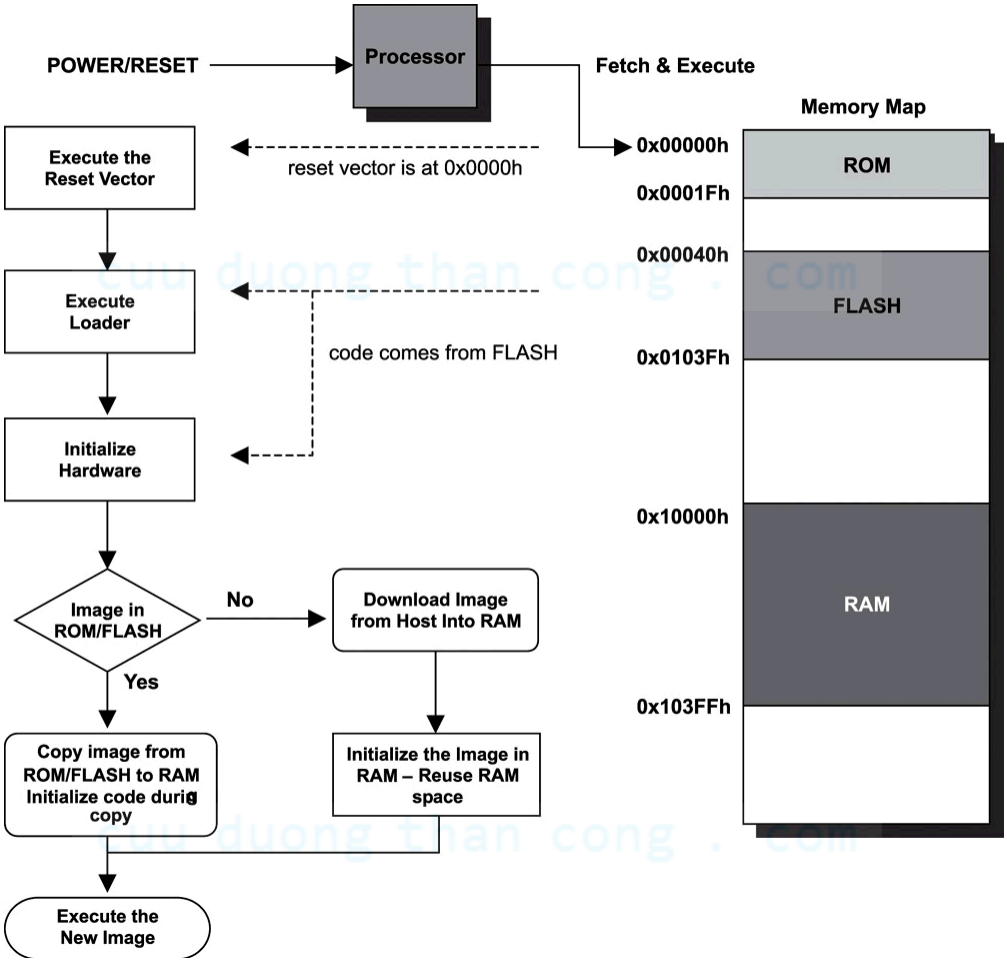
Tùy theo mỗi nhà sản xuất vi xử lý sẽ có các Embedded Loader riêng, ví dụ các vi xử lý Atmel thì có Bootstrap và uboot, PowerPC thì sử dụng uboot, Cirrus Logic thì sử dụng redboot. Một số chip vi xử lý khác thì sử dụng bootloader.

- **Embedded Monitor:** là một phần mềm ứng dụng trên hệ thống nhúng thông thường được cung cấp bởi các nhà sản xuất. Nó cho phép các nhà phát triển hệ thống có thể gỡ rối hệ thống, giống như một chương trình boot, Embedded Monitor thực thi khi hệ thống được cấp nguồn và thực hiện một số thao tác trên hệ thống như sau:

CuuDuongThanCong.com https://fb.com/tailieudientucntt

- Khởi tạo, thiết lập cho các thiết bị ngoại vi như cổng nối tiếp, bộ định thời chip, số lần làm tươi RAM…
- Khởi tạo bộ nhớ hệ thống chuẩn bị cho quá trình download Image.
- Khởi tạo chương trình điều khiển ngắt, cài đặt các ngắt hệ thống.

- Embedded Monitor hỗ trợ 1 công cụ giao tiếp người dùng thông qua giao tiếp nối tiếp đến các chương trình mô phỏng. Thông thường thì nó hỗ trợ các lệnh điều khiển như sau:
  - Download Image.
  - Đọc và ghi các thanh ghi hệ thống.
  - Đọc và ghi bộ nhớ hệ thống.
  - Thiết lập và xóa các break point.
  - Cho phép thực thi từng lệnh để gở rối hệ thống.
  - Reset và reboot hệ thống.
- **Quá trình boot của hệ thống:**



Hình 1-4 Quá trình khởi động hệ thống

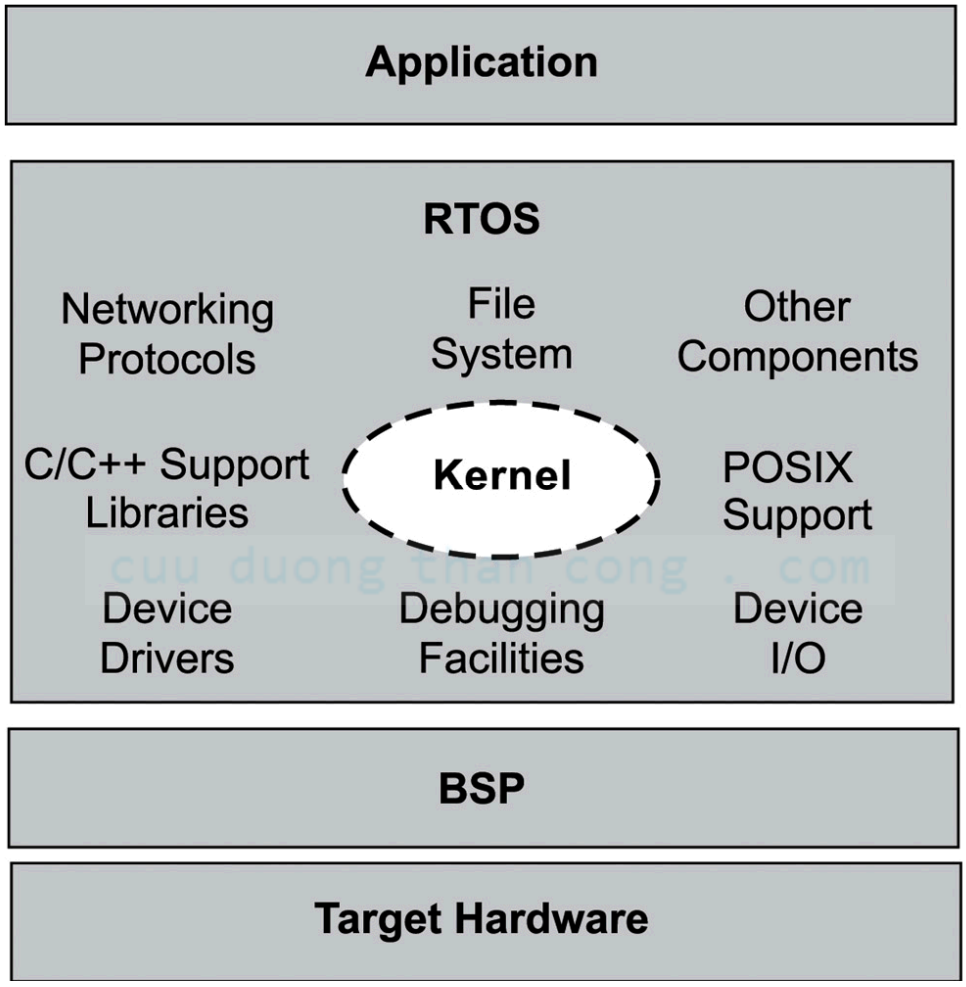
5.Hệ điều hành thời gian thực (Real-time Operating System)

- Hệ điều hành thời gian thực là một chương trình lập lịch cho các hoạt động của hệ thống thi hành chính xác theo thời gian định trước, quản lý tài nguyên hệ thống và cung cấp một sự

CuuDuongThanCong.com https://fb.com/tailieudientucntt

thiết lập thích hợp cho quá trình phát triển mã nguồn của hệ thống. Mã nguồn hệ thống có thể thay đổi được.

- Trong một số ứng dụng, RTOS bao gồm một kernel (nhân). Kernel là một lõi mềm giám sát hệ thống, cung cấp các khối logic, các giải thuật lập lịch, các giải thuật quản lý tài nguyên. Mỗi một hệ thống thời gian thực đều có một kernel. Mỗi hệ thống thời gian thực là một sự tổng hợp của nhiều module trong đó bao gồm kernel, file system, network protocol stack, và các module khác tùy thuộc vào yêu cầu chức năng của hệ thống.



Hình 1-5 Tổ chức trong hệ thống nhúng

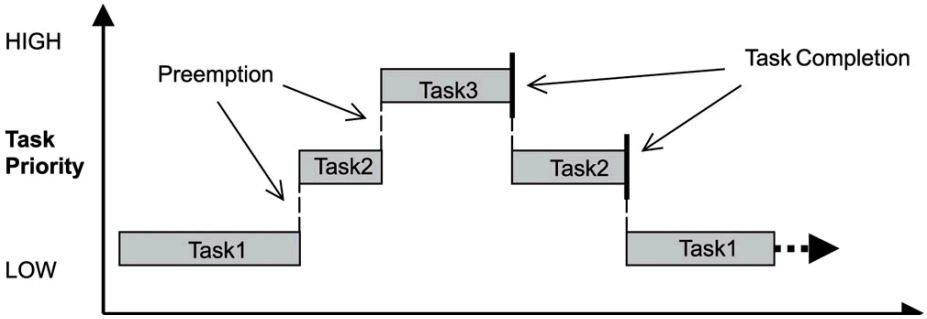
6.Scheduler (Bộ lập lịch)

- Scheduler được xem như trái tim của kernel. Một scheduler cung cấp các giải thuật cần thiết để xác định một tác vụ khi nào được phép thực hiện.
- Các giải thuật lập lịch (schedule Algorimth)
  - Preemtive priority-based scheduling
  - Round-Robin scheduling
- Các nhà sản xuất RTOS cung cấp các giải thuật lập lịch trên, tuy nhiên trong một số trường hợp các nhà phát triển hệ thống có thể định nghĩa thêm các giải thuật lập lịch.

6.1 Preemtive Priority-Based scheduling.

CuuDuongThanCong.com https://fb.com/tailieudientucntt

- Hầu hết các hệ thống nhúng sử dụng giải thuật này như là một giải thuật mặc định. Trong giải thuật này một task được thực thi ở bất kỳ vị trí nào là task có mức ưu tiên lớn nhất giữa các task khác trong hệ thống.

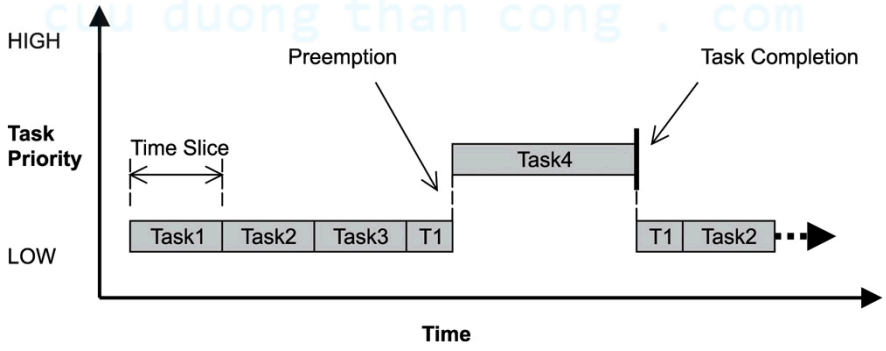


Hình 1-6 Giải thuật lập lịch Preemtive Priority-Based scheduling

- RTOS Kernel cung cấp 255 mức ưu tiên trong đó mức 0 là mức ưu tiên thấp nhất, 255 là mức ưu tiên cao nhất. Một số kernel thì định nghĩa ngược lại, 255 là mức ưu tiên thấp nhất, 0 là mức ưu tiên cao nhất.
- Với Preemtive Priority-Based scheduling, mỗi task có một mức ưu tiên khác nhau, task có mức ưu tiên cao nhất sẽ thi hành trước. Nếu một task có mức ưu tiên cao hơn một task đang thi hành mà task này ở trạng thái ready to run (yêu cầu được thực thi) thì kernel lập tức lưu task hiện hành vào TCB (task control block) và chuyển sang thực hiện task có độ ưu tiên cao hơn.
- Mặc dù mỗi task khi được tạo ra sẽ được gán một giá trị ưu tiên, xong giá trị ưu tiên có thể được thay đổi thông qua việc sử dụng các lệnh do kernel hỗ trợ.
- Khả năng thay đổi mức ưu tiên các task động cho phép các ứng dụng trên hệ thống nhúng dễ dàng hiệu chỉnh các sự kiện xảy ra, tạo ra một hệ thống thời gian thực.

6.2 Round -Robin scheduling:

- Round-Robin scheduling cung cấp mỗi task một khoảng thời than thực hiện của CPU.
- Round-Robin scheduling không thể đáp ứng các yêu cầu của hệ thống thời gian thực bởi vì trong hệ thống thời gian thực một task có thể thực hiện ở nhiều mức độ ưu tiên khác nhau, thay vào đó preemtive priority scheduling có thể tốt hơn nếu kết hợp với Round-Robin, một giải thuật sử dụng các khoảng thời gian bằng nhau thực hiện của CPU.



Hình 1-7 Round-Robin và Preemtive priority-based scheduling

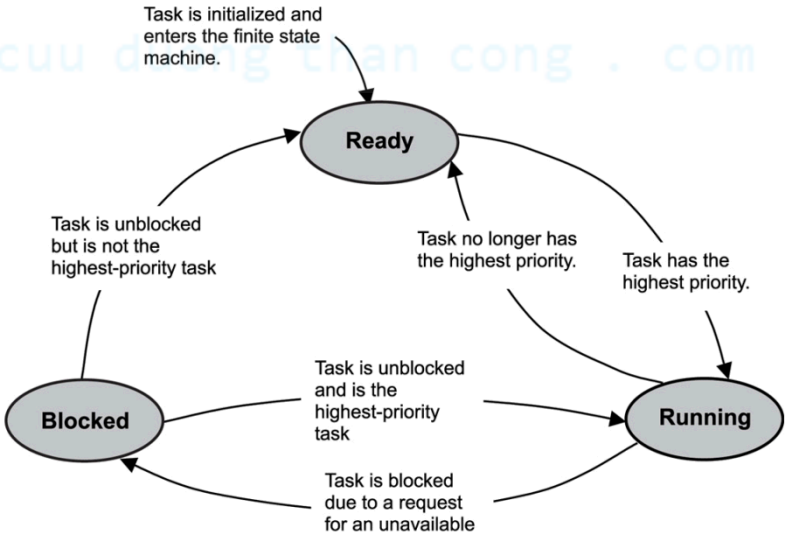
CuuDuongThanCong.com https://fb.com/tailieudientucntt

7.Task (Tác vụ):

- Một phần mềm ứng dụng đơn giản được thiết kế đặt thù hoạt động tuần tự, một lệnh được thi hành tại một thời điểm, các lệnh được thực hiện liên tiếp nhau. Mô hình này trở nên không thích hợp trong hệ thống nhúng, trong hệ thống nhúng thông thường có nhiều ngõ vào và nhiều ngõ ra, các phần mềm ứng dụng cho hệ thống nhúng phải được thiết kế để hoạt động đồng thời.
- Trong các thiết kế đồng thời đòi hỏi các nhà phát triển phải phân tích ứng dụng ra thành nhiều đơn vị chương trình nhỏ hoạt động liên tiếp nhau. Khi thực hiện phân tích xong, các thiết kế đồng thời cho phép hệ thống đa task vụ có thể hoạt động dựa trên yêu cầu chặt chẽ về thời gian cho hệ thống thời gian thực.
- Hầu hết các kernel cung cấp các task và quản lý các task để thích hợp cho các thiết kế đồng thời.
- Task là một luồng độc lập của quá trình thực hiện, các task giành nhau quá trình thực hiện của CPU. Như đề cặp ở trên các nhà phát triển chia ứng dụng thành nhiều task để tối ưu hóa các quản lý xuất nhập trong một trong các khoảng thời gian xác định.
- Một task có thể giành thời gian thực hiện của CPU theo các giải thuật lập lịch do kernel tạo ra, task được xác định dựa và các thông số và cấu trúc dữ liệu riêng biệt của nó. Mỗi task khi được tạo có một tên, số hiệu (ID) và mức ưu tiên khác nhau.

7.1 Các trạng thái của task:

- Dù là task hệ thống hay task ứng dụng, tại mỗi thời điểm, mỗi task tồn tại ở một trong các trạng thái sau: ready, running hay block. Khi hệ thống đang hoạt động các task có thể chuyển qua lại giữa các trạng thái.



cuu duong than cong . com

- **Ready state:** Task sẵn sàng thực thi nhưng không thể vì một task khác với mức ưu tiên cao hơn đang thực thi.
- **Block state:** Task gởi yêu cầu nhưng không được chấp nhận, yêu cầu được thi hành phải chờ một số sự kiện khác diễn ra hay bị trì hoãn trong một khoảng thời gian.
- **Running state:** Task có mức ưu tiên cao nhất và đang được thi hành.

CuuDuongThanCong.com https://fb.com/tailieudientucntt

Chương 2

GIỚI THIỆU HỆ ĐIỀU HÀNH NHÚNG

1. hệ điều hành linux

1.1 Giới thiệu hệ điều hành Linux:

Linux là hệ điều hành mô phỏng Unix, Linux được xây dựng dựa trên phần nhân (kernel) và các gói phần mềm mã nguồn mở. Linux được công bố dưới bản quyền của GPL (General Public Licence).

Unix ra đời giữa những năm 1960, ban đầu được phát triển bởi AT&T sau đó được đăng ký thương mại và phát triển theo nhiều dòng với các tên khác nhau. Năm 1990, xu hướng phát triển phần mềm mã nguồn mở xuất hiện và được thúc đẩy bởi tổ chức GNU. Một số Licence về mã nguồn mở xuất hiện như BSD, GPL. Năm 1991, Linus Torvald viết thêm phiên bản nhân V0.01(kernel) đầu tiên và đưa lên cho cộng đồng người dùng để sử dụng và phát triển. Năm 1996, nhân V1.0 chính thức công bố và ngày càng được sự quan tâm của người dùng. Năm 1999, phiên bản nhân V2.0 với nhiều đặc tính hỗ trợ nhiều cho các ứng dụng server. Năm 2000, phiên bản V2.4 ra đời hỗ trợ nhiều hơn và Linux bắt đầu bước chân vào thị trường mà chủ yếu là trong các ứng dụng mạng, các ứng dụng cho các thiết bị cầm tay.

Các phiên bản của Linux là sản phẩm đóng gói kernel và các gói phần mềm miễn phí khác. Các phiên bản này được công bố dưới licence GPL.

Giống như Unix, Linux gồm có 3 phần chính: Kernel, Shell và cấu trúc tập file.

Kernel là chương trình nhân, một số tài liệu còn gọi là nhân hệ điều hành Linux. Kernel chạy các chương trình hệ thống và quản lý hoạt động của hệ thống.

Shell là môi trường cung cấp các giao diện cho người sử dụng còn được mô tả như một bộ biên dịch, Shell là cầu nối giao tiếp giữa người sử dụng và nhân hệ điều hành(kernel). Shell nhận các lệnh từ người dùng và gởi các lệnh đến nhân hệ điều hành để thực thi. Hiện nay chủ yếu tồn tại 3 shell: Bourne, Korn, C Shell. Bourne được phát triển tại phòng thí nghiệm Bell. C Shell được phát triển cho phiên bản BSD của Unix. Korn Shell là phiên bản cải tiến của Bourne Shell. Những phiên bản hiện nay của Unix bao gồm Linux đều tích hợp cả 3 shell trên.

Cấu trúc File hay hệ thống file (file system) quy định cách lưu trữ các file trên đĩa. File được đặt trong các thư mục. Mỗi thư mục có thể chứa File và các thư mực con khác. Một số thư mục là các thư mục chuẩn do hệ thống sử dụng. Người dùng có thể tạo ra các file hay thư mục riêng và có thể thay đổi các file hay thư mục đó. Trong môi trương Linux/ Unix, người dùng còn có thể thay đổi các quyền truy cập trên các file hay thư mục cho phép hạn chế quyền truy cập đối với một người dùng hay một nhóm người dùng. Các thư mục trong Linux được tổ chứ hình cây, bắt đầu là thư mục gốc (root), các thư mục khác được phân nhánh từ thư mục này.

Kernel, Shell và hệ thống File tạo nên cấu trúc hệ điều hành. Với các thành phần trên người dùng có thể chạy chương trình, quản lý file và tương tác với hệ thống.

1.2 Một số phiên bản của Linux:

Redhat và Fedora Core bản Linux có lẽ là thịnh hành nhất trên thế giới, phát hành bởi công ty Redhat. Từ năm 2003, Redhat Inc chuyển hướng kinh doanh. Họ đầu tư phát triển dòng sản phẩm Redhat Interprise Linux (RHEL) với mục đích thương mại, nhắm vào các công ty, xí nghiệp. Đối với người dùng bình thường, họ mở một dự án tên là Fedora. Redhat bỏ tiền và một số kỹ sư của mình hỗ trợ cho dự án này đồng thời kêu gọi các chuyên viên thiết kế trên khắp thế giới qui tụ lại để phát triển Fedora Core. Bản Linux của Redhat cuối cùng dừng ở phiên bản 9.0. Version của Fedora Core được đếm từ 1. Có thể nghĩ đại khái là FC1 tương đương Redhat 10, FC2 tương đương Redhat 11. Thực tế thì khác nhiều, đặc biệt là từ FC2.

CuuDuongThanCong.com https://fb.com/tailieudientucntt

WhiteBox Linux Bản clone của Redhat Enterprise Linux 3.0. Build trên source code của RHEL bởi một nhóm các kỹ sư ở LA, Hoa Kỳ. Hiện nay server Nhatban.NET đang dùng bản này.

SuSE Linux được sản xuất ở Đức. Bản Linux cực kỳ thịnh hành ở châu Âu và Bắc Mỹ. Năm 2003, công ty SuSE bị ông lớn Novell mua. Novell đang dốc sức đầu tư cho SuSE để nhắm vào các nhà doanh nghiệp hòng giành lại thị phần từ tay Redhat. Bản SuSE mới nhất hiện nay là 9.1

Mandrake Linux Made in France. Cũng là một bản Linux rất thịnh hành ở châu Âu, Mỹ, và Việt Nam. Đây cũng là bản được ưu ái nhất trong vấn đề Việt hoá. Theo thông tin mới nhất ngày 22/7/2004 thì quá trình Việt hoá cho Mandrake Linux (MDK) đã đạt 85%. Bản MDK mới nhất hiện nay là 10.0

Turbo Linux Nổi tiếng ở Nhật, Trung Quốc. Công ty Turbo đang đầu tư mạnh nhằm thống trị thị trường Linux Trung Quốc. Bản Turbo mới nhất hiện nay là 10F.

Debian Linuxm, một ông lớn nữa trong làng Linux. Nhiều người có ý kiến cho rằng: "người không chuyên nên dùng Fedora Core để có thể làm quen được với những kỹ thuật mới nhất của Linux, còn dân chuyên nghiệp nên dùng Debian vì sự ổn định tuyệt vời của nó". Bản mới nhất: 3.0R2

Vine Linux Cực kỳ được ưa chuộng tại Nhật. Được phát triển trên nền Redhat 6.2. Đặc điểm của bản này là rất nhẹ (duy nhất 1 đĩa CD) và hỗ trợ tiếng Nhật 100%. Vine Linux cũng được tích hợp thêm một số tính năng của Debian ví dụ như apt-get. Bản mới nhất hiện nay là 2.6R4. Bản 3.0 được tung ra trong tháng 8/2004.

Knoppix Linux được sản xuất ở Đức. Bản live Linux được ưa chuộng nhất hiện nay. Khởi động trực tiếp từ CD mà không cần cài đặt vào ổ cứng. Phiên bản mới nhất là 3.4.

Vietkey Linux được sản xuất tại Việt Nam. Hoàn toàn không có tiếng tăm gì ngoài chuyện được giải trong cuộc thi TTVN 2003. Phát triển bởi nhóm Vietkey trên nền Redhat 7.2. Cũng nên thử cho biết sản phẩm đoạt giải nhất của TTVN nó ra sao.

vnlinuxCD Bản live CD by Larry Nguyễn. Nguyên tắc của vnlinuxCD giống Knoppix nhưng được build trên nền Mandrake 9.2. Hỗ trợ khá tốt các vấn đề về tiếng việt.

Các phiên bản khác còn rất nhiều nhà phân phối khác. Các bạn tự tìm hiểu thêm có thể check: Slackware, Gentoo, College, Yellow Dog, SGI, Momonga...

1.3 Hệ thống tập tin và thư mực trên Linux:

Trong môi trường Windows (ví dụ 2000 hay XP), mặc dù người dùng có toàn quyền tổ chức cấu trúc thư mục, nhưng một số quy định truyền thống vẫn được tuân theo. Ví dụ các tập tin hệ thống thường nằm trong thư mục :\Windows, các chương trình thường được cài đặt vào C:\Program Files, v.v. . . Trong Linux cũng có một cấu trúc thư mục kiểu như vậy và thậm chí còn nghiêm ngặt hơn. Hơn nữa có một tiêu chuẩn xác định cấu trúc thư mục cho các hệ điều hành dòng UNIX. Tiêu chuẩn này được gọi là Filesystem Hierarchy Standart (FHS).

/bin: Thư mục này gồm chủ yếu các chương trình, phần lớn trong số chúng cần cho hệ thống trong thời gian khởi động (hoặc trong chế độ một người dùng khi bảo trì hệ thống). Ở đây có lưu rất nhiều những câu lệnh thường dùng của Linux.

/boot: Gồm các tập tin cố định cần cho khởi động hệ thống, trong đó có nhân (kernel). Tập tin trong thư mục này chỉ cần trong thời gian khởi động.

/dev: Thư mục này chứa các file thiết bị. Trong thế giới Unix và Linux các thiết bị phần cứng được xem như làm một file. Đĩa cứng và phân vùng là các file như hda1, hda2. Đĩa mềm là fd0… Các tập tin thiết bị này đặt trong thư mục /dev.

/etc: Thư mục này chứa các file cấu hình toàn cục của hệ thống. Có thể có nhiều thư mục con của thư mục này nhưng nhìn chung chúng chứa các file **script** để khởi động hay phục vụ cho mục đích cấu hình chương trình trước khi khởi động.

CuuDuongThanCong.com https://fb.com/tailieudientucntt

/home: Thư mục này chứa các thư mục con đại diện cho mỗi User khi đăng nhập. Nơi đây tựa như ngôi nhà của người dùng. Khi người quản trị tạo tài khoảng cho người dùng, họ cấp cho người dùng một thư mục con trong **/home**. Người sử dụng có thể sao chép, xóa file, tạo thư mục con trong thư mục **/home** mà không ảnh hưởng đến các người dùng khác.

/lib: thư mục này chứa các file thư viện **.so** hoặc **.a**. Các thư viện C và các thư viện liên lết động cần cho chương trình khi chạy và cần cho toàn hệ thống. Thư mục này tương tự như thư mục SYSTEM32 của Windows.

/lost + found: Thư mục này được đặt tên hơi lạ nhưng đúng nghĩa của nó. Khi hệ thống khởi động hoặc khi chạy chương trình **fsck**, nếu tìm thấy một chuỗi dữ liệu nào đó bị thất lạc trên đĩa cứng không liên quan đến các tập tin, Linux sẽ gộp chúng lại và đặt trong thư mục này để nếu cần người dùng có thể đọc và giữ lại dữ liệu đã mất.

/mnt: Thư mục này chứa các thư mục kết gán tạm thời đến các ổ đĩa hay thiết bị khác.

/sbin: Thư mục này chứa các file hay chương trình thực thi của hệ thống thường chỉ cho phép sử dụng bởi người quản trị.

/tmp: Đây là thư mục tạm dùng để chứa các file tạm mà chương trình sử dụng chỉ trong quá trình chạy. Các file trong thư mục này sẽ được hệ thống dọn dẹp nếu không còn dùng đến nữa.

/usr: thư mục này chứa rất nhiều thư mục con như **/usr/bin** hay **/usr/sbin**. Một trong những thư mục quan trọng trong **/usr** là **/usr/local**. Bên trong thư mục local này bạn có đủ các thư mục con tương tự ngoài thư mục gốc như **sbin,lib, bin**… Nếu bạn nâng cấp hệ thống thì các chương trình cài đặt trong **/usr/local** vẫn giữ nguyên và không sợ bị mất mát. Hầu hết các ứng dụng Linux đều thích cài đặt vào **/usr/local**. Thư mục này tương tự như Program File trên Windows.

/var: Thư mục này chứa các file biến thiên bất thường như các file dữ liệu đột nhiên tăng kích thước trong một thời gian ngắn sau đó lại giảm kích thước xuống còn rất nhỏ. Điển hình là các file dùng làm hàng đợi chứa dữ liệu cần đưa ra máy in hoặc các hàng đợi chứa mail.

1.4 Các lệnh cơ bản trên Linux:

- **Hiển thị thông tin người dùng:**

Lệnh who am I, hay whoami

```
[root@localhost ~]# who
root     tty7         2010-03-11 03:33 (:0)
root     pts/1        2010-03-11 03:33 (:0.0)
root     pts/2        2010-03-11 03:34 (:0.0)
root     pts/3        2010-03-12 02:39 (:0.0)
root     pts/4        2010-03-13 01:45 (192.168.1.2)
[root@localhost ~]# who am i
root     pts/4        2010-03-13 01:45 (192.168.1.2)
[root@localhost ~]# whoami
root
[root@localhost ~]#
```

- **Tạo tài khoản người:**

Lệnh **useradd**

useradd [tên tài khoản]

Lệnh cho phép tạo một tài khoản người dùng, người dùng có thể sử dụng tài khoản này để đăng nhập vào hệ thống ngay trên máy Linux hoặc từ các máy khác thông qua mạng.

Bạn chỉ khi đăng nhập vào hệ thống với tài khoản root bạn mới có thể tạo tài khoản người dùng, đăng nhập hệ thống với tài khoản **root** có thể đăng nhập trực tiếp trên máy hoặc thông qua một máy khác trên mạng.

CuuDuongThanCong.com https://fb.com/tailieudientucntt

Sau khi tạo tài khoản thành công, thư mục có tên tài khoản vừa tạo sẽ được tạo ra trong thư mục **/home /.**

- **Xóa tài khoản người dung:**

  Lệnh **userdel**

  userdel [tên tài khoản]

- **Thay đổi mật khẩu đăng nhập:**

  Lệnh **passwd**

```
[root@localhost ~]# passwd
Changing password for user root.
New UNIX password:
BAD PASSWORD: it is too simplistic/systematic
Retype new UNIX password:
passwd: all authentication tokens updated successfully.
```

Bạn chú ý trong hệ thống Linux, khi bạn nhập password thì các ký tự sẽ không được hiển thị ra màn hình dưới dạng các ký tự * như trong hệ thống windows.

- **Thay đổi tài khoản đăng nhập:**

  Lệnh **su**

  Giả sử bạn đăng nhập với tài khoản là root. Sau khi đăng nhập dấu nhắc hệ thống sẽ có dạng #.

  Giờ chúng ta tạo thêm một tài khoản user có tên là **user01** sử dụng lệnh **useradd** như sau:

  useradd user01

  Chuyển đăng nhập sang **user01** như sau:

  su user01

  Lúc này dấu nhắc hệ thống có dạng $

```
[root@localhost /]#
[root@localhost /]#
[root@localhost /]# useradd user01
[root@localhost /]# su user01
[user01@localhost /]$
```

  Thay đổi đăng nhập bằng tài khoảng root:

```
[user01@localhost /]$ su root
Password:
[root@localhost /]#
```

- **Xem trợ giúp về lệnh:**

  Lệnh **man**

  Cú pháp: **man tên lệnh**

  Lệnh man cho phép hệ thống hiển thị thông tin của lệnh được chỉ ra trong lệnh man, đây là một trình trợ giúp hiệu quả cho người sử dụng Linux.

  Một số phím chức năng trong lệnh man:

  :q Kết thúc

  :b Về trang trước

  :f Về trang sau

- **Thay đổi thư mục hiện hành:**

  Lệnh **cd** (change directory)

CuuDuongThanCong.com https://fb.com/tailieudientucntt

$ cd <pathname>

pathname = đường dẫn tương đối (tính từ thư mục hiện hành) hoặc tuyệt đối (tính từ thư mục gốc)

Thư mục đặc biệt:

Thư mục hiện hành: **.**

Thư mục cha: **..**

Thư mục home: ~ hoặc ***~username***

$cd /tmp (chuyển tới thư mục /tmp)

$cd ../home/a01 (chuyển tới thư mục /home/a01)

$pwd

/home/a01

- **Liệt kê nội dung thư mục:**

Lệnh **ls** (listing directory):

ls [option] path_name

Ví dụ:

$ ls

addr.c env.c fork.c lockf.c pipe1.c a.out exec1.c forkex.c lockf.h

-a liệt kê các file ẩn

-d chỉ liệt kê tên của thư mục, không liệt kê nội dung

-F liệt kê các file và cho biết kiểu của file qua ký hiệu ở cuối

Không có ký hiệu gì:	file thường
'/'	directories
'*'	executable files
"@"	linked files

-i cho biết số inode của file

-l liệt kê đầy đủ thông tin về file/thư mục

-R liệt kê các thư mục con đệ quy

-t sắp xếp theo thời gian cập nhật

- **Tạo thư mục:**

Dùng lệnh **mkdir**

mkdir *path_name*

Ví dụ:

$pwd

/export/home/a01

$mkdir examples

$ls –aF

./ .bash_logout .bashrc .emacs ex.tar .screenrc

../ .bash_profile Desktop/ examples/ .kde/ .wl

Ví dụ cần tạo 3 thư mục a, b, c như sau a/b/c

Dùng 3 lệnh mkdir

$mkdir a

$mkdir a/b

CuuDuongThanCong.com https://fb.com/tailieudientucntt

$mkdir a/b/c

Dùng một lệnh mkdir

$mkdir –p a/b/c

1.5 Lệnh xóa file và thư mục:

Lệnh **rm**

Xoá thư mục rỗng (không chứa thư mục con hay file)

rmdir *path_name(s)*

Xoá thư mục không rỗng

rm –r *path_name(s)*

Xoá file

rm –option *file_name(s)*

- **Lệnh Copy files:**

  cp [-option] from(s) to

  Copy thư mục

  cp -r from(s) to

Vídụ:

$cp /etc/passwd.

$cp p*.pas /tmp

$cp /etc/sysconfig/network-sripts /tmp

1.6 Di chuyển file và thư mục

Lệnh **mv** (move):

mv [option] *filename dest_file*

mv [option] *directory dest_dir*

mv [option] *filename dest_dir*

Ví dụ:

$mv examples lab1

- **Tạo file và nhập vào nội dung**

  cat > name_of_file

Sau khi nhập nội dung, gõ <Enter> để xuống dòng.

Ấn **Ctrl-d** để ghi nội dung soạn thảo vào file và kết thúc thao tác.

Ví dụ

$cat > test.txt <Enter>

this is my file <Enter>

Ctrl + D

$

Tạo file rỗng (0 bytes) bằng lệnh **touch**

touch new_file

- **Tạo file có nội dung dài:**

Lệnh **more:**

more filename

Dấu nhắc --More--(nn%) xuất hiện bên dưới màn hình.

Có thể dùng các phím điều khiển trong lúc đang xem nội dung file

CuuDuongThanCong.com https://fb.com/tailieudientucntt

space bar	hiển thị trang kế tiếp
<RETURN>	hiển thị dòng kế tiếp
q	thoát khỏi lệnh more
b	về trang trước.
h	xem trợ giúp

Hiển thị ***n*** dòng đầu tiên của một text file, dùng lệnh ***head***

head -n filename

(nếu n=10, có thể bỏ option –n đi: ***head filename***)

- **Hiển thị nội dung file:**

Hiển thị ***n*** dòng sau cùng của một text file, dùng lệnh ***last***

last -n filename

(nếu n=10, có thể bỏ option –n đi: ***last filename***)

1.7 Tìm kiếm một file trong hệ thống file (file system): dùng lệnh *find*

find pathname -name filename -print

(Có thể dùng wildcard đặt trong dấu nháy kép)

Ví dụ:

$find / -name "*.cpp" -print

Cũng có thể định vị một file bằng các lệnh which, whereis, locate (lưu ý là các lệnh này chỉ tìm trong phạm vi biến môi trường PATH hoặc xxxPATH)

Ví dụ:

$ which find

$ locate ls

- **Tìm trong nội dung của file:**

Tìm một chuỗi ký tự trong một text file bằng lệnh

grep *pattern filename(s)*

pattern: chuỗi ký tự cần tìm kiếm. Nếu chuỗi có ký tự đặc biệt thì phải đặt trong dấu nháy đơn.

Ví dụ:

$ grep UNIX /usr/man/man*/*

$ grep -n '[dD]on\'t' notes

$ grep a01 /etc/passwd

- **Các quyền trên file và thư mục:**

Hệ thống *NIX bảo vệ các file và thư mục thông qua các quyền thiết lập trên đó.

Có 3 quyền:

r–read - đọc

w–write –ghi

x–execute -thực thi

Các quyền được áp dụng trên 3 nhóm người dùng, kí hiệu bằng ba kí tự tương ứng u, g, o

u = owner user = chủ sở hữu

g = group = những người cùng nhóm với chủ sở hữu

o = others = tất cả những người khác

- **Phân quyền:**

CuuDuongThanCong.com https://fb.com/tailieudientucntt

Các quyền áp dụng cho 3 nhóm người dùng kết hợp lại thành 9 bit như sau:

rwx rwx rwx

user group other

Có thể xem thông tin về quyền truy cập bằng lệnh **ls -l**

Ví dụ:

$ls -l

-rwxr-xr-x

Với ví dụ trên:

Chủ sở hữu có quyền **r** (đọc), **w** (ghi), và **x** (thực thi).

Các thành viên cùng nhóm với chủ sở hữu có quyền **r** và **x**.

Những người khác có quyền **r** và **x**.

- **Thay đổi quyền trên file và thư mục:**

Dùng lệnh **chmod.**

chmod access_mode file(s)

Quyền truy cập có thể thiết lập theo 2 dạng

Dạng dùng ký hiệu (symbolic): **[ugo][+ -=][rwx]**

Dạng dùng số bát phân (octal): **[0-7][0-7][0-7]**

1.8 Kết gán ổ đĩa và thư mục:

Lệnh mount

Lệnh **mount** cho phép kết gán các phân vùng hay thiết bị vật lý như A, CD_ROM thành một thư mục trong cây thư mục thống nhất của hệ điều hành bắt đầu từ thư mục gốc /. Lệnh mount đơn giản có cú pháp như sau:

mount –t vfstype devicefile mdir

devicefile là đường dẫn đến file thiết bị (thường lưu trong thu mực **/dev**). Linux thường quy định ổ đĩa A là file thiết bị **/dev/fd0**. Ổ đĩa CD-ROM là **/dev/cdrom**, các phân vùng là **dev/hda1**, **dev/hda2**….Tùy chọn **–t** sẽ kết gán theo kiểu hệ thống file trên thiết bị do vfstype quy định. mdir là đường dẫn cần kết gán vào hệ thống file của Linux. Hiện tại Linux có thể đọc được rất nhiều hệ thống file, **vfstype** có thể bao gồm những kiểu hệ thống file thông dụng sau:

Msdos là hệ thống file và thư mục theo bảng FAT16, FAT32 của DOS. Linux đọc được mọi kiểu đĩa và định dạng của DOS / Windows.

Ntfs Định dạng hệ thống file NTFS của Windows NT

Ext2 Định dạng hệ thống file chuẩn của Unix và Linux.

Nfs Định dạng hệ thống file truy xuất qua mạng (Network File System)

Muốn tháo kết gán có thể sử dụng lệnh **umount.** Lệnh umount chỉ yêu cầu tham số là đường dẫn đến thư mục đang kết gán. Sau khi tháo kết gán bạn không còn truy xuất vào thiết bị được nữa.

Ví dụ kết gán ổ đĩa A vào thư mục trong **/tmp**

mount – t msdos /dev/fd0 /mnt/mydrive

Đọc ghi thư mục mydrive tương ứng với đọc ghi ổ đĩa a của hệ thống

Tháo kết gán ổ đĩa A

umount /mnt/mydrive

- **Đóng gói các tập tin:**

CuuDuongThanCong.com https://fb.com/tailieudientucntt

Để đóng gói các file của chương trình, ta thường nén chúng lại thành một file duy nhất với các dạng nén như **.tar, .gz, .tgz**. Thường file TAR (tape archive) trước đây là một file dạng lưu trữ của UNIX nên tỉ lệ nén không cao. Bạn nén các file lại thành file .tar sử dụng lệnh **tar**. Sau này thuật giải nén zip cho phép nén nhiều dữ liệu hơn nên các file .tar có thể được nén thêm một lần nữa bằng trình gzip (trên Windows là winzip).

Ví dụ:

Giả sử trong thư mục hiện hành có các file như sau: **main.c, a.c, a.h, b.c b.h**, **Makefile**. Giờ ta nén các file lại và lưu trong một file nén có tên **myapp.tar**

tar cvf mayapp.tar main.c a.c a.h b.c b.h

```
[root@localhost sonl]# ls
a.c  a.h  b.c  b.h  main.c
[root@localhost sonl]# tar cvf myapp.tar main.c a.c a.h b.c b.h
main.c
a.c
a.h
b.c
b.h
[root@localhost sonl]# ls
a.c  a.h  b.c  b.h  main.c  myapp.tar
[root@localhost sonl]#
```

Kết quả thu được tập tin nén myapp.tar trong cùng một thư mục.

Trình gzip có thể cho kích thước file nhỏ hơn như sau:

gzip myapp.tar

```
[root@localhost sonl]# gzip myapp.tar
[root@localhost sonl]# ls
a.c  a.h  b.c  b.h  main.c  myapp.tar.gz
[root@localhost sonl]#
```

Kết quả ta thu được tập tin nén myapp.tar.gz

Khi nhận được tập tin nén **myapp.tar.gz.** quá trình giải nén ngược lại có thể được tiến hành như sau:

- Giải nén trở lại tập tin **.tar**

  gzip –d myapp.tar.gz
- Giải nén tập tin **.tar**

  tar xvf myapp.tar

  Cú pháp và tùy chọn của lệnh tar thường dùng:

  Tar [option] [list of file]

  Trong đó option sẽ mang các giá trị kết hợp sau:

  c Tạo file tar mới

  f tên tập tin cần đưa dữ liệu vào

  t Liệt kê nội dung hay danh sách file chứa trong file tar

  v yêu cầu lệnh tar hiển thị thông báo khi thực thi lệnh

  x Bung các file trong tập tin tar trở lại đĩa cứng.

Thông thường khi nén các tập tin tạo thành file **tar** ta dùng tùy chọn **cvf**. Ngược lại khi giải nén tập tin **.tar** ta dùng tùy chọn **xvf**.

CuuDuongThanCong.com https://fb.com/tailieudientucntt

Chương 3
LẬP TRÌNH HỆ VỎ SHELL
1. Sử dụng biến

Thường bạn không cần phải khai báo biến trước khi sử dụng. Thay vào đó biến sẽ được tự động tạo và khai báo khi lần đầu tiên tên biến xuất hiện, chẳng hạn như trong phép gán. Mặc định, tất cả các biến đều được khởi tạo và chứa trị kiểu chuỗi (string). Ngay cả khi dữ liệu mà bạn đưa vào biến là một con số thì nó cũng được xem là định dạng chuỗi. Shell và một vài lệnh tiện ích sẽ tự động chuyển chuỗi thành số để thực hiện phép tính khi có yêu cầu. Tương tự như bản thân hệ điều hành và ngôn ngữ C, cú pháp của shell phân biệt chữ hoa chữ thường, biến mang tên foo, Foo, và FOO là ba biến khác nhau.

Bên trong các script của shell, bạn có thể lấy về nội dung của biến bằng cách dùng dấu $ đặt trước tên biến. Để hiển thị nội dung biến, bạn có thể dùng lệnh **echo**. Khi gán nội dung cho biến, bạn không cần phải sứ dụng ký tự $. Ví dụ trên dòng lệnh, bạn có thể gán nội dung và hiển thị biến như sau:

```
$ xinahao=hello
$ echo $xinchao
Hello
$ xin chao= "I am here"
$echo $xin chao
I am here
$ xinchao=12+1
$echo $xin chao
12+1
```

Có thể sử dụng lệnh **read** để đọc nhập liệu do người dùng đưa vào và giữ lại trong biến để sử dụng. Ví dụ:

```
$ read yourname
XYZ
$echo "Hello " $yourname
Hello XYZ
```

Lệnh **read** kết thúc khi bạn nhấn phím Enter (tương tự scanf của C hay readln của Pascal).

2. Các ký tự đặc biệt

Một tiến trình Unix/Linux bao giờ cũng gắn liền với các đầu xử lí các dòng (stream) dữ liệu: đầu vào chuẩn (stdin hay 0), thường là từ bàn phím qua chức năng getty(); đầu ra chuẩn (stdout, hay 1), thường là màn hình, và cơ sở dữ liệu lỗi hệ thống (stderr, hay 2). Tuy nhiên các hướng vào/ra có thể thay đổi được bởi các thông báo đặc biệt:

Kí hiệu Ý nghĩa (… tượng trưng cho đích đổi hướbg)

CuuDuongThanCong.com https://fb.com/tailieudientucntt

>	Đầu ra hướng tới …
>>	Nối vào nội dung của …
<	Lấy đầu vào từ < …
<< word	đầu vào là ở đây …
2>	đầu ra báo lỗi sẽ hướng vào …
2>>	đầu ra báo lỗi hướng và ghi thêm vào …

Ví dụ:

$date > login.time

Lệnh date không kết xuất ra đầu ra chuẩn (stdout) mà ghi vào tệp login.time. *>login.time* không phải là thành phần của lệnh date, mà đơn giản mô tả tiến trình tạo và gởi kết xuất ở đâu (bình thường là màn hình). Nhìn theo cách xử lí thì như sau: cả cụm lệnh trên chứa hai phần: lệnh *date*, tức chương trình thực thi, và thông điệp *(>login.time)* thông báo cho shell biết kết xuất lệnh sẽ được xử lí như thế nào (khác với mặc định. Bản thân date cũng không biết chuyển kết xuất đi đâu, shell chọn mặc định).

Ví dụ:

$cat < file1

Bình thường cat nhận và hiển thị nội dung tệp có tên (là đối đầu vào). Với lệnh trên cat nhận nội dung từ file1 và kết xuất ra màn hình. Thực chất không khác gì khi gõ:

$cat file1.

cuu duong than cong . com

Hãy xem:

$cat < file1 > file2

Lệnh này thực hiện như thế nào ? Theo trình tự sẽ như sau: cat nhận nội dung của file1 sau đó ghi vào tệp có tên file2, không đưa ra stdout như mặc định. Lệnh cho thấy ta có thể thay đổi đầu và đầu ra cho lệnh như thế nào. Những lệnh cho phép đổi đầu ra/vào gọi chung là qúa trình lọc (filter).

3. Ống dẫn

Shell cho phép kết quả thực thi một lệnh (đầu ra của lệnh), kết hợp trực tiếp (nối vào) đầu vào của một lệnh khác, mà không cần xử lí trung gian (lưu lại trước tại tệp trung gian).

Ví dụ:

$who | ls –l

Đầu ra (stdout) của *who* (đáng lẽ sẽ ra màn hình), sẽlà đầi vào (stdin) của *ls –l*.

Ví dụ:

$ (date ; who) | ls -

ng than cong . com

Tóm tắt:

cmd &	*đặt lệnh cmd chạy nền (background)*
cmd1 ; cmd2	*chạy cmd1 trước, sau đó chạy cmd2*
(cmd)	*thực hiện cmd trong một shell con (subshell)*
`cmd`	*đầu ra của cmd sẽ thay cho đàu ra của lệnh trong dòng lệnh*
cmd1 | cmd2	*nối đầu ra của cmd1 vào đầu vào của cmd2*

4. Lệnh rẽ nhánh

CuuDuongThanCong.com https://fb.com/tailieudientucntt

Nền tảng cơ bản trong tất cả ngôn ngữ lập trình, đó là khả năng kiểm tra điều kiện và đưa ra quyết định rẽ nhánh thích hợp tùy theo điều kiện đúng hay sai. Trước khi tìm hiểu cấu trúc điều khiển của ngôn ngữ script, ta hãy xem qua cách kiểm tra điề kiện.

Một script của shell có thể kiểm tra mã lỗi trả về của bất kỳ lệnh nào có khả năng triệu gọi từ dòng lệnh, bao gồm ả những tập tin lệnh script khác. ĐÓ là lý do tại sao chúng ta thường sử dụng lệnh exit ở cuối mỗi scipt khi kết thúc.

Thực tế, các script sử dụng lệnh [] hoặc test để kiểm tra điều kiện boolean rất thường xuyên. Trong hầu hết các hệ thống UNIX và Linux thì [ ] và test có ý nghĩa tương tự nhau, thường lệnh [ ] được dùng nhiều hơn. Lệnh [ ] trông đơn giản, dễ hiểu và rất gần với các ngữ lập trình khác.

Trong một số shell của Unix, lệnh test có khả năng là một lời triệu gọi đến chương trình bên ngoài chứ không phải lệnh nội tại của ngôn ngữ script. Bởi vì test ít khi được dùng và hầu hết các lập trình viên có thói quen thường tạo các chương trình với tên test, cho nên khi thử lệnh test không thành công bên trong script, thì hãy xem lại đây đó bên trong hệ thống có một chương trình tên là test khác biệt nào đó đang tồn tại. Hãy thử dùng lệnh ***which test****, lệnh này sẽ trả về cho bạn đường dẫn đến thư mục test được triệu gọi. Chẳng hạn /bin/test hay /usr/bin/test.*

Dước đây là cách sử dụng lệnh test đơn giản nhất. Dùng lệnh test để kiểm tra xem file mang tên hello.c có tồn tại trong hệ thống hay không. Lệnh test trong trường hợp này có cú pháp như sau: *test -f <mename>*, trong script ta có thể viết lệnh theo cách sau:

```
if test -f hello.c
then
        . . .
fi

Cũng có thể sử dụng [ ] để thay thế test
if  [-f hello.c ]
then
        . . .
fi
```

cuu duong than cong . com

CuuDuongThanCong.com https://fb.com/tailieudientucntt

Mà lỗi và giá trị trả về của lệnh mà test kiểm tra sẽ quyết định điều kiện kiểm tra là đúng hay sai.
Lưu ý, phải đặt khoảng trắng giữa lệnh [ ] và biểu thức kiểm tra. Để dễ nhớ thể xem [ ] tương đương với lệnh test, và dĩ nhiên giữa một lệnh và tham số truyền cho lệnh phải phân cách nhau bằng khoảng trắng để trình biên dịch có thể hiểu.

Nếu thích đặt từ khóa **then** chung một dòng với lệnh **if**, bạn phải phân cách **then** bằng dấu chấm phẩy (;) như sau:

```
if [ -f hello.c ] ; then
. . .
fi
```

Điều kiện mà lệnh test cho phép kiểm tra có thể rơi vào một trong 3 kiểu sau:

So sánh chuỗi

So sánh	Kết quả
stringl = string2	*true* nếu 2 chuỗi bàng nhau (chính xác từng ký tự)
tring1 != string2	*true* nếu 2 chuỗi không bằng nhau
-n string1	*true* nếu string1 không rỗng
-z stringl	*true* nếu string1 rỗng (chuỗi null)

So sánh toán học

So sánh	Kết quả
expression1 -eq expression2	*true* nếu hai biểu thức bằng nhau
expression1 -ne expression2	*true* nếu hai biểu thức không bằng nhau
expression1 -gt expression2	*true* nếu biểu thức expression1 lớn hơn expression2
expreesion1 -ge expression2	*true* nếu biểu thức expression1 lớn hơn hay bằng expression2
expression1 -lt expression2	*true* nếu biểu thức expression1 nhỏ hơn expression2
expression1 -le exprebbion2	*true* nếu biểu thức expression1 nhỏ hơn hay bằng expression2
!expression	*true* nếu biểu thức expression là *false* (toán tử *not*)

Kiểm tra điều kiện trên tập tin

-d file	*true* nếu file là thư mục
-e file	*true* nếu file tồn tại trên đĩa
-f file	*true* nếu file là tập tin thông thường
-g file	*true* nếu set-group-id được thiết lập trên file
-r file	*true* nếu file cho phép đọc
-s f ile	*true* nếu kích thước file khác 0
-u file	*true* nếu set-ser-id được áp đặt trên file
-w file	*true* nếu file cho phép ghi
-x file	*true* nếu file được phép thực thi

Lưu ý về mặt lịch sử thì tùy chọn -e không khả chuyển (portable) và -f thường được sử dụng thay thế.

4.1 Lệnh if

Lệnh **if** tuy đơn giản nhưng được sử dụng nhiều nhất. **if** kiểm tra điều kiện đúng hoặc sai để thực thi biểu thức thích hợp

CuuDuongThanCong.com https://fb.com/tailieudientucntt

if ***condition***
then
statements
else
statements

Ví dụ, đoạn script sau sử đụng if tùy vào cấu trả lời của bạn mà đưa ra lời chào thích hợp

```
#!/bin/sh
echo nhap a=
read a
echo nhap b=
read b
if [ $a -gt $b ] ; then echo 'a>b‘
elif [ $a -eq $b ]   ; then
echo 'a=b‘
else echo 'a<b‘
fi
```

cuu duong than cong . com

5. Vòng lặp

Sử dụng **for** để lặp lại một số lần với các giá trị xác định. Phạm vi lặp có thể nằm trong một tập hợp chuỗi chỉ định tường minh bởi chương trình hay là kết qủa trả về từ một biến hoặc biểu thức khác.

Cú pháp:

for *variable* **in** *values*
do
statemens
done

```
#!/bin/sh
dem=0
echo "ban nhap vao mot chuoi"
read word
echo "tach tu:"
for i in $word
do
    dem="$(($dem+1))"
```

cuu duong than cong . com

CuuDuongThanCong.com https://fb.com/tailieudientucntt

```
        echo "$i"
done
echo "tong so tu la:$dem"
exit 0
```

Mặc dù lệnh **for** cho phép lặp trong một tập hợp giá trị biết trước, nhưng trong trường hợp một tập hợp lớn hoặc số lần lặp không biết trước, thì **for** không thích hợp. Ví dụ .

```
for foo in 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16
do
echo $foo
done
```

Lệnh **while** cho phép thực hiện lặp vô hạn khi điều kiện kiểm tra vẫn còn đúng.

Cú pháp của while như sau:

while *condition* **do**
    statements
done

Ví dụ sau sẽ cho thấy cách while liên tục kiểm tra mật khẩu (password) của người dùng cho đến khi đúng bằng chuỗi secret, mới chấp nhận.

```
#!/bin/sh

echo "Enter password"
read trythis

while [ "$trythis" != "secret" ]; do
  echo "Sorry, try again"
  read trythis
done

exit 0
```

cuu duong than cong . com

6. Lệnh CASE

CuuDuongThanCong.com https://fb.com/tailieudientucntt

Lệnh **case** có cách sử dụng hơi phức tạp hơn các lệnh đã học. Cú pháp của lệnh **case** như sau:

case variable **in**
pattern [| partten] . . .) *statements;;*
pattern [| partten] . . .) *statements;;*
. . .
esac

Mặc dù mới nhìn hơi khó hiểu, nhưng lệnh **case** rất linh động. **case** cho phép thực hiện so khớp nội dung của biến với một chuỗi mẫu *pattern* nào đó. Khi một mẫu được so khớp, thì (lệnh) *statement* tương ứng sẽ được thực hiện. Hãy lưu ý đặt hai dấu chấm nhảy **;;** phía sau mỗi mệnh đề so khớp *pattern*, shell dùng dấu hiệu này để nhận dạng mẫu *pattern* so khớp tiếp theo mà biến cần thực hiện.

```
#!/bin/sh
echo "Is it morning? Please answer yes or no"
read timeofday
case "$timeofday" in
"yes") echo "Good Morning";;
"no" ) echo "Good Afternoon";;
"y" ) echo "Good Morning";;
"n" ) echo "Good Afternoon";;
* ) echo "Sorry, answer not recognised";;
esac
exit 0
```

cuu duong than cong . com

cuu duong than cong . com

CuuDuongThanCong.com https://fb.com/tailieudientucntt

7. Lệnh số học

Danh shell AND cho phép thực thi một chuỗi lạnh kề nhau, lệnh sau chỉ thực hiện khi lệnh trước đã thực thi và trả về mã lỗi thành công. Cú pháp sử dụng như sau:

Statement1 && statement2 && statement3 && . . .

Bắt đầu từ bên trái *statement1* sẽ thực hiện trước, nếu trả về *true* thì *statement2* tiếp tục được gọi. Nếu *statement2* trả về *false* thì shell chấm dứt danh shell AND ngược lại *statement3* sẽ được gọi ... Toán tự **&&** dùng để kiểm tra kết qủa trả về của statement trước đó.

Kết quả trả về của AND sẽ là *true* nếu tất cả các lệnh *statement* đều được gọi thực thi. Ngược lại là *false*.

Hãy xét ví dụ sau, dùng lệnh touch file_one (để kiểm tra file_one tồn tại hay chưa, nếu chưa thì tạo mới) tiếp đến rm file_two. Sau cùng danh shell AND sẽ kiểm tra xem các file có đồng thời tồn tại hay không để đưa ra thông báo thích hợp.

```
#!/bin/sh

touch file_one
rm -f file_two

if [ -f file_one ] && echo "hello" && [ -f file_two ] && echo
"there"
then
    echo -e "in if"
else
    echo -e "in else"
fi

exit 0
```

Danh shell OR cũng tương tự với AND là thực thi một dãy các lệnh, nhưng nếu có một lệnh trả về true thì việc thực thi ngừng lại. Cú pháp như sau:

statementl || statement2 || statement3 && . . .

Bắt đầu từ bên trái, *statementl* được gọi thực hiện, nếu *statement1* trả về *false* thì *statement2* được gọi, nếu *statement2* trả về *true* thi biểu thức lệnh chấm đứt, ngược lại *statement3* được gọi. Kết qủa sau cùng của danh shell OR chỉ đúng (true) khi có một trong các *statement* trả về *true*. Nếu **&&** gọi lệnh tiếp theo khi các lệnh trước đó *true*, thì ngược lại || gọi lệnh tiếp theo khi lệnh trước đó *false*.

cuu duong than cong . com

CuuDuongThanCong.com https://fb.com/tailieudientucntt

```
#!/bin/sh

rm -f file_one

if [ -f file_one ] || echo "hello" || echo "there"
then
    echo "in if"
else
    echo "in else"
fi

exit 0
```

8. Express

Lệnh **expr** tính các đối đầu vào như một biểu thức. Thường **expr** được dùng trọng việc tính toán các kết qủa toán học khi đổi giá trị từ chuỗi sang số. Ví dụ :

```
x= “12"
x= `expr $x + 1`
```

Kết quả x=13. Lưu ý, cặp dấu ‘ ‘ bọc biểu thức expr không phải là dấu nháy đơn (Ký tự này là phím nằm dưới phím ESC và bên trái phím 1, chung với phím ~. Các toán hạng và toán tử phải cách nhau bằng khoảng trắng. Ở đây $x và 1 cách ký tự + khoảng trắng. Nếu để chúng sát nhau, khi diễn dịch shell sẽ báo lỗi biểu thức.

Dưới đây là một số biểu thức ước lượng mà expr cho phép:

Biểu thức	Ý nghĩa
axprl | expr2	Kết quả là expr1 nếu expr1 khác 0 ngược lại là expr2
axprl & expr2	0 nếu một trong hai biểu thức là zero ngược lại kết quả là expr1
exprl = expr2	Bằng
exprl >expr2	Lớn hơn
exprl >= expr2	Lớn hơn hay bằng
exprl < expr2	Bé hơn
exprl <= expr2	Bé hơn hay bằng
exprl != expr2	không bằng
exprl + expr2	Cộng
exprl - expr2	Trừ
exprl * expr2	Nhân
exprl / expr2	Chia
exprl % expr2	Modulo (lấy số dư)

Trong các shell mới sau này lệnh **expr** được thay thế bằng cú pháp $ ((. . .)) hiệu quả hơn, và sẽ đề cập cú pháp này ở phần sau.

9. Lấy kết quả

CuuDuongThanCong.com https://fb.com/tailieudientucntt

Khi viết lệnh cho script chúng ta thường có nhu cầu lấy về kết xuất hay kết quả của một lệnh để dùng cho lệnh tiếp theo. Ví dụ, ta gọi thực thi một lệnh và muốn lấy kết quả trả về của lệnh làm nội đung lưu trữ vào biến. Ta có thể làm điều này dựa vào lệnh có cú pháp **$(command)** (chúng ta đã gặp ở ví dụ khi minh họa lệnh **set**). Cú pháp này còn có thể dùng ở dạng khác là `**command** (lưu ý về dấu nháy ` là phím nằm chung với ký tự ~ chứ không phải là nháy đơn thông thường, dấu này được gọi là dấu bao ngược - backquote).

Kết qủa của **$(command)** đơn giản chỉ là kết xuất của **command**. Nó không phải là mã lỗi trả về của lệnh.

```
#!/bin/sh
echo Current directory is $PWD
echo It contents $(ls -a) files
exit 0
```

Nếu đang ở thư mục /root và thư mục này có các the .bash_profile, use_command.sh, thì kết quả kết xuất sẽ như sau:

$./use_command.sh
Current directory is /root
It contents $ (ls -a) .bash_profile use_command.sh files.

Cách chương trình làm việc

Lệnh ls -a dùng liệt kê nội dung thư mục /root. Kết quả trả về được đặt trong $ ( ) sẽ được diễn dịch thành nội dung:

$ (ls -a), ch kết quả: $ (.bash_profile use_conunand.sh)

cuu duong than cong . com

cuu duong than cong . com

CuuDuongThanCong.com https://fb.com/tailieudientucntt

Chương 4

LẬP TRÌNH TRÊN LINUX

Rất nhiều bạn nghĩ rằng lập trình trên Unix , Linux luôn luôn phải dùng ngôn ngữ C. Điều này hoàn toàn đúng bởi vì nguyên thủy Unix được viết từ C và phần lớn các ứng dụng cho Unix cũng dùng C để viết. Mặc dù vậy C không phải là một lựa chọn duy nhất và bắt buộc. Ngoài ngôn ngữ C còn có thể sử dụng nhiều ngôn ngữ khác để lập trình như Pascal, Assembler hay Perl, Fortran, Prolog…

Tuy nhiên **C/C++** và **Pascal** là hai ngôn ngữ có khả năng biên dịch mạnh và gần gũi với chúng ta nhất. Trình biên dịch C và Pascal trên Linux hoàn toàn có khả năng biên dịch cả mã nguồn viết bằng ngôn ngữ Assembler.

Chương trình ứng dụng trên Unix, Linux tồn tại ở 2 dạng: dạng thực thi (tập tin nhị phân) và dạng thông dịch **script**. Tập tin chương trình thực thi ở dạng nhị phân tương tự như tập tin **.exe** của **DOS. Script** là những chỉ thị lệnh bằng văn bản diễn dịch và thực thi bởi shell hay trình thông dịch nào đó. Các file **script** tương tự như các **file.bat** của DOS.

Hầu như script hay chương trình mã máy đều có khả năng và sức mạnh ngang nhau. Bạn khó phân biệt đâu là lệnh gọi chương trình nhị phân đâu chương trình gọi lệnh script trong Unix và Linux trừ khi xem nội dung của chúng. Chương trình nhị phân và chương trình gọi lệnh script có thể hoán đổi cho nhau. Một chương trình script nếu thích bạn có thể chuyển thành chương trình nhị phân bằng ngôn ngữ biên dịch C hay Pascal.

Khi lần đầu tiên đăng nhập vào hệ thống Linux và gọi một chương trình từ dòng lệnh, hệ điều hành Linux sẽ tìm đường dẫn đến nơi chứa tập tin chương trình trong biến môi trường PATH. Thường thì biến môi trường này sẽ chứa các đường dẫn cơ bản như: **/bin, /user/bin, /usr/local/bin**

Các thành phần công cụ hỗ trợ hệ điều hành thường được cài đặt vào thư mục **/opt**. Linux không tìm đường dẫn chương trình trong thư mục hiện hành trừ khi bạn thêm ký tự **(.)** vào biến môi trường PATH.

Lưu ý là Unix, Linux sử dụng ký tự: để phân cách các đường dẫn trong biến môi trường PATH trong khi DOS sử dụng ký tự ; Ví dụ:

PATH= /bin:/user/bin:/usr/local/bin

4.1 Chương trình helloworld.c

Hello World là chương trình kinh điển đối với hầu hết các lập trình viên khi tìm hiểu một ngôn ngữ lập trình mới. Trong phần này sẽ giúp các bạn hiểu về các bước tiến hành lập trình một ứng dụng bằng ngôn ngữ C, biên dịch và thực thi ứng dụng trên môi trường linux.

Trước hết, mở cửa sổ **Terminal**. **Terminal** là giao diện dòng lệnh cho phép người dùng tương tác với hệ thống thông qua các lệnh điều khiển, nó tương tự như bạn nhập các lệnh trong của sổ cmd của windows.

Có nhiều cách khởi động cửa sổ Terminal. Trong các phiên bản RedHat, chúng ta chỉ cần Click phải lên **Destop**, chọn menu lệnh **Terminal**. Trong các phiên bản khác như Fedora Core, để mở của sổ **Termina**l Chọn menu **Applications / System Tools / Terminal**

```
[root@localhost ~]#
[root@localhost ~]#
[root@localhost ~]#
```

Dấu nhắc trong cửa sổ Terminal

Tạo một file có tên **helloworld.c**. Có nhiều cách để tạo một tập tin mã nguồn. Bạn có thể tạo trực tiếp trên cửa sổ lệnh của Linux, hoặc có thể tạo bằng các chương trình hỗ trợ trên

CuuDuongThanCong.com https://fb.com/tailieudientucntt

Windows và sao đó chép sang máy Linux (phần này sẽ hướng dẫn cụ thể sau). Trong phần này sẽ hướng dẫn tạo một tập tin mã nguồn trực tiếp trên Linux.

Sử dụng trình soạn thảo vi như sau:

vi helloworld.c

Trong trường hợp tập tin chỉ định trong lệnh không tồn tại, trình soạn thảo **vi** tự hiểu phải tạo một tập tin mới có tên chỉ ra trong lệnh, ngược lại nếu nó sẽ mở tập tin cho phép người dùng thay đổi.

Lần đầu tiên **vi** mở tập tin, để có thể chèn nội dung vào tập tin, bạn nhấn phím **I** để chuyển sang chế độ **insert,** Nhập nội dung như sau:

```
#include<stdio.h>
int main ()
 {
printf (“Hello World”) ;
exit(0) ;
}
```

Thoát khỏi chế độ INSERT và chuyển sang chế độ lệnh bằng cách nhấn ESC .

Để lưu lại nội dung tập tin nhấn tổ hợp phím :w

Để thoát khỏi trình **vi** nhấn tổ hợp phím :q

Lúc này trong thư mục hiện hành sẽ tạo ra tập tin có tên **helloworld.c**

Để biên dịch tập tin mã nguồn **.c** trên Linux sử dụng trình biên dịch **gcc**. Trước hết bạn cần kiểm tra sự tồn tại và phiên bản của trình **gcc** trên máy Linux của bạn. Thông thường khi cài đặt hệ điều hành Linux thì gcc được cài đặt sẵn. Để xem thông tin về trình biên dịch b, tại dòng lệnh bạn nhập lệnh **gcc –v**.

Một loạt các thông tin trong đó đáng chú ý là phiên bản của **gcc**:

```
[root@localhost sonl]#
[root@localhost sonl]# gcc -v
Using built-in specs.
Target: i386-redhat-linux
Configured with: ../configure --prefix=/usr --mandir=/usr/share/man --infodir=
 --enable-threads=posix --enable-checking=release --with-system-zlib --enable-
-enable-java-awt=gtk --disable-dssi --enable-plugin --with-java-home=/usr/lib/
r=/usr/share/java/eclipse-ecj.jar --disable-libjava-multilib --with-cpu=generi
Thread model: posix
gcc version 4.3.0 20080428 (Red Hat 4.3.0-8) (GCC)
[root@localhost sonl]#
```

Biên dịch tập tin mã nguồn:

gcc helloworld.c –o helloworld

```
[root@localhost sonl]# gcc helloworld.c -o helloworld
helloworld.c: In function âmainâ:
helloworld.c:8: warning: incompatible implicit declaration of built-in function âexitâ
[root@localhost sonl]# ls
helloworld  helloworld.c
[root@localhost sonl]#
```

Sau khi quá trình biên dịch thành công, tập tin thực thi **helloworld** sẽ được tạo ta. Để gọi tập tin này chạy trên môi trường Linux, có thể gọi trực tiếp như sau:

./helloworld

CuuDuongThanCong.com https://fb.com/tailieudientucntt

```
[root@localhost sonl]# ./helloworld

 Hello World
  [root@localhost sonl]#
```

Kết quả khi chạy lệnh printf (**"Hello World "**) sẽ hiển thị thông tin ra của sổ.

Trình biên dịch gcc yêu cầu file chứa mã nguồn C trên tham số dòng lệnh để biên dịch. Ở đây ta chỉ định helloworld.c ở đối số dòng lệnh thứ nhất. Tùy chọn **–o** yêu cầu trình biên dịch tạo ra file kết xuất (file chương trình thực thi) mang tên helloworld (bạn có thể chỉ định một tên file kết xuất khác với file mã nguồn). Nếu bạn không chỉ định tùy chọn **–o** thì trình biên dịch sẽ tạo ra file thực thi với tên **a.out**. Khi đó thay vì gọi **helloworld** trên dòng lệnh bạn phải gọi **a.out**

4.2 Chương trình trên Linux:

Với tư cách là nhà phát triển chương trình bạn cần nắm rõ một số vị trí đặt tài nguyên để xây dựng chương trình như trình biên dịch, file thư viện, các file header khai báo hàm cấu trúc dữ liệu, các file chương trình sau khi biên dịch sẽ được đặt ở đâu..

Trình biên dịch gcc thường được đặt trong thư mục **/usr/bin** hoặc **/usr/local/bin**. Tuy nhiên khi biên dịch gcc cần đến nhiều file hỗ trợ nằm trong các thư mục khác như các file C header thường được đặt trong thư mục **/usr/include** hay **/usr/local/include**. Các file thư viện liên kết thường được gcc tìm trong thư **mục /lib** hoặc **/usr/local/lib**. Các thư viện chuẩn của gcc thường đặt trong thư mục **/usr/lib/gcc-lib**

Chương trình của bạn sau khi biên dịch ra có thể đặt ở bất kỳ nơi đâu trong hệ thống, miễn là hệ điều hành có thể tìm thấy trong biến môi trường PATH hoặc trên đường dẫn tuyệt đối khi bạn gọi chương trình từ dòng lệnh.

Các file header trong C thường định nghĩa hàm và khai báo các hàng cộng với cấu trúc dữ liệu cần thiết cho quá trình biên dịch. Hầu hết các chương trình trên Linux khi biên dịch sử dụng các file **header** trong thư mục **/usr/include** hoặc các thư mục con bên dưới thư mục này. Ví dụ như **/usr/include/asm, /usr/include/sys**. Trong các chương trình C sau này có thể bạn sẽ gặp các khai báo như:

#include<sys/types.h>

Trình biên dịch lúc này sẽ tìm file **header** mang tên **types.h** trong thư mục con **sys** của **/usr/include**. Một số thư mục chứa file header được các trình biên dịch dò tìm măc định như **/usr/include/x11** đối với các khai báo hàm lập trình đồ họa X-Window. Hoặc thư mục **/usr/include/g++ -2** đối với trình biên dịch **GNU g++.**

Ví dụ viết một chương trình giải phương trình bậc 2 bằng ngôn ngữ C, biên dịch và chạy trên Linux.

```
#include<stdio.h>
#include<math.h>
int main()
{
        float a,b,c,x1,x2,delta;
        do{
        printf("\nNhap vao he so a \n") ;
        scanf("%f",&a) ;
            }
        while(a==0) ;
        printf("\nNhap vao he so b \n") ;
        scanf("%f",&b) ;
        printf("\nNhap vao he so c \n") ;
```


CuuDuongThanCong.com https://fb.com/tailieudientucntt

```
        scanf("%f",&c) ;
        delta = b*b - 4*a*c ;

        if (delta<0)
        printf("\nphuong trinhh vo nghiem") ;
        else if (delta==0)
        printf("\nphuong trinh co nghiem kep x= %f ",-b/(2*a));
        else
        printf("\nPhuong trinh co nghiem \n x1=%f \nx2=%f");

        exit(0);

}
```

Biên dịch chương trình

gcc main.c main

./main

4.3 Tiến trình (process)

Một thể hiện đang chạy của một chương trình được gọi là một tiến trình (process). Ví dụ chúng ta có 2 ứng dụng cửa sổ **terminal** trên màn hình chúng ta xem như có 2 tiến trình đang chạy. Mỗi cửa sổ **terminal** đang thực thi một **shell**. Mỗi **shell** là một tiến trình khác nhau. Khi nhập một lệnh vào từ mỗi **shell**, chương trình sẽ thực thi một tiến trình mới, khi tiến trình mới này kết thúc thì tiến trình **shell** lại được tiếp tục.

Lập trình trên Linux thường sử dụng nhiều tiến trình trong một ứng dụng để cho phép ứng dụng có thể thực thi nhiều công việc hơn nhằm làm gia tăng tính hiệu quả của ứng dụng.

Hầu hết các hàm xử lý các tiến trình mô tả trong phần này tương tự như trong các hệ thống Unix khác. Hầu hết được khai báo trong tập tin ***unistd.h***

4.4 Số hiệu của tiến trình (process IDs):

Mỗi tiến trình trong hệ thống Linux được xác định bởi một số hiệu duy nhất. Thông số này còn được gọi là **pid.** Số hiệu tiến trình là một số nguyên **16 bit** được cấp liên tục bởi hệ thống Linux khi một tiến trình mới được tạo.

Mỗi tiến trình có một tiến trình mẹ **(parent process)** ngoại trừ các tiến trình đặc biệt như **init process**. Ví thế các tiến trình trong hệ thống Linux được xếp theo hình cây với tiến trình **init** là gốc. **parent Process ID** hay **ppid** đơn thuần là các số hiệu của các tiến trình mẹ.

Khi thao tác với các process ID trong các chương trình viết bằng **C/C++** chúng ta thường sử dụng các kiểu dữ liệu đã định nghĩa sẵn **pid_t** trong thư viện **<sys/types.h>.** Một chương trình có thể đọc một số hiệu của một tiến trình đang chạy bằng cách gọi hàm hệ thống **getpid().** Tương tự cũng có thể lấy số hiệu của tiến trình mẹ bằng cách gọi hàm hệ thống **getppid()**.

Ví dụ:

```
#include <stdio.h>
#include <unistd.h>
int main ()
{
printf ("The process ID is %d\n", (int) getpid ());
```


CuuDuongThanCong.com https://fb.com/tailieudientucntt

```
printf ("The parent process ID is %d\n", (int) getppid ());
return 0;
}
```

4.5 Tạo tiến trình:

Có 2 kỹ thuật chung được dùng để tạo mới một tiến trình. Kỹ thuật thứ nhất hầu như đơn giản nhưng tính hiệu quả không cao do không an toàn. Kỹ thuật thứ 2 thì phức tạp hơn nhưng có khả năng mềm dẻo hơn, nhanh hơn và an toàn hơn.

4.5.1 Sử dụng *system:*

Hàm system trong thư viện chuẩn của C cung cấp là cách thức đơn giản để thực thi một lệnh từ bên trong một chương trình, tương tự như một lệnh được nhập vào từ cửa sổ **shell**. Thực tế hàm **system** tạo một tiến trình phụ chạy trong shell chuẩn Bourne (**/bin/sh).** Ví dụ đoạn chương trình sau sẽ gọi thực thi lệnh **ls** hiển thị nội dung của cây thư mục gốc tương tự như khi bạn gõ **ls –l** trong môi trường shell.

```
#include <stdlib.h>
int main ()
{
int return_value;
return_value = system ("ls -l /");
return return_value;
}
```

Hàm **system** trả về trạng thái kết thúc của lệnh thực thi **shell**. Nếu shell không thực hiện lệnh nó sẽ trả về giá trị **127**, nếu một lỗi khác xảy ra, hàm system sẽ trả về giá trị **-1**.

Bởi vì hàm **system** sử dụng **shell** để gọi lệnh của bạn nên nó lệ thuộc vào đặc điểm, giới hạn và luật bảo mật của **shell** hệ thống. Bạn không thể dựa vào bất kỳ khả năng nào của các phiên bản riêng biệt của **Bourne shell**. Trong nhiều hệ thống **Unix /bin/sh** là một liên kết đến một **shell** khác. Ví dụ trong hầu hết các hệ thống **GNU/UNIX /sbin/sh** trỏ đến bash shell. Những phiên bản phân phối khác của **GNU/UNIX** thì sử dụng những phiên bản khác nhau của **bash shell**. Cụ thể là một **chương trình với một quyền root, với hàm system dĩ nhiên là có nhiều kết quả khác nhau trên các hệ thống GNU/UNIX** khác nhau. Chính vì điều đó mà sử dụng **fork** và **exec** để tạo mới một tiến trình.

4.5.2 Sử dụng fork và exec:

DOS và API trên Winddows bao gồm họ các hàm SPAWN. Thông số của các hàm này là tên của một chương trình để chạy và tạo một tiến trình mới thể hiện của chương trình đó. Linux thì không bao gồm một hàm đơn để xử lý tất cả trong một bước. Thay vào đó Linux cung cấp một hàm fork, nó có thể tạo một tiến trình con như một sao chép chính xác tiến trình mẹ. Để tạo một tiến trình trước tiên sử dụng lệnh **fork** để tạo một bản sao của tiến trình hiện tại. Sau đó sử dụng hàm **exec** để chuyển đổi một trong những tiến trình này thành thể hiện của chương trình mà bạn muốn tạo tiến trình.

Khi chương trình gọi lệnh **fork**, một bản sao của tiến trình còn được gọi là tiến trình con **(child process)** được tạo ra. Tiến trình mẹ tiếp tục thực hiện chương trình từ vị trí **fork** được gọi.

Giữa 2 tiến trình khác nhau như thế nào? Trước hết tiến trình con là một tiến trình mới vì thế nó có một số hiệu (ID) mới khác biệt với số hiệu của tiến trình cha. Một cách khác để chương trình có thể nhận ra đâu là tiến trình cha và đâu là tiến trình con là gọi hàm ***getpid.*** Tuy nhiên hàm **fork** cung cấp giá trị trả về khác nhau cho tiến trình cha và tiến trình con khi được tạo. Giá trị trả về của tiến trình cha là số hiệu của tiến trình con, giá trị trả về của tiến trình con là **zero** bởi vì không có tiến trình nào có số hiệu bằng **zero**.

CuuDuongThanCong.com https://fb.com/tailieudientucntt

Ví dụ sử dụng lệnh **fork** để tạo một tiến trình con. Chú ý rằng trong khối đầu tiên của phát biểu **if** chỉ được thực thi trong tiến trình cha, trong khi phát biểu trong mệnh đề **else** lại được thực thi trong tiến trình con.

```
#include <stdio.h>
#include <sys/types.h>
#include <unistd.h>
int main ()
{
pid_t child_pid;
printf ("the main program process ID is %d\n", (int) getpid ());
child_pid = fork ();
if (child_pid != 0)
        {
        printf ("this is the parent process, with id %d\n", (int) getpid ());
        printf ("the child's process ID is %d\n", (int) child_pid);
        }
else
printf ("this is the child process, with id %d\n", (int) getpid ());
return 0;
}
```

Hàm **exec** đặt một chương trình đang chạy trong một tiến trình với một chương trình khác. Khi một chương trình gọi hàm **exec**, tiến trình đó ngay lặp tức dừng chương trình đang thực thi và bắt đầu thực thi một chương trình mới từ đó giả sử rằng gọi hàm **exec** không phát sinh ra lỗi. Một số các hàm **exec** thông dụng như sau:

Các hàm bao gồm ký tự **p** sau tên của **exec (execvp, execlp)** nhận vào tên chương trình và tìm chương trình thông qua tên của nó trong đường dẫn của chương trình thực thi hiện hành. Hàm không bao gồm ký tự **p** phải được cung cấp đường dẫn đầy đủ của chương trình để thực thi.

Các hàm bao gồm ký tự **v** sau tên **exec (execv, execvp, execve)** nhận vào danh sách các đối số cho chương trình mới như là một mảng khác NULL của con trỏ chuỗi. Các hàm bao gồm ký tự **l (execl, execlp và execle)** nhận vào danh sách các đối số sử dụng cơ chế của ngôn ngữ C.

Các hàm bao gồm ký tự **e (execve, execle)** nhận vào các đối số truyền thống, một mảng các biến môi trường. Các đối số phải là một mảng khác NULL con trỏ tới chuỗi ký tự, mỗi chuỗi ký tự có dạng "VARIABLE=value"

Một cách chung nhất để chạy một chương trình con bên trong một chương trình kết hợp gọi lệnh fork và exec như ví dụ sau:

```
#include <stdio.h>
#include <stdlib.h>
#include <sys/types.h>
#include <unistd.h>
/* Spawn a child process running a new program. PROGRAM is the name
of the program to run; the path will be searched for this program.
ARG_LIST is a NULL-terminated list of character strings to be
passed as the program's argument list. Returns the process ID of
the spawned process.  */
```


CuuDuongThanCong.com https://fb.com/tailieudientucntt

```
int spawn (char* program, char** arg_list)
{
pid_t child_pid;
/* Duplicate this process.  */
child_pid = fork ();
if (child_pid != 0)
/* This is the parent process.  */
return child_pid;
else {
/* Now execute PROGRAM, searching for it in the path.  */
execvp (program, arg_list);
/* The execvp function returns only if an error occurs.  */
fprintf (stderr, "an error occurred in execvp\n");
abort ();
}
}
int main ()
{
/* The argument list to pass to the "ls" command.  */
char* arg_list[] = {
"ls",   /* argv[0], the name of the program.  */
"-l",
"/",
NULL    /* The argument list must end with a NULL.  */
};
/* Spawn a child process running the "ls" command.  Ignore the
returned child process ID.  */
spawn ("ls", arg_list);
printf ("done with main program\n");
return 0;
}
```

4.6 Tuyến (thread):

Tuyến **(thread)** là một phần của tuyến trình sở hữu riêng ngăn xếp và thực thi độc lập ngay trong mã lệnh của tiến trình. Nếu như một hệ điều hành có nhiều tiến trình thì bên trong mỗi tiến trình, bạn có thể tạo ra nhiều tuyến hoạt động song song với nhau tương tự như các tiến trình hoạt động song song bên trong hệ điều hành. Ưu điểm của tuyến là chúng hoạt động trong cùng một không gian địa chỉ của tiến trình. Tập hợp một nhóm các tuyến có thể chia sẻ chung vùng nhớ của một tiến trình và do đó có thể sử dụng chung biến toàn cục, vùng nhớ **heap**… của tiến trình. Cơ chế liên lạc giữa các tuyến đơn giản và hiệu quả hơn cơ chế liên lạc giữa các tiến trình.

4.6.1 Tạo tuyến:

Mỗi tuyến trong một tiến trình được định nghĩa bởi một số hiệu (thread ID). Khi tương tác với các số hiệu của tuyến trong lập trình **C**, **C++** ta có thể sử dụng kiểu cấu trúc đã định nghĩa trước **pthread_t**.

CuuDuongThanCong.com https://fb.com/tailieudientucntt

Mỗi tuyến thực thi một hàm của tuyến. Hàm này cũng giống như các hàm thông thường khác, nó bao gồm các đoạn mã chương trình mà tuyến phải thực thi. Khi hàm này trả về giá trị, tuyến sẽ được kết thúc. Trong các hệ thống Unix, hàm tuyến chỉ bao gồm thông số con trỏ void*, kiểu giá trị trả về void*. Chương trình sử dụng thông số để truyền vào một tuyến khi tạo một tuyến mới. Tương tự chương trình cũng có thể sử dụng giá trị trả về của tuyến để truyền dữ liệu từ một tuyến đã kết thúc về chương trình tạo nó.

Hàm **pthread_create** tạo một tuyến mới, chúng ta có thể sử dụng với các thông số như sau:

Con trỏ đến kiểu dữ liệu **pthread_t ma** trong đó số hiệu của tuyến mới tạo được lưu trữ.

Con trỏ đến đối tượng thuộc tính của tuyến. Đối tượng này điều khiển việc tuyến tương tác như thế nào với các thành phần khác của chương trình. Nếu truyền vào con trỏ NULL cho thuộc tính của tuyến, tuyến sẽ được tạo với thuộc tính mặc định.

Con trỏ đến hàm của tuyến, đây là con trỏ hàm thông thường có dạng sau:

void* (*) (void*)

Giá trị đối số của tuyến có kiểu **void*.** Tất cả những gì truyền vào cho tuyến thực chất là truyền tham số cho hàm của tuyến khi tuyến được tạo.

Ví dụ chương trình sau tạo một tuyến và gọi tuyến thực thi:

```
#include <pthread.h>
#include <stdio.h>

void * display(void* unused)
{
        while(1)
                fputc('x',stderr);
        return NULL ;
}
int main ()
{
        pthread_t thread_id ;
        pthread_create(&thread_id,NULL,&display,NULL) ;
        while(1)
                fputc('o',stderr) ;
                return 0 ;
}
```

cuu duong than cong . com

Đặt tên tập tin chương trình **test.c**. Biên dịch chương trình trên dòng lệnh như sau:

```
gcc test.c –o test –lpthread
```

Thực thi chương trình trên dòng lệnh:

```
./test.
```

Kết quả bạn sẽ thấy các ký tự **x** và o xuất hiện trên màn hình, lệnh **putc** truyền một ký tự ra vùng đệm xuất **stderr**.

4.6.2 Hủy tuyến:

Hàm **pthread_exit()** được dùng để chấm dứt một tuyến hiện hành. Thường hàm này chỉ được dùng khi muốn thoát ra khỏi tuyến bất ngờ giữa quá trình thực hiện lệnh. Lệnh này không cần đặt ở cuối mỗi hàm thực thi tuyến**. pthread_exit()** cũng có thể dùng để trả dữ liệu về nơi đã gọi tuyến.

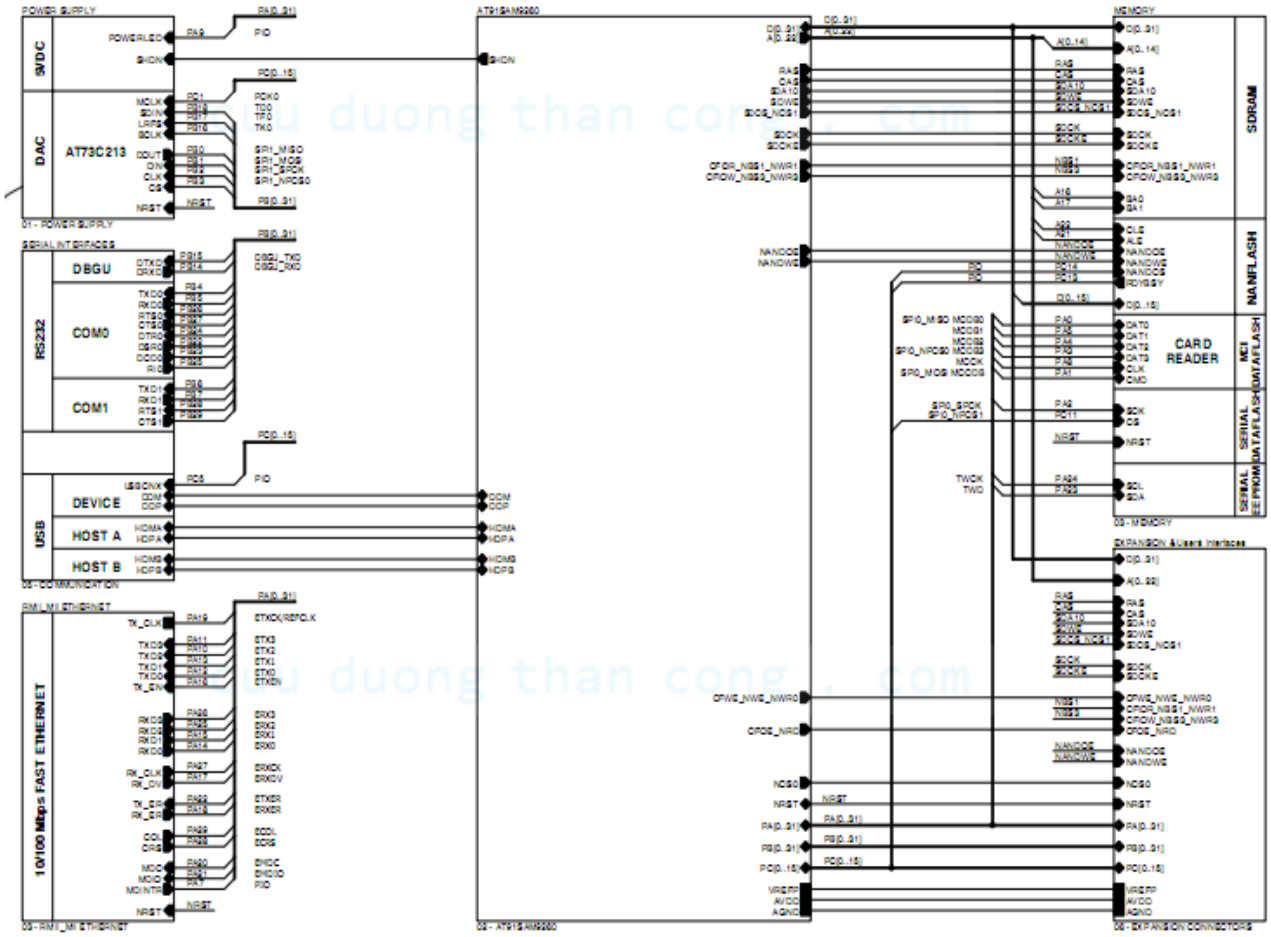
CuuDuongThanCong.com https://fb.com/tailieudientucntt

Chương 5
PHẦN CỨNG HỆ THỐNG NHÚNG

Hệ thống nhúng cũng là một hệ thống vi xử lý đảm nhiệm một chức năng riêng biệt nào đó. Tuy nhiên khi thiết kế hệ thống nhúng thì đòi hỏi phải quan tâm nhiều yếu tố hơn một hệ thống vi xử lý thông thường. Một hệ thống nhúng thông thường là một hệ thống dựa trên vi xử lý mà chủ yếu là các vi xử lý 32 bit kết hợp với bộ nhớ, các ngoại vi cần thiết phục vụ cho chức năng của hệ thống. Vi điều khiển ARM thường là lựa chọn tối ưu cho hệ thống nhúng bởi chúng hỗ trợ nhiều. Các hệ thống nhúng sử dụng ARM có thể làm giảm các ngoại vi, làm cho hệ thống đơn giản nhưng vẫn đáp ứng được yêu cầu. Hiện nay ARM được nhiều hãng nghiên cứu và sản xuất với tính năng ngày càng vượt trội.

Một hệ thống nhúng cơ bản có thể xem bao gồm các phần cứng như vi xử lý, vi điều khiển, bộ nhớ, các ngoại vi.

Trong tài liệu này sẽ giới thiệu phần cứng hệ thống nhúng dựa trên nền vi điều khiển ARM9 AT91Sam9260 của Atmel.



Hình 5. 1 Sơ đồ khối một hệ thống nhúng của Atmel

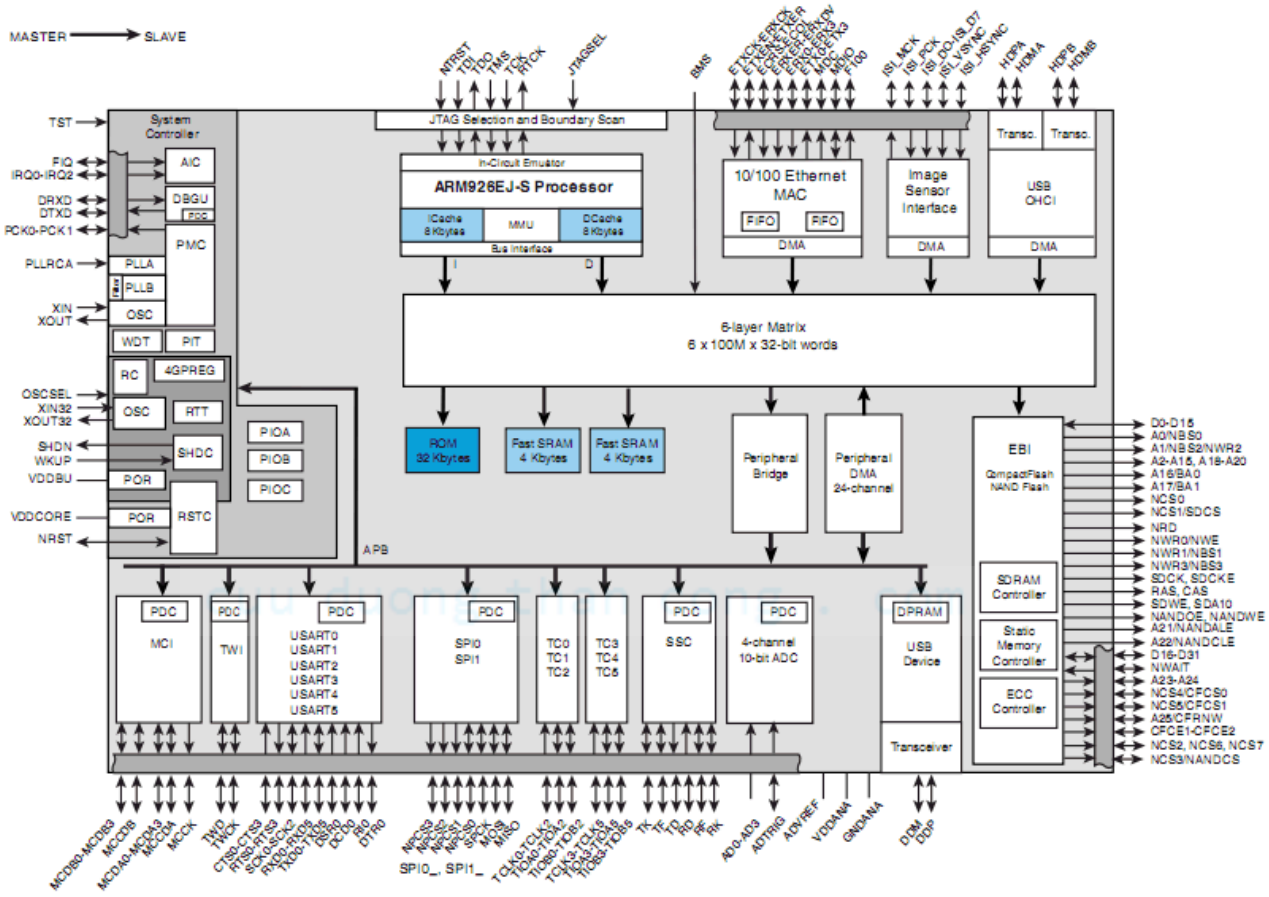
5.1 Vi xử lý, vi điều khiển:

CuuDuongThanCong.com https://fb.com/tailieudientucntt

Vi xử lý, vi điều khiển là thành phần quan trọng trong hệ thống nhúng, các vi điều khiển, vi xử lý phải đảm bảo về tốc độ xử lý, khả năng giao tiếp và mở rộng các ngoại vi để đáp ứng yêu cầu hệ thống.

(Phần này lên mạng giới thiệu chung các họ vi xử lý của các hãng nổi tiếng như Renesas, Samsung, Cirus Logic, Atmel).

Kiến trúc của một vi điều khiển ARM AT91SAM9260 của Atmel cơ bản như sau:



Hình 5.2 cấu trúc AT91SAM9260 của Atmel

AT91SAM9260 là một mạch tích hợp bộ xử lý ARM926EJ-S, bộ nhớ, một số IO và hệ thống bus.

Bộ xử lý ARM926EJ-S là một trong số các ARM9, thuộc họ các vi xử lý đa mục đích của Atmel. Bộ xử lý ARM926EJ-S bao gồm một kiến trúc ARM phiên bản 5TEJ được thiết kế cho các ứng dụng đa tác vụ với bộ quản lý bộ nhớ (MMU), tốc độ xử lý cao, kích thước bé và tiêu tán năng lượng thấp.

AT91SAM9260 có một số đặc điểm như sau:

- Bộ xử lý ARM926EJ-S, 8Kbyte bộ nhớ đệm lệnh, 8 Kbyte bộ nhớ đệm dữ liệu.
- Bus giao tiếp bộ nhớ ngoài 32 bit hỗ trợ 4 bank SDRAM/LPSDR, static Memory, CompactFlash, Nand Flash với EEC.
- 4 kbyte bộ nhớ SRAM nội, 32 Kbyte bộ nhớ ROM nội để chứa bootstrap.
- Ma trận bux AHB 6 lớp 32 bit với tốc độ 90Mhz
- 22 kênh DMA
- Boot từ Nand Flash, Sdcard, DataFlash, SiralDataFlash,
- On-chip Power-on Reset

CuuDuongThanCong.com https://fb.com/tailieudientucntt

- 4 bộ ADC 10 bit
- USB device và USB host
- 4 bộ USART, 2 UART
- 10/100Mbps ethernet.
- Giao tiếp SPI, SSC, 2 wire…

5.2 **Chương trình khởi động vi điều khiển AT91SAM9260 (boot program)**:

Chương trình khởi động tích hợp một chương trình có khả năng dowload hay upload một chương trình khác lên một vùng nhớ khác của vi điều khiển.

Trước hết nó khởi động đơn vị gỡ rối qua cổng nối tiếp và thiết bị USB. Tiếp theo đó chương trình khởi động trên DataFlash được thực thi, nó tìm một cách liên tục trong 8 vector của các dataFlash được kết nối đến vi điều khiển thông qua chuẩn SPI.

Nếu một vector hợp lệ được tìm thấy, mã chương trình sẽ được dowload vào bên trong SRAM nội, chương trình sẽ ánh xạ bộ nhớ và nhảy đến địa chỉ đầu tiên của SRAM

Nếu không có một vector hợp lệ nào được tìm thấy, chương trình khởi động trên DataFlash sẽ được thực thi trên chân chọn chip thứ 2(NPCS1).

Nếu không có một Vector hợp lệ nào được tìm thấy trên cả 2 chân chọn Serial DataFlash, chương trình khởi động trên NAND Flash sẽ được thực thi. Tương tự chương trình sẽ tìm 8 vector hợp lệ, nếu một vector hợp lệ được tìm thấy, mã chương trình sẽ được download lên SRAM, chương trình sẽ ánh xạ bộ nhớ và nhảy đến địa chỉ đầu tiên của SRAM.

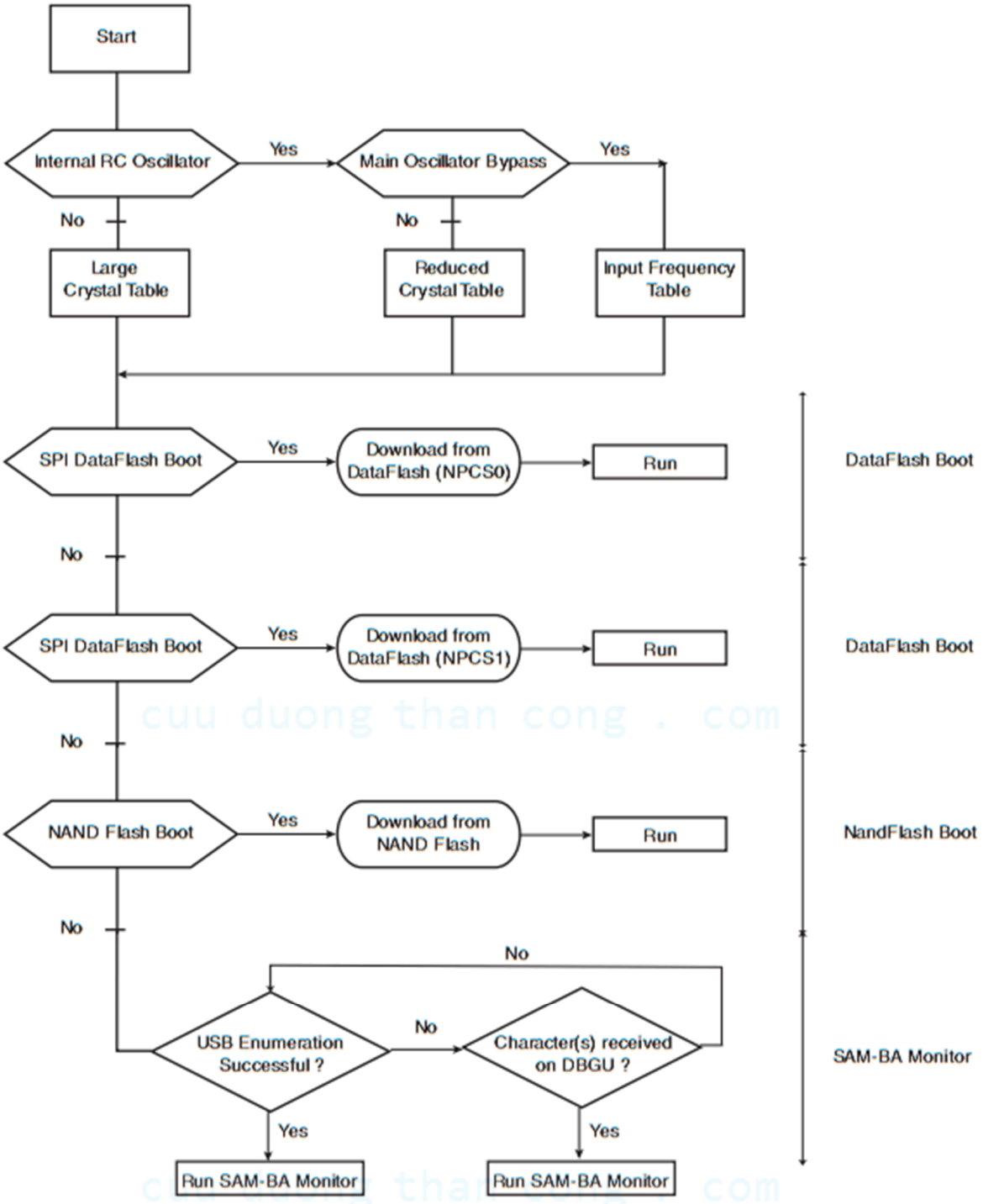
Nếu không có vector hợp lệ nào được tìm thấy trên NAND Flash, chương trình SAM-BA Monitor sẽ được thực thi và chờ quá trình truyền dữ liệu chương trình trên các cổng USB và JTAG.

cuu duong than cong . com

cuu duong than cong . com

CuuDuongThanCong.com https://fb.com/tailieudientucntt

Lưu đồ chương trình khởi động hệ thống được biểu diễn như sau:



Hình 5.3 Quá trình khởi động

5.3 Bus giao tiếp bộ nhớ ngoài:

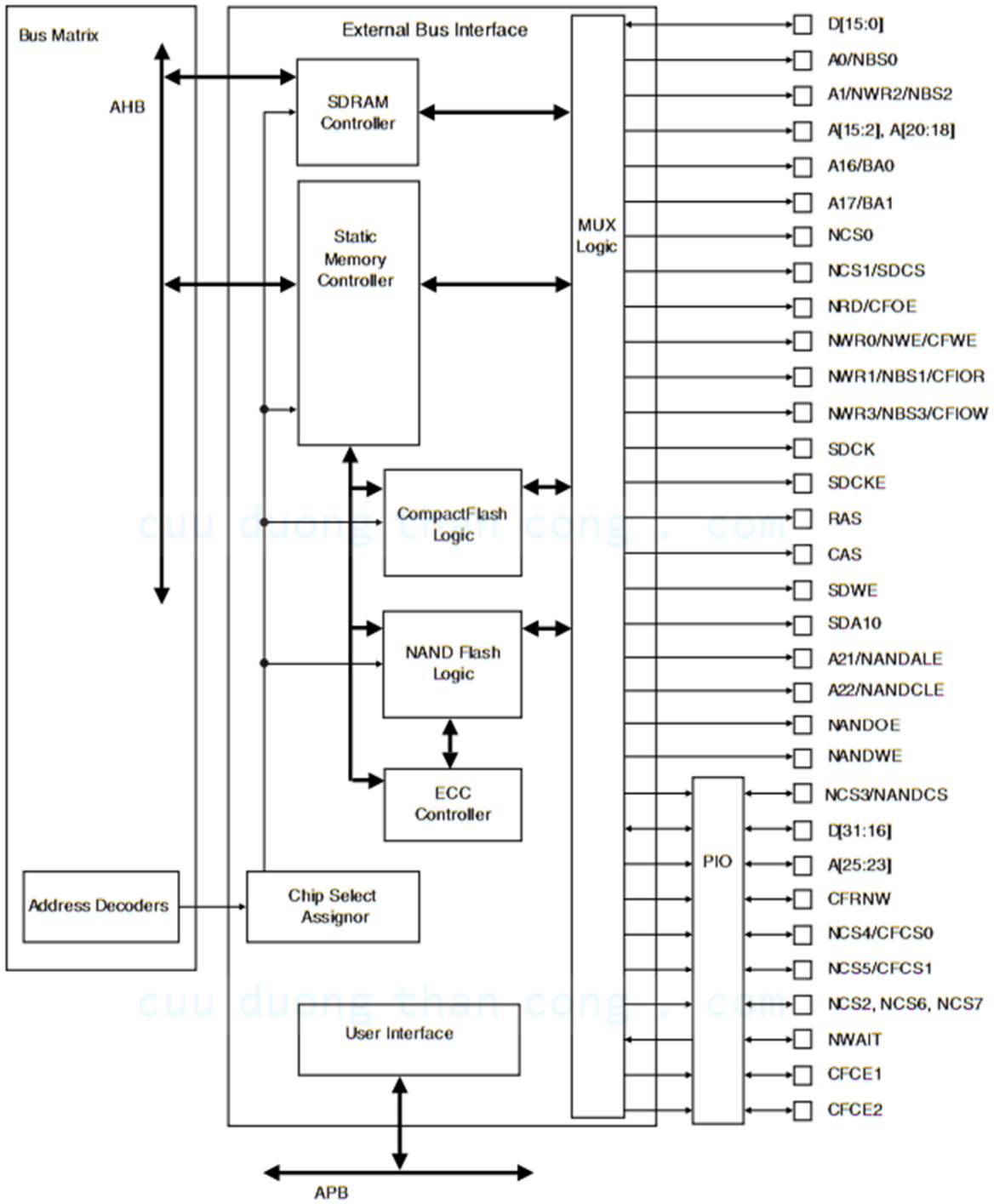
Bus giao tiếp bộ nhớ ngoài (External Bus Interface) được thiết kế để hỗ trợ quá trình truyền dữ liệu giữa một số thiết bị ngoại vi bên ngoài với bộ nhớ bên trong kiến trúc ARM. Bộ nhớ tĩnh (static memory), SDRAM, EEC cũng được đưa ra trình quản lý bộ nhớ trên bus giao

CuuDuongThanCong.com https://fb.com/tailieudientucntt

tiếp bộ nhớ ngoài. Các trình quản lý bộ nhớ ngoài này có khả năng giao tiếp một số loại bộ nhớ như SRAM, PROM, EPROM, EEPROM, Flash và SDRAM.

EBI cũng hổ trợ giao tiếp với nand Flash và CompactFlash. EBI quản lý truyền dữ liệu với hơn 6 thiết bị ngoại vi và được thiết kế 6 vùng nhớ do trình quản lý bộ nhớ nhúng bên trong định nghĩa. Dữ liệu truyền có thể được thực hiện 16 bit hay 32 bit, bus địa chỉ lên đến 26 bit, 8 đường chọn chip NCS[7..0].

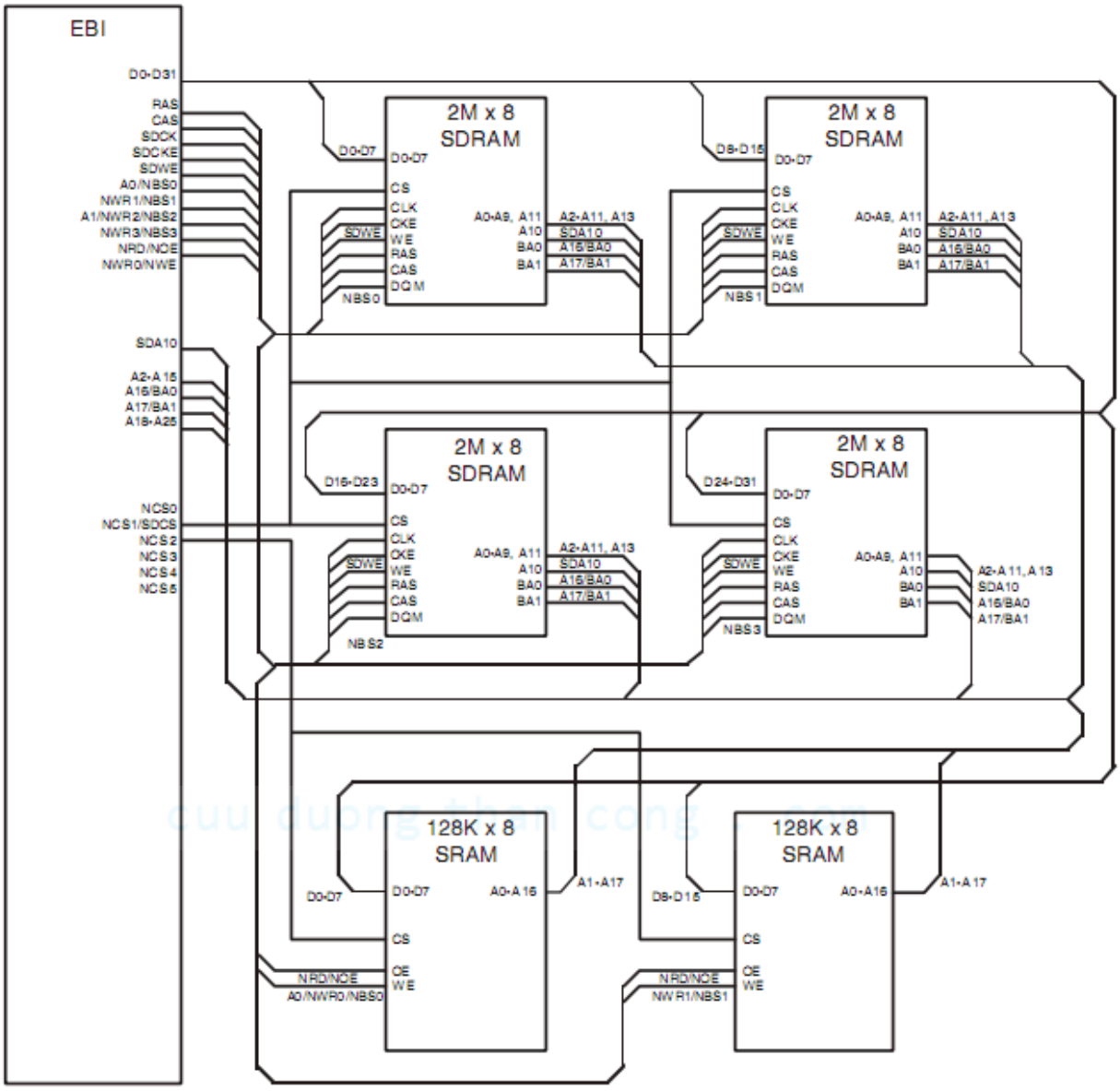
9-1. Organization of the External Bus Interface



Hình 5.4 Bus giao tiếp bộ nhớ ngoài

Ví dụ thiết kế với bộ nhớ SRAM ngòai như sau:

CuuDuongThanCong.com https://fb.com/tailieudientucntt



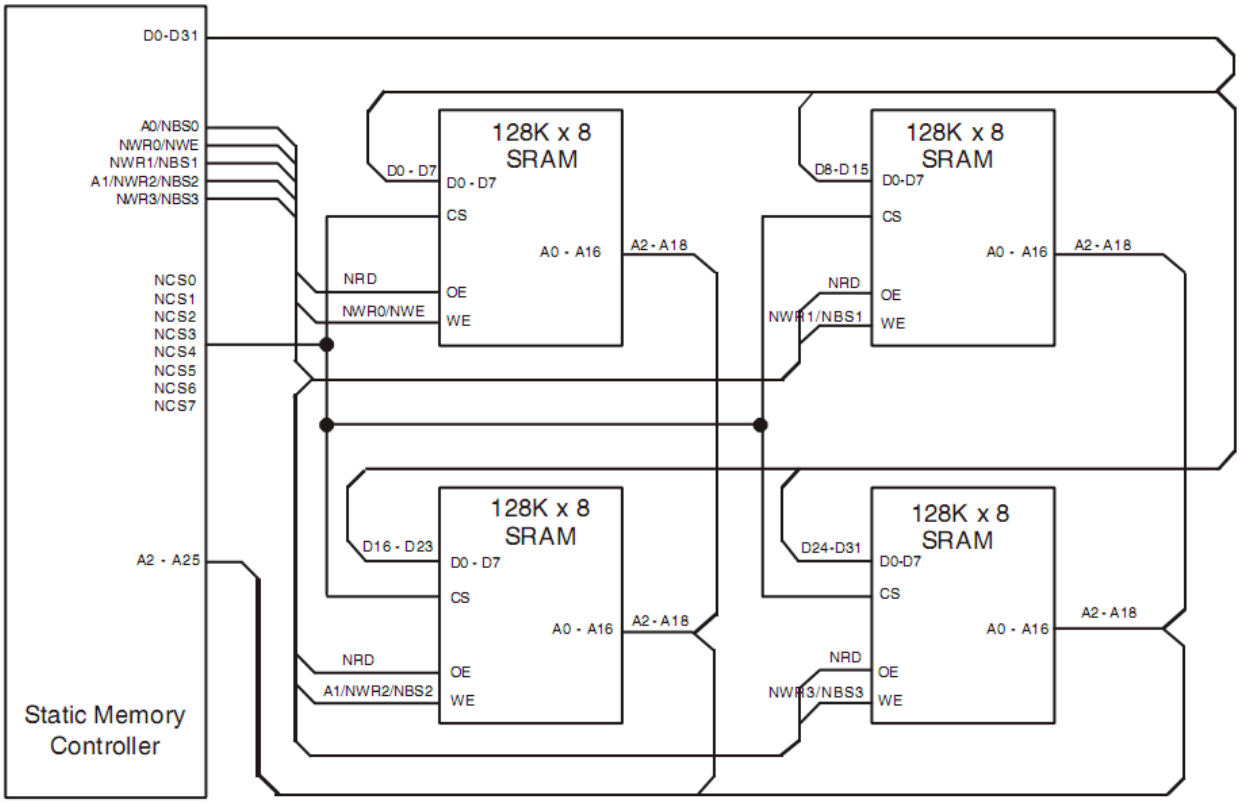
Hình 5.5 kết nối bộ nhớ SRAM ngoài

5.4 Static memory Controller:

Static memory Controller (SMC) tạo ra các tín hiệu điều khiển truy xuất đến vùng nhớ các thiết bị ngoài hay các ngoại vi kết nối bên ngoài chip. Nó bao gồm 8 chân tín hiệu chọn chip với bus địa chỉ 26 bit. Bus dữ liệu 32 bit có thể được cấu hình với độ rộng 8 bit, 16 bit, 32 bit. Các tín hiệu điều khiển đọc và ghi độc lập cho phép truy xuất vùng nhớ trực tiếp đến các thiết bị.

SMC kết nối đến bộ nhớ tĩnh như sau:

CuuDuongThanCong.com https://fb.com/tailieudientucntt

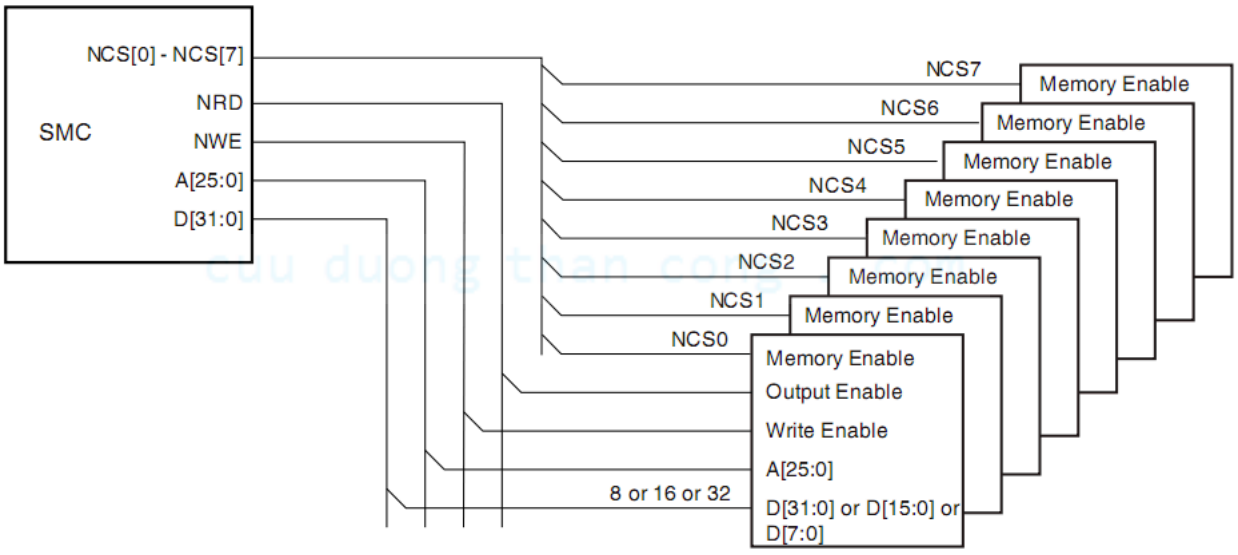


Hình 5.6 Kết nối bộ nhớ tĩnh

SMC cung cấp đến 26 đường địa chỉ, cho phép mỗi chân chọn chip có thể lên đến 64Mbyte bộ nhớ.

A[25..0] được sử dụng khi giao tiếp bộ nhớ 8 bit, A[25..1] được sử dụng khi giao tiếp bộ nhớ 16 bit, A[25..2] được sử dụng khi giao tiếp bộ nhớ 32 bit. Điều này giúp chúng ta hiểu hơn các thiết kế của hệ thống nhúng sau này.

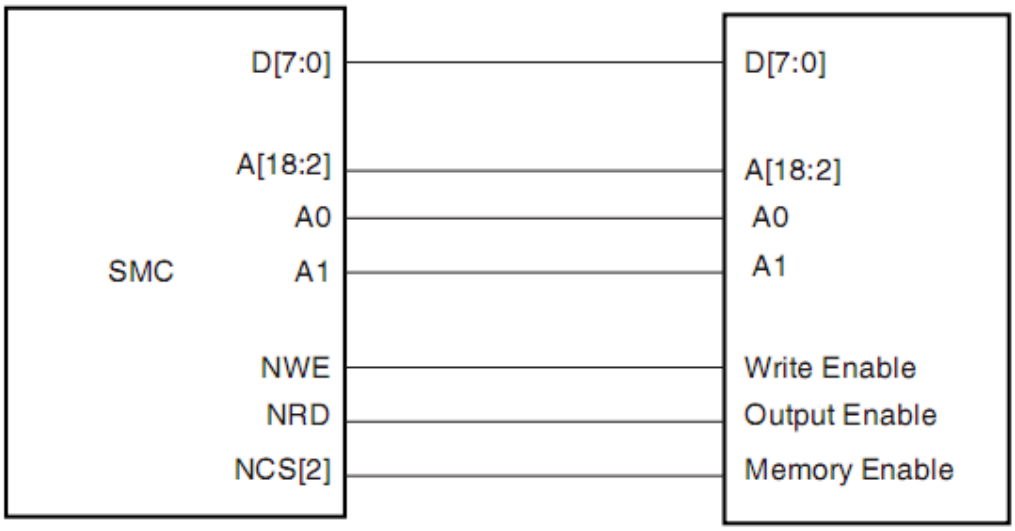
SMC kết nối với 8 thiết bị mở rộng như sau:



Hình 5.7 Kết nối với các thiết bị mở rộng

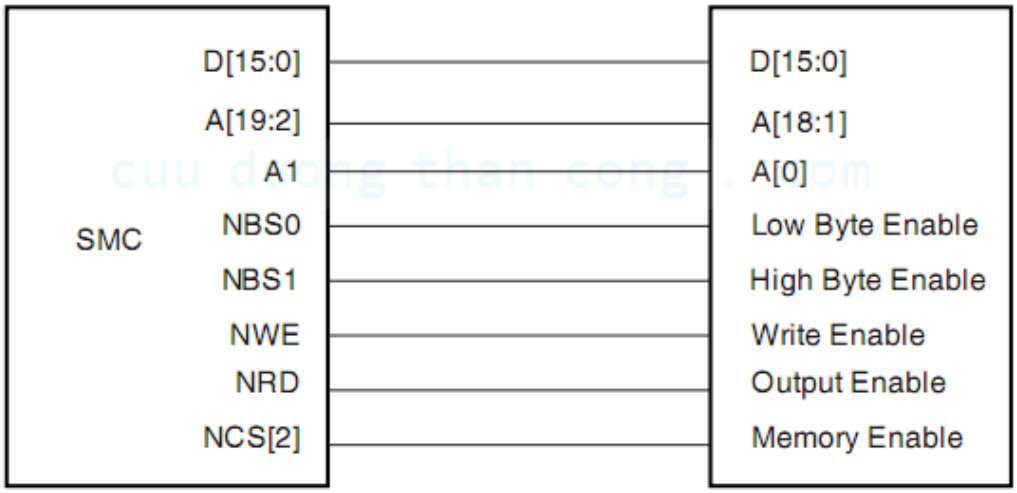
Kết nối bộ nhớ 512K x 8bit sử dụng tín hiệu chọn chip NCS2:

CuuDuongThanCong.com https://fb.com/tailieudientucntt



Hình 5.8 Kết nối bộ nhớ 512K X 8

Kết nối bộ nhớ 512K x 16 bit sử dụng tín hiệu chọn chip NCS2:

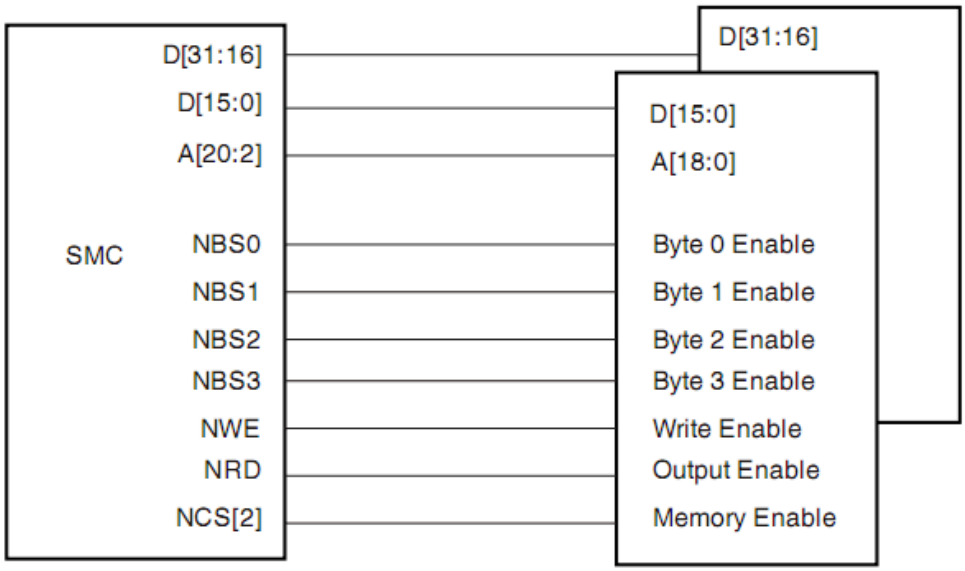


Hình 5.9 Kết nối bộ nhớ 512 X16

Kết nối 2 bộ nhớ 512 x 16 bit như bộ nhớ 32 bit sử dụng tín hiệu chọn chip NCS2

cuu duong than cong . com

CuuDuongThanCong.com https://fb.com/tailieudientucntt



Hình 5.10 Kết nối 2 bộ 512 X16

5.5 Bộ nhớ:

Bộ nhớ trong hệ thống nhúng thường bao gồm 2 bộ nhớ chính: bộ nhớ chương trình và bộ nhớ dữ liệu. Khác với các hệ thống vi xử lý thông thường, hệ thống nhúng thường có một hệ thống phần mềm điều khiển phức tạp hơn nên bộ nhớ cũng phức tạp hơn. Trong các hệ thống nhúng thì bộ nhớ chương trình có nhiệm vụ lưu trữ các phần mềm khởi động (Boot program) ảnh hệ điều hành (OS image) và hệ thống tập tin (File System). Trong tài liệu này sẽ trình bày một số thiết kế bộ nhớ của hệ thống nhúng dựa trên vi điều khiển ARM9 của Atmel.

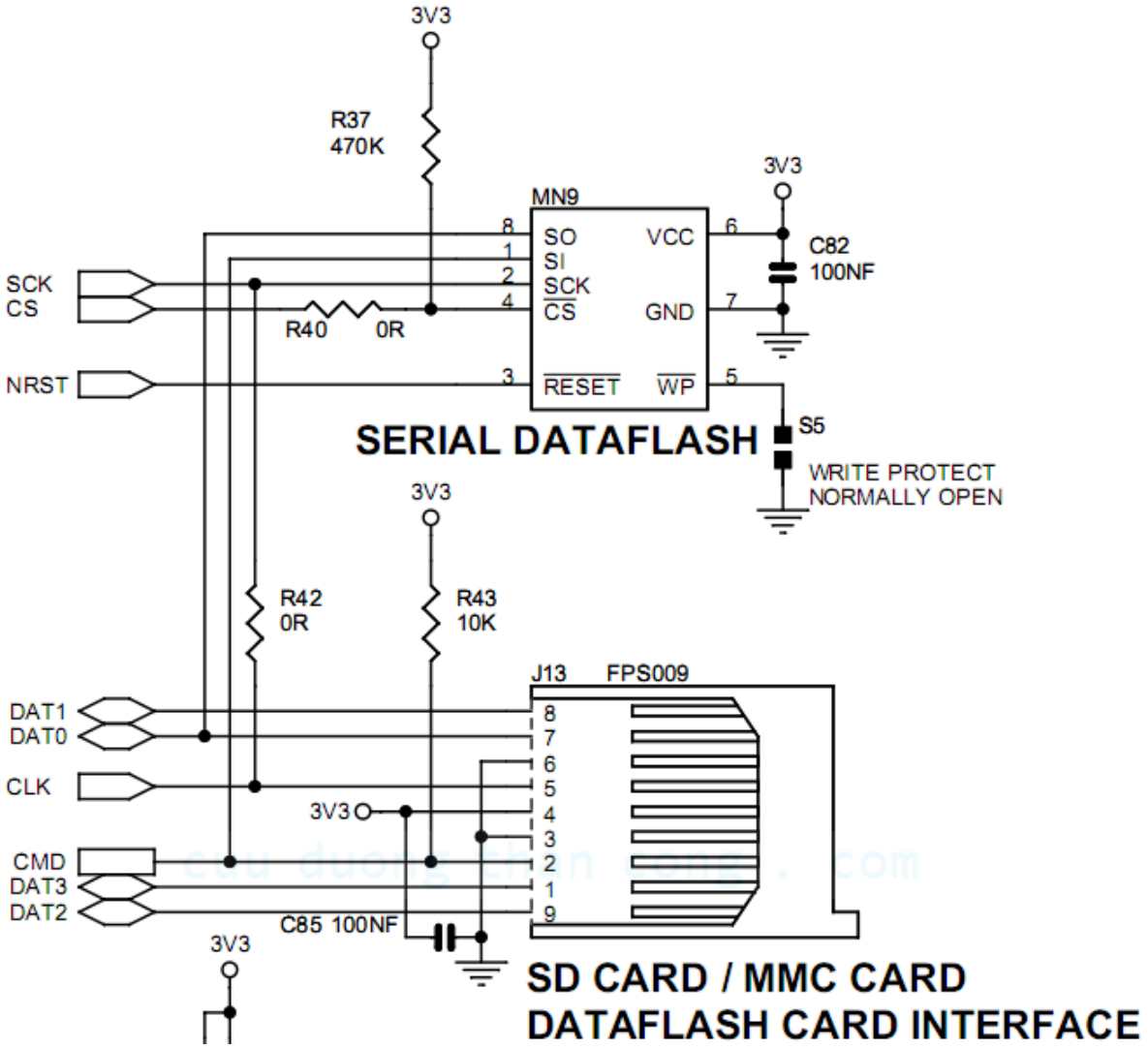
5.6 Bộ nhớ Serial DataFlash.

Serial DataFlash là các IC nhớ dạng Flash sử dụng giao tiếp nối tiếp SPI. Chuẩn giao tiếp SPI là chuẩn giao tiếp thông dụng nhất hiện nay nó cho phép truyền thông với tốc độ cao, Serial data Flash được sử dụng để lưu trữ chương trình khởi động cho hệ thống. Khi hoạt động chương trình khởi động này sẽ được bộ xử lý đọc và ghi lên SRAM sau đó mới thực thi. Trình khởi động có nhiều tên gọi khác nhau cho các hệ thống nhúng khác nhau, xong bạn có thể hình dung chúng giống như chương trình CMOS trên hệ thống máy tính mà bạn đang sử dụng. Trong các hệ thống nhúng sử dụng các vi điều khiển của Atmel, AMCC với Power PC thì là u-boot, hay Ciruss Logic thì sử dụng tên gọi Red boot, một số khác lại gọi tên là boot loader..

Hệ thống phát triển AT91SAM9260 sử dụng Serial Data Flash của Atmel, AT45DB041D, dung lượng 4 Mbit,

Một bộ nhớ nối tiếp cũng được sử dụng trong hệ thống đó là thẻ nhớ SDCard, thẻ nhớ có dung lượng lớn được sử dụng để lưu trữ hệ thống tập tin (File System) cho hệ thống nhúng.

CuuDuongThanCong.com https://fb.com/tailieudientucntt

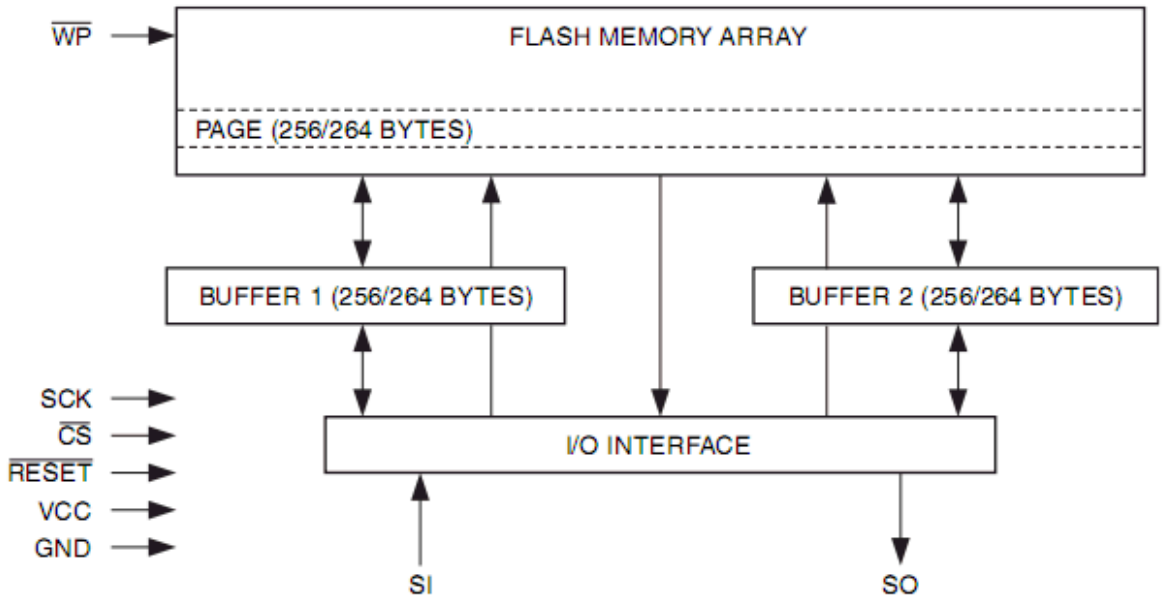


Hình 5.11 kết nối giao tiếp bộ nhớ nối tiếp

AT45DB041D có bộ nhớ Flash được chia thành nhiều trang, mỗi trang có thể cấu hình 256 byte hay 264 byte, tần số xung clock truy xuất lên đến 66Mhz sử dụng chuẩn giao tiếp SPI mode 0 và 3. Đặc biệt khác với các chip nhớ khác AT45DB041D có 2 vùng nhớ đệm SRAM 256 /264 byte, điều này giúp cho quá trình giao tiếp truyền dữ liệu nhanh hơn.

cuu duong than cong . com

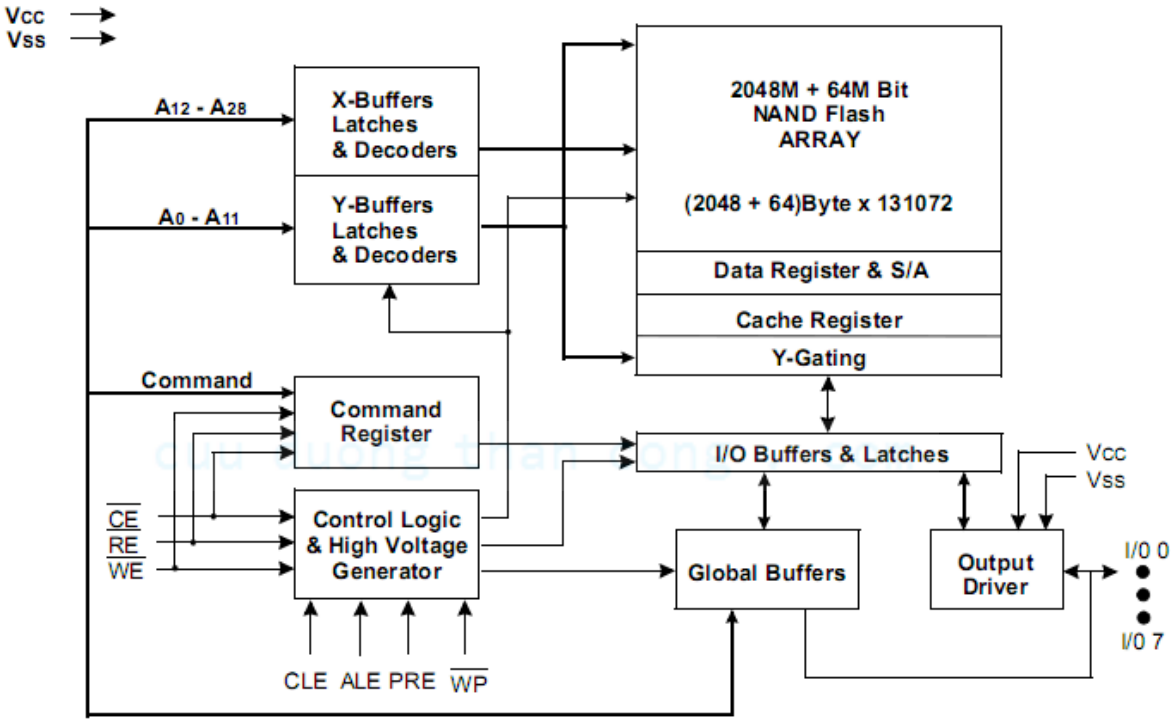
CuuDuongThanCong.com https://fb.com/tailieudientucntt



Hình 5.12 Cấu trúc bên trong bộ nhớ nối tiếp

5.7 Bộ nhớ NAND Flash

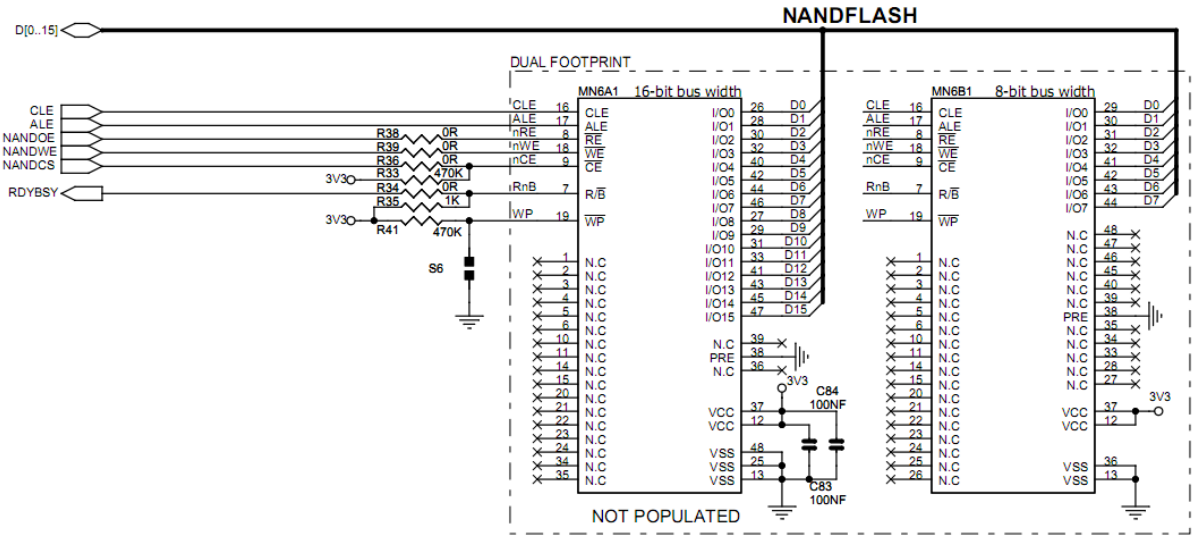
Nand Flash là một dạng của bộ nhớ Eeprom được sử dụng để lưu trữ ảnh của hệ điều hành, tuy nhiên khác với các bộ nhớ EEPROM thông thường, NAND Flash đa hợp các tín hiệu địa chỉ hàng và cột của ma trận các ô nhớ làm hạn chế các đường địa chỉ giao tiếp. Công nghệ chế tạo các NAND Flash sử dụng các MOSFET kết hợp tương tự các cổng Logic NAND cho phép tốc độ truy xuất cao. Hình dưới trình bày một sơ đồ khối của một NAND Flash.



Hình 5.13 Cấu trúc bên trong NAND Flash

Giao tiếp NAND Flash trên hệ thống nhúng:

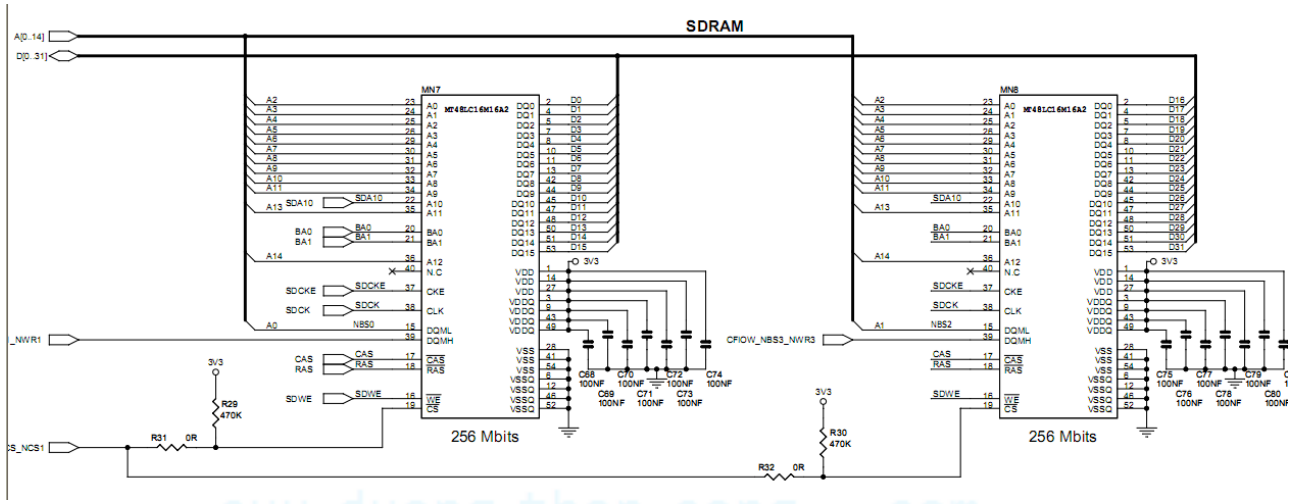
CuuDuongThanCong.com https://fb.com/tailieudientucntt



Hình 5.14 Giao tiếp NAND Flash

5.8 Bộ nhớ SRAM:

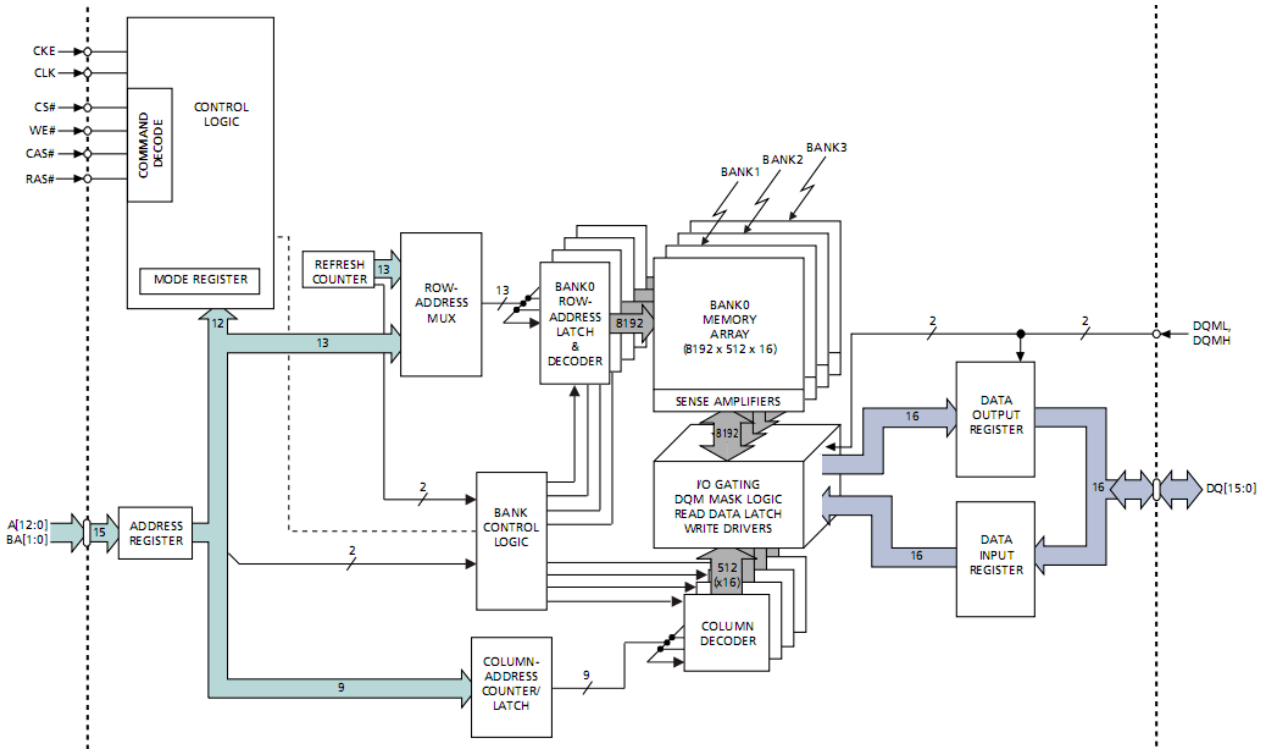
Bộ nhớ SRAM là bộ nhớ dữ liệu chính trong hệ thống nhúng, Các SRAM có dung lượng từ vài chục Mbyte tùy theo các hệ thống khác nhau, tuy nhiên bộ nhớ SRAM quyết định đến nhiều vấn đề về phần mềm trong hệ thống nhúng vì bất kỳ một ứng dụng nào trước khi thực thi đều được load xuống SRAM. Hệ thống phát triển hệ thống nhúng dựa trên vi điều khiển ARM9 AT91SAM9260 sử dụng SDRAM của Micron có dụng lượng 256Mb.



Hình 5.15 Giao tiếp SDRAM

CuuDuongThanCong.com https://fb.com/tailieudientucntt

Sơ đồ khối của một SDRAM được mô tả như sau:



Hình 5.16 Cấu trúc bên trong một SDRAM

cuu duong than cong . com

MT48LC16M16A2P là SDRAM 256Mb công nghệ CMOS, bên trong được thiết kế 4 bank DRAM, với giao tiếp đồng bộ.

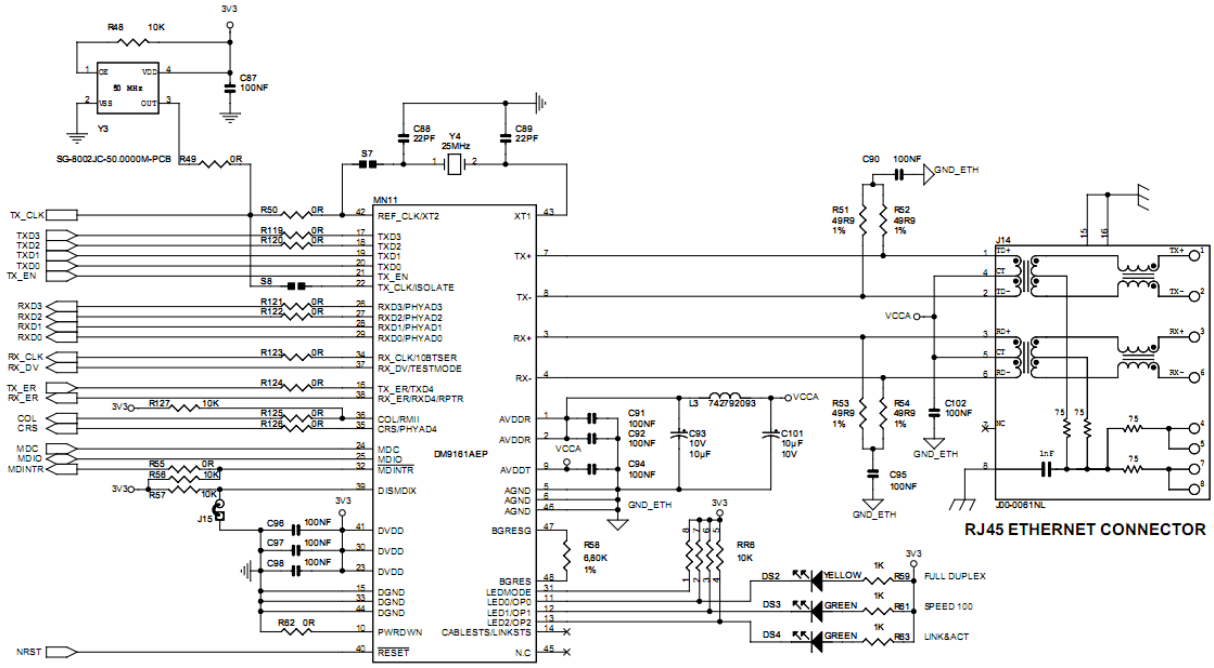
5.9 Giao tiếp ngoại vi:

Hệ thống nhúng thông thường được thiết kế với các ngoại vi phục vụ cho chức năng hệ thống nhúng như ADC, Graphic LCD, USB, PWM.. Tuy nhiên một số ngoại vi được thiết kế chung cho cả ứng dụng đồng thời hỗ trợ cho quá trình phát triển hệ thống.

Ethernet PHY DM9161A cho phép hoạt động 100BASE-TX , 10BASE-TX, giao tiếp mạng trực tiếp sử dụng cáp xoắn đôi UTP5 cho 100BASE –TX Fast Ethernet, UPT5/UTP3 cho 10BASE-TX ethernet. DM9161A đóng vai trò lớp vật lý trong quá trình giao tiếp mạng.

cuu duong than cong . com

CuuDuongThanCong.com https://fb.com/tailieudientucntt



Hình 5.17 Giao tiếp PHY Ethernet

Giao tiếp ethernet được sử dụng để load hệ điều hành, file System xuống hệ thống trong quá trình phát triển hệ thống, đồng thời còn sử dụng trong các ứng dụng mạng cho hệ thống nhúng.

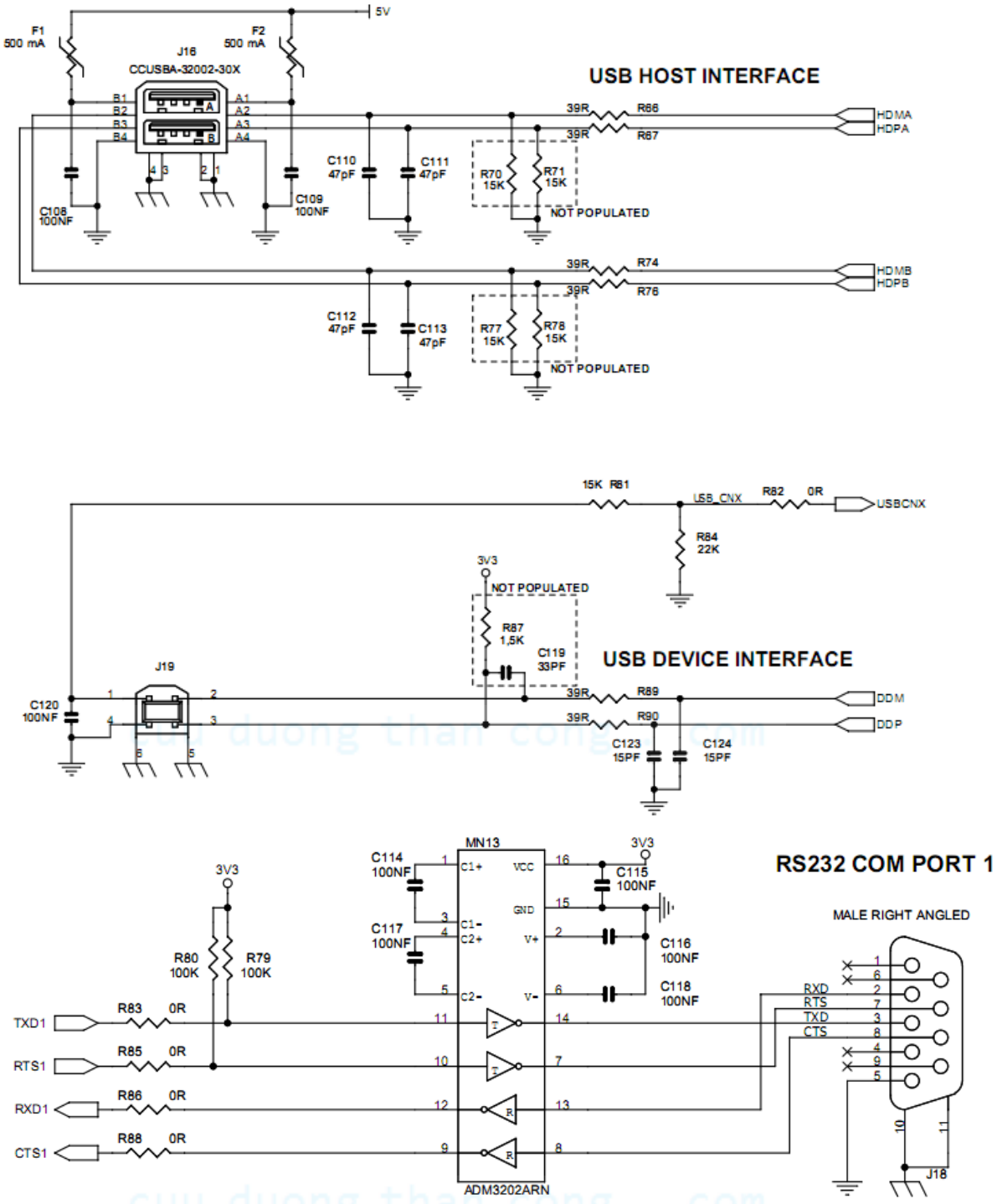
USB host và device: USB host cho phép xây dựng các ứng dụng giao tiếp USB như Camera...

USB device còn có nhiệm vụ giao tiếp máy chủ để load phần mềm xuống cho hệ thống như u-boot, bootstrap…

Serial Port hỗ trợ giao tiếp máy chủ để hiển thị thông tin trong quá trình hệ thống hoạt động.

cuu duong than cong . com

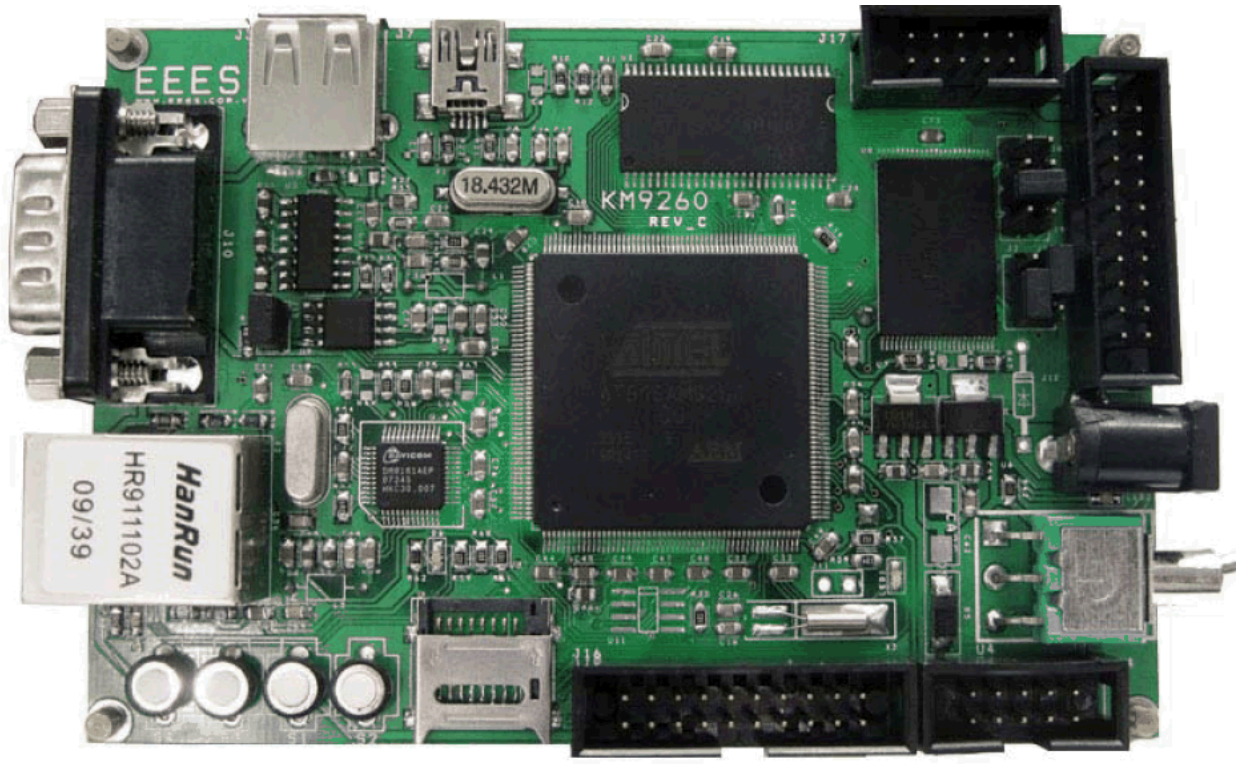
CuuDuongThanCong.com https://fb.com/tailieudientucntt



Hình 5.18 Giao tiếp USB và RS232

CuuDuongThanCong.com https://fb.com/tailieudientucntt

SPEC kit KM9260



Các thành phần trên board bao gồm:

No.	Name	Function
1	J3	USB Host Connector (Type A)
2	J7	USB Device Connector (Type Mini B)
3	U2	MT48LC16M16A2, SDRAM 256Mb (32MB) 133Mhz
4	J17	SCI Interface (I2S) Extension Connector
5	J10	BD9 Male Connector (DBU, RS232)
6	U8	K9F2G08UOM, NAND Flash (256MB)
7	J5	JTAG ICE Interface
8	U9	Serial dataflash (512kB)
9	U1	AT91SAM9260, 16/32 bit ARM926EJ-S 180Mhz
10	J12	5VDC Power Connector
11	U5	DM9161EA, Ethernet 10/100 Full-Duplex
12	RJ1	Integrated Transformer Ethernet Connector (RJ45)
13	S1, S2, S3, S4	Wake Up, Reset, User Button 1, User Button 2
14	U10	MicroSD Card Socket
15	U11	I2C EEPROM
16	J12	Power Switch
17	J16	Uart, TWI, ADC Extension Connector
18	J14	SPI Extension Connector

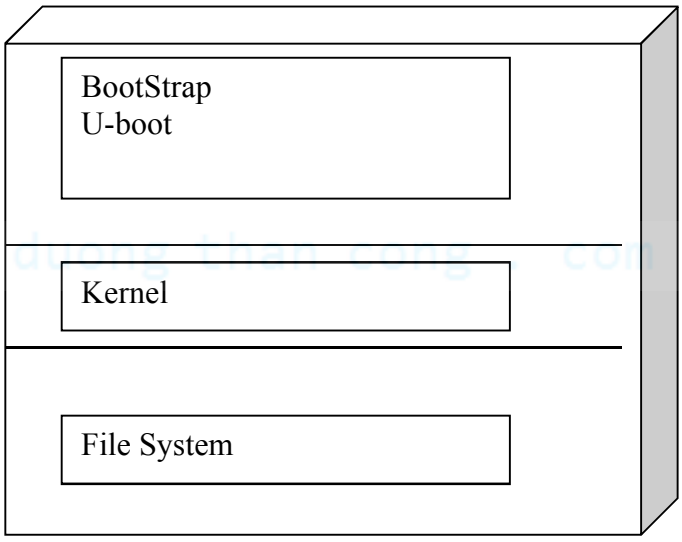
CuuDuongThanCong.com https://fb.com/tailieudientucntt

Chương 6
PHẦN MỀM HỆ THỐNG NHÚNG

6.1 Cấu trúc phần mềm trong hệ thống nhúng:

Tương tự như một hệ thống máy tính đơn giản, một hệ thống nhúng là một sự kết hợp chặt chẽ giữa phần cứng và phần mềm, khác với các hệ thống vi xử lý thông thường khác, hệ thống nhúng có một hệ thống phần mềm điều khiển phức tạp hơn. Chúng ta hãy hình dung hệ thống phần mêm trong một máy tính. Khi bạn mua một máy tính một phần mềm đã được cài đặt sẵn trong hệ thống đó là CMOS. Chương trình này làm nhiệm vụ giao tiếp giữa người sử dụng với phần cứng và thực hiện các thao tác tiếp theo. Sau đó bạn tùy thích cài hệ điều hành mà bạn muốn, sau khi có hệ điều hành các ứng dụng lại được lựa chọn để phục vụ nhu cầu của bạn. Phần mềm trong hệ thống nhúng cũng tương tự như một hệ thống máy tính. Hệ điều hành trong các hệ thống nhúng cũng tương đối phong phú. Hiện nay có nhiều hệ điều hành cho hệ thống nhúng như Window CE, Linux…Tuy nhiên Linux là được lựa chọn nhiều do mã nguồn mở và nhiều tính năng như đa nhiệm, đa người dùng… Ngòai ra Linux còn cho phép tái biên dịch nhân hệ điều hành trên ngay chính hệ điều hành. Việc phát triển hệ thống nhúng sử dụng hệ điều hành Linux được thực hiện trên một máy tính sử dụng hệ điều hành Linux, đây cũng là lý do mà các chương trước tài liệu đã hướng dẫn cáo thao tác và cách lập trình trên hệ thống Linux. Một hệ thống nhúng sử dụng hệ điều hành Linux thông thường có 3 lớp phần mềm như sau:

Boot Loader là tên gọi chung một phần mềm được cài đặt trước vào hệ thống, phần mềm này có nhiệm vụ khởi tạo hệ thống và thực hiện giao tiếp giữa hệ thống với người sử dụng. Có nhiều phiên bản cũng như nhiều loại **bootloader** khác nhau. Các bộ xử lý của Ciruss Logic như EP9315…thì sử dụng red boot, AMCC với Power PC và các ARM9 của Atmel thì sử dụng **u-boot**, một số bộ sử lý khác thì dùng luôn tên **Boot Loader**. Trong tài liệu này sẽ trình bày một tổ chức phần mềm trong một hệ thống nhúng sử dụng bộ xử lý ARM9 của Atmel. Trong các hệ thống nhúng sử dụng bộ xử lý của Atmel thì phần **boot loader** bao gồm 2 phần Bootstrap và **U-boot**, phần hệ điều hành Linux là kernel linux được cấu hình cho phù hợp, phần file system được sử dụng một số file system của linux như Amstrong, Debian...

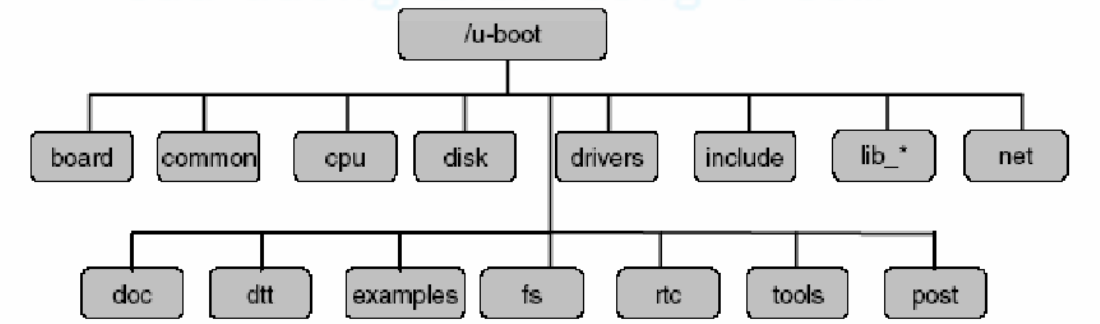


6.1.1 BootStrap và U- boot:

- **BootStrap** là một chương trình khởi động được nạp xuống trước tiên cho các vi điều khiển dòng ARM 9 của Atmel.

CuuDuongThanCong.com https://fb.com/tailieudientucntt

- **BootStrap là một module ứng dụng, nó được sử dụng để thực hiện các chức năng sau:**
  - Khởi tạo phần cứng như tần số xung clock, thiết lập các PIO (programmable Input Output).
  - Thiết lập các ngoại vi như PIO, PCM, SDRAMC...
  - Thực hiện các thuật toán truy xuất vật lý các ngoại vi như DataFlash, NANDFlash, Paralell Flash..
  - Điều khiển các tập tin hệ thống như JFFS2, FAT..
  - Thực thi các ứng dụng như ELF, Linux.
- **BootStrap** có thể được đặt trong vùng bootLoader, cụ thể là được đặt trong vùng DataFlash. **BootStrap** được chép lên RAM nội bởi trình **SAM-BA Boot**. **BootLoader** thực hiện khởi tạo vi xử lý (PLL, PIO, SDRAMC, SPI).
- **BootStrap** thực hiện load **U-boot** từ **DataFlash** lên SRAM và trỏ đến thực hiện chương trình **U-Boot**.
- **U-boot** (**universal bootLoader**) là một tập mã nguồn mở, hỗ trợ **bootLoader** cho nhiều kiến trúc nền khác nhau. **U-boot** hỗ trợ các lệnh tương tác, các biến môi trường, các lệnh thực thi và **boot** hệ thống từ các thiết bị media bên ngoài. **U-boot** hỗ trợ nhiều loại CPU và các họ CPU thông dụng hiện nay. **U-boot** hỗ trợ các **board** phát triển trên nền các vi xử lý thông dụng hiện nay.
- **U-boot** thực hiện cấu hình các khối phần cứng trong một **board** và đặt chúng vào trạng thái hoạt động. Nó có thể load và thực thi hệ điều hành một cách tự động (**auto-boot**) hoặc ngược lại nó cho phép người dùng khởi động hệ điều hành thông qua các lệnh giao tiếp mà **u-boot** hỗ trợ. Tập lệnh chuẩn của **u-boot** cung cấp khả năng cho phép người sử dụng thao tác trên bộ nhớ, mạng và nhiều thao tác khác khi hệ thống khởi động.
- Thông thường u-boot được đặt trong phân vùng đầu tiên của Flash, bắt đầu từ sector hay block nào được định nghĩa bởi vi xử lý. U-boot khởi tạo CPU và một vài phần cứng trên board, tạo một vài cấu trúc dữ liệu để cho kernel sử dụng và load nó lên phân vùng đầu tiên của bộ nhớ.
- Khi quyền điều khiển được chuyển đến cho **u-boot**, nó sẽ khởi tạo các ngắt và các thiết bị ngoại vi. Sau đó u-boot chờ nhập các lệnh từ người dùng. Nếu **u-boot** nhận được lệnh boot ảnh của **kernel** hoặc nếu nó được sử dụng để **boot kernel** trực tiếp thì **u-boot** sẽ giải nén kernel image, load **kernel** lên bộ nhớ và chuyển điều khiển đến **kernel**. Kernel sẽ thực thi mà không có sự tương tác với u-boot.
- **U-boot** cung cấp các hàm chuẩn để hiệu chỉnh quá trình khởi động và khởi tạo **kernel**. Thường thì nó cung cấp các thao tác dưới dạng các lệnh (**command-line).**
- **U-boot** có nhiều phiên bản, tuy nhiên từ phiên bản 1.3.4 trở đi thì mới hỗ trợ bộ xử lý AT91SAM9260 của Atmel.
- Cấu trúc thư mục của **U-boot.**



CuuDuongThanCong.com https://fb.com/tailieudientucntt

Nắm rõ kiến trúc các thành phần trong u-boot giúp người phát triển hệ thống biên dịch và cấu hình **u-boot** cho tương thích với các phần cứng khác nhau.

Thư mục board xác định nhiều kiến trúc nền khác nhau của các hãng khác nhau như Philip, AMCC, Atmel, davici, Cirus Logic... thư mục board cũng bao gồm các hàm khởi tạo các board. Các hàm này có thể gọi từ thư viện **lib_<arch>/board.c**

Thư mục **board / <boardname>** xác định thông tin chi tiết cho hệ thống, thư mục này chưa các tập tin khởi tạo cần thiết cho mỗi hệ thống bao gồm các tập tin như **asm_init.S, congif.mk, flash.c**

Thư mục **common** chứa tập tin định nghĩa các lệnh của **uboot**, các biến môi trường như **cmd_boot.c, cmd_date.c, environment, env.c, main.c...**

Thư mục **CPU** bao gồm các tập tin xác định các thông số, khởi tạo các ngắt, bộ nhớ đệm... cho CPU như **cpu.c, cpu_init.c, interrupts.c, cache.s, start.s...**

Thư mục **disk** chứa các tập tin phân vùng và thông tin thiết bị cho ổ đĩa.

Thư mục **driver** chứa các tập tin thiết bị như **ethernet, usb, sirial...**

Thư mục **include** bao gồm nhiều tập tin header như **console.h, version.h, usb.h, pci.h**. Trong tập tin **version.h** định nghĩa phiên bản của **u-boot #define u-boot_VERSION "xxx". Include/congifs** chứa các cấu hình cho nhiều board.

Thư mục **lib_generic** các thư viện chung như **bzlib.c vsprintf, string.c...**

Thư mục **net** chứa các tập tinh Ethernet như **eth.c, net.[ch], nfs.[ch], bootp.c [ch].**

Thư mục **fs** chứa các file hệ thống như **fat, fdos, jffs2**

Thư mục bao gồm các tập tin hỗ trợ chỉ đồng hồ thời gian thực **rtc date.c, mpc8xx.c, etc.**

Thư mục **tools** các thư mục và các tập tin hỗ trợ biên dịch, gỡ rối như **env, gdb, logos, scripts, mkimage.c**

Thư mục **post** bao gồm các tập tin và thư mục như **post.c codec.c, cache.c, memory.c, uart.c**, etc.

6.1.2 Biên dịch lại u-boot:

Sau khi thiết kế một hệ thống nhúng mới, quá trình biên dịch lại **u-boot** để thực thi trên hệ thống mới là hết sức cần thiết. Quá trình này hay con gọi bằng một thuật ngữ **porting. Porting u-boot** là quá trình tạo ra một u-boot với các thông số phù hợp cho bộ xử lý và các thiết bị ngoại vi. Một cách chung nhất để thực hiện vấn đề này là tạo ra từ các **u-boot** có sẵn từ các board có cấu hình gần giống với hệ thống mà bạn đang thiết kế.

Trước hết tạo một số tập tin và thư mục phục vụ cho việc cấu hình hệ thống:

u-boot/board/<boardname>

u-boot/include/configs/<boardname>.h

u-boot/Makefile

Chép các tập tin và thay đổi **makefile** và kiến trúc nền cho **board** như **include, cpu, lib_arch, và board**.

Trong thư mục **include,** tập tin **config/<boardname>.h** sẽ bao gồm các tập tin cấu hình phần cứng cho hệ thống như ánh xạ bộ nhớ và các ngoại vi. Cấu hình bộ xử lý, cấu hình bộ nhớ khởi động, cấu hình Nor hay Nand flash, cấu hình bộ nhớ SDRAM, các giao tiếp nối tiếp, ethernet và network...tập tin **<core>.h** như **arm926ejs.h** bao gồm các định nghĩa cần thiết cho bộ xử lý như địa chỉ các thanh ghi...

Trong thư mục **cpu** bao gồm các tập tin như **cpu.c** chưa các khai báo cho bộ xử lý, các hoạt động đọc và ghi IO, Reset CPU, cho phép hay không cho phép hoạt động của bộ nhớ đệm lệnh và bộ nhớ đệm dữ liệu, khởi tạo ngăn xếp cho các ngắt. Thư mục **interrupt.c** bao gồm các hàm phục vụ cho ngắt và timer, như cho phép ngắt hay không cho phép ngắt, các hàm liên quan

CuuDuongThanCong.com https://fb.com/tailieudientucntt

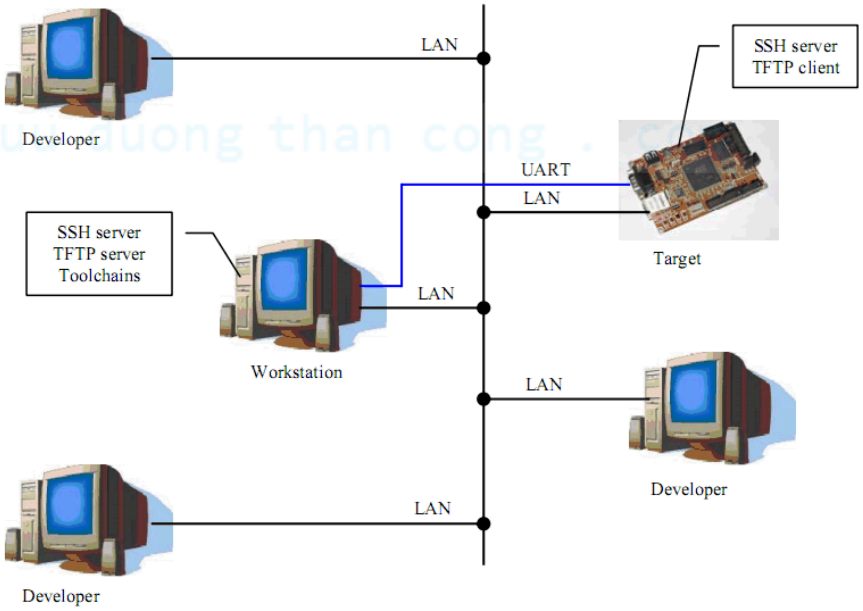
đến bộ định thời. Tập tin start.S là chương trình khởi động cho lõi của bộ xử lý. Tập tin < **boardname >/flash.c** chứa các hàm liên quan đến bộ nhớ flash như khởi tạo flash, các hàm reset, xóa, đọc và ghi bộ nhớ flash.

Sau khi tạo ra một tập mã nguồn của **u-boot**, bước cuối cùng là biên dịch **u-boot** tạo ra tập tin thực thi **u-boot.bin**.

Tài liệu sẽ trình bày các bước biên dịch **u-boot phiên bản 1.3.4** cho hệ thống nhúng sử dụng vi điều khiển **ARM91SAM9260** của Atmel.

Trước hết download mã nguồn **u-boot** dưới dạng tập tin nén về từ một số trang web như http://www.denx.de/wiki/U-Boot/SourceCode. bạn được file **u-boot-1.3.4.tar.bz2**

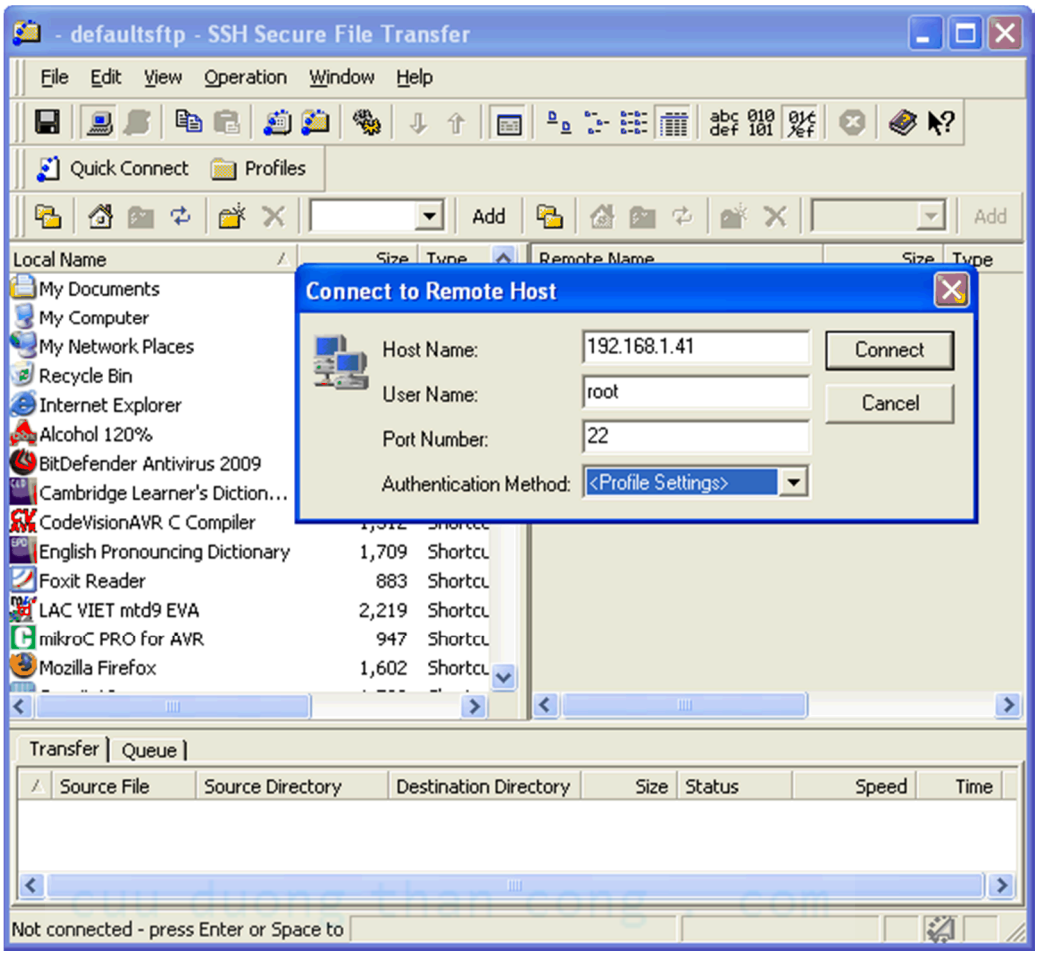
Chép vào thư mục trên máy Linux. Ví dụ **/sontruong**. Chúng ta có thể download mã nguồn trực tiếp trên một máy Linux kết nối với mạng internet, thực hiện các thao tác trên một máy tính chạy hệ điều hành Linux. Tuy nhiên trong thực tế thì nhiều nhà phát triển phần mềm lại dùng máy tính với hệ điều hành Windows XP. Vista. Điều này hoàn toàn thích hợp vì sự thông thạo cũng như quen thao tác trên hệ điều hành Windows. Linux chỉ có nhiệm vụ biên dịch mã nguồn mà thôi. Chính vì thế mà người ta thường kết nối một máy Linux với nhiều máy Windows để cùng lúc nhiều người có thể phát triển hệ thống nhúng, nhiều người có thể truy cập và máy Linux tại một thời điểm do Linux là một hệ điều hành đa nhiệm, đa người dùng. Các máy tính trên Windows có thể giao tiếp và truy xuất vào máy Linux thông qua một số phần mềm hỗ trợ như **SSH client..**



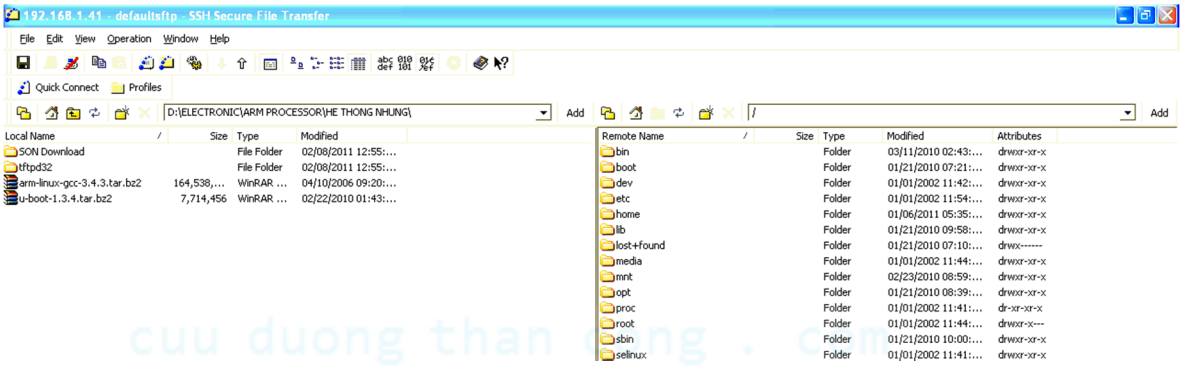
Trên máy tính sử dụng hệ điều hành windows, việc chép một tập tin từ giữa 2 máy tính được thực hiện một cách đơn giản thông qua việc kéo thả các tập tin, thư mục.

Sau khi cài đặt **SSH secure File Transfer**, thực hiện kết nối đến máy Linux thông qua tên **user**, địa chỉ IP, Port

CuuDuongThanCong.com https://fb.com/tailieudientucntt



Cửa sổ bên trái là các thư mục và tập tin trên máy Windows, bên phải là các tập tin và thư mục trên máy Linux



Giải nén gói mã nguồn **u-boot** bằng dòng lệnh: **Tar –xjvf u-boot-1.3.4.tar.bz2**

Ta thu được thư mục **u-boot-1.3.4** như hình dưới.

CuuDuongThanCong.com https://fb.com/tailieudientucntt

```
u-boot-1.3.4  u-boot-1.3.4.tar.bz2
[root@localhost sontruong]# cd u-boot-1.3.4
[root@localhost u-boot-1.3.4]# ls
api                          cpu           lib_blackfin    lib_sparc             nios_config.mk
api_examples                 CREDITS       libfdt          m68k_config.mk        onenand_ipl
arm_config.mk                disk          lib_generic     MAINTAINERS           post
avr32_config.mk              doc           lib_i386        MAKEALL               ppc_config.mk
blackfin_config.mk           drivers       lib_m68k        Makefile              README
board                        examples      lib_microblaze  microblaze_config.mk  rules.mk
CHANGELOG                    fs            lib_mips        mips_config.mk        sh_config.mk
CHANGELOG-before-U-Boot-1.1.5 i386_config.mk lib_nios      mkconfig              sparc_config.mk
common                       include       lib_nios2       nand_spl              tools
config.mk                    lib_arm       lib_ppc         net
COPYING                      lib_avr32     lib_sh          nios2_config.mk
[root@localhost u-boot-1.3.4]#
```

Biên dịch uboot cho ARM không sử dụng trình biên dịch **gcc** có sẵn trên máy tính cài linux. Trước hết bạn cần cài gói phần mềm biên dịch **arm-linux-gcc**. Để cài đặt gói phần mềm biên dịch, download gói phần mềm **arm-linux-gcc-3.4.3.tar.bz2** về và giải nén vào thư mục gốc của máy Linux.

Gói phần mềm biên dịch bạn có thể tải về từ trên mạng.

Trước khi biên dịch, cần thiết lập biến môi trường về đường dẫn cho trình biên dịch, ví dụ như sau:

PATH=/usr/local/arm/3.4/bin:$PATH

Trong đó thư mục **PATH=/usr/local/arm/3.4/bin** chứa các trình biên dịch như **arm-linux-gcc…**

Biên dịch uboot cho hệ thống:

Make clean: xóa hết các cấu hình biên dịch trước đó.

Make at91sam9260ek_config: khai báo biên dịch **uboot** cho hệ thống **board at91sam9260ek.**

```
[root@localhost u-boot-1.3.4]# PATH=/usr/local/arm/3.4/bin:$PATH
[root@localhost u-boot-1.3.4]# make clean
make: Warning: File `/sontruong/u-boot-1.3.4/board/atmel/at91sam9260ek/config.mk' has modification time
2.1e+08 s in the future
Generating include/autoconf.mk.dep
make: Warning: File `/sontruong/u-boot-1.3.4/board/atmel/at91sam9260ek/config.mk' has modification time
2.1e+08 s in the future
Generating include/autoconf.mk.dep
make: Warning: File `/sontruong/u-boot-1.3.4/board/atmel/at91sam9260ek/config.mk' has modification time
2.1e+08 s in the future
Generating include/autoconf.mk.dep
make: warning:  Clock skew detected.  Your build may be incomplete.
[root@localhost u-boot-1.3.4]# make at91sam9260ek_config
make: Warning: File `/sontruong/u-boot-1.3.4/board/atmel/at91sam9260ek/config.mk' has modification time
2.1e+08 s in the future
Generating include/autoconf.mk.dep
make: Warning: File `/sontruong/u-boot-1.3.4/board/atmel/at91sam9260ek/config.mk' has modification time
2.1e+08 s in the future
Generating include/autoconf.mk.dep
Configuring for at91sam9260ek board...
make: warning:  Clock skew detected.  Your build may be incomplete.
[root@localhost u-boot-1.3.4]#
```

Make all

Nếu quá trình biên dịch thành công trong thư mục **u-boot-1.3.4** sẽ tồn tại file **u-boot.bin**

6.1.3 Các thao tác trên u-boot:

CuuDuongThanCong.com https://fb.com/tailieudientucntt

- Sau khi **U-boot** đã được load vào hệ thống. **Bootstrap** thực thi, khởi tạo các thông số cho b, **load uboot** và thực thi **uboot**. Khi **uboot** thực thi, **u-boot command** cho phép người sử dụng có thể giao tiếp với hệ thống thông qua các lệnh **u-boot** hỗ trợ.
- Trước hết cần thiết lập các thông số môi trường cho hệ thống.
- Để thiết lập các thông số môi trường chúng ta sử dụng **command setenv** như sau:
  - Thiết lập địa chỉ **IP** cho **board**: setenv ipaddr <địa chỉ IP cho board>
  - Thiết lập địa chỉ **IP** cho **host:** setenv serverip <địa chỉ host>
  - Thiết lập địa chỉ **ethernet**(MAC): setenv ethaddr <MAC address>
  - Thiết lập mặt nạ: setenv netmask <submask net>
  - Lưu lại các biến môi trường vào **dataflash**: save

  Để xem lại các biến môi trường tại dòng lệnh bạn nhập lệnh **prinenv**

```
U-Boot>
U-Boot> printenv
bootdelay=3
baudrate=115200
ethact=macb0
bootcmd=nand read 0x20000000 0x0 0x200000; bootm 0x20000000
ipaddr=192.168.1.35
serverip=192.168.1.34
ethaddr=00:11:22:33:44:55
netmask=255.255.255.0
bootargs=console=ttyS0,115200 root=/dev/mmcblk0p1 rootdelay=5
stdin=serial
stdout=serial
stderr=serial

Environment size: 294/16892 bytes
U-Boot>
```

- Chuẩn bị **kernel** cho hệ thống.
- Tại **command line uboot** của:

  Xóa vùng nhớ NAND FLASH để chuẩn bị cho kernel:

- nand erase offset length
- nand erase 0x0 0x200000
- Lệnh trên cho phép xóa 2 Mbyte bộ nhớ **Nand Flash** ở địa chỉ offset là 0. (vùng nhớ đầu tiên của **NandFlash**)

  Chép uImage từ máy tính vào SRAM ở địa chỉ 0x20000000

- Tftp 0x20000000 uImage
- Để có thể chép thông qua giao thức **tftp** thì trên máy tính **host** phải cài đặt và chạy dịch **vụ tftp**, **file uImage** được lưu trong thực mục của **tftp server**.
- **U-boot** sẽ dò trong biến môi trường xem địa chỉ của **server** là bao nhiêu và nó sẽ lên **sever** này tìm file có tên **uImage** và chép vào bộ nhớ SRAM từ địa chỉ 0x20000000.

  Thực thi kernel

- Bootm 0x20000000
- Hệ thống sẽ chuyển đến địa chỉ SRAM 0x20000000 để boot Umage:
  - Trong trường hợp này lần sau khi mở điện nội dung trên SRAM sẽ mất đi. Để có thể sử dụng uImage cho các lần khởi động sau chúng ta cần chep **uImage** lên **NandFlask.**
  - **Nand write** 0x20000000 0x0 0x200000

CuuDuongThanCong.com https://fb.com/tailieudientucntt

- Chép **2 Mbyte** từ địa chỉ SRAM 0x20000000 lên nandflash có địa chỉ **offset** là 0 (vùng đầu tiên của **NandFlash**)
- Các lần khởi động sau sẽ chép **uImage** từ **Nanflash** xuống SRAM và thực thi nó trên SRAM
- **Nand read** 0x20000000 0x0 0x200000
- Bạn chú ý các thao tác chép **uImage** từ **Nandflash** xuống SRAM và thực thi nó được thiết lập trong biến môi trường **uboot** để thực hiện một cách tự động.

Bảng tổng hợp các lệnh của u-boot:

autoscr	run script from memory
base	print or set address offset
bdinfo	print Board Info structure
bootm	boot application image from memory
bootp	boot image via network using BootP/TFTP protocol
bootd	boot default, i.e., run 'bootcmd'
cmp	memory compare
cp	memory copy
crc32	checksum calculation
echo	echo args to console
erase	erase FLASH memory
flinfo	print FLASH memory information
go	start application at address 'addr'
help	print online help
iminfo	print header information for application image
loadb	load binary file over serial line (kermit mode)
loadc	load binary file over serial line (ymodem-c mode)
loadg	load binary file over serial line (ymodem-g mode)
loads	load S-Record file over serial line
loop	infinite loop on address range
md	memory display
mm	memory modify (auto-incrementing)
mtest	simple RAM test
mw	memory write (fill)
nm	memory modify (constant address)
printenv	print environment variables
protect	enable or disable FLASH write protection
rarpboot	boot image via network using RARP/TFTP protocol
reset	perform RESET of the CPU
run	run commands in an environment variable
saveenv	save environment variables to persistent storage
setenv	set environment variables
sleep	delay execution for some time

CuuDuongThanCong.com https://fb.com/tailieudientucntt

tftpboot	boot image via network using TFTP protocol and env variables ipaddr and serverip
version	print monitor version
? alias for 'help'	

6.2 Biên dịch nhân hệ điều hành Linux cho hệ thống:

Hệ thống nhúng sử dụng bộ xử lý ARM9 của Atmel sử dụng hệ điều hành Linux có phiên bản từ 2.6.27. Biên dịch nhân hệ điều hành cho hệ thống là quá trình thay đổi cấu hình của nhân hệ điều hành Linux sao cho tương thích với hệ thống, quá trình này thông thường một phần được thực hiện bởi các nhà sản xuất, một phần được thực hiện bởi các nhà phát triển hệ thống. Download phiên bản **Linux 2.6.27** về máy tính dưới dạng gói tập mã nguồn **linux-2.6.27.tar.bz2**. Copy gói mã nguồn Linux vào máy Linux để thực hiện biên dịch. Trước khi biên dịch, download trình biên dịch **cross Compiler** về và cài vào máy Linux.

arm-2009q3-67-arm-none-linux-gnueabi-i686-pc-linux-gnu.tar.bz2 chứa các trình biên dịch cần thiết phụ vụ cho quá trình biên dịch nhân hệ điều hành Linux. Đổi tên gói lại thành **arm-kernel_compiler.tar.bz2**.

Giải nén gói vừa mới đổi tên tar –xjvf arm-kernel-compiler.tar.bz2

Trước khi tiến hành biên dịch ta phải khai báo đường dẫn đến trình biên dịch, nếu quá trình này bị bỏ qua, trình make sẽ không tìm thấy trình biên dịch để gọi, chương trình bạn sẽ bị lỗi. Giả sử thư mục arm-kernel-compiler được bạn chép vào thư mục /sontruong. Sau khi biên dịch thư mục arm2009q3 được tạo ra. Như vậy các trình biên dịch nằm trong thư mục:

/sontruong/arm2009q3/bin

Tiến hành thêm đường dẫn đến trình biên dịch như sau:

Export PATH=$PATH:/sontruong/arm2009q3/bin

uboot-mkimage.tar.bz2 là chương trình tạo ra ảnh của nhân hệ điều hành được chép vào trong thư mục máy Linux, quá trình cài đặt có thể được tiến hành như sau:

tar –jxvf uboot-mkimage.tar.bz2

cd uboot-mkimage

make clean

make

chmod 777 mkimage

cp mkimage /bin

Giải nén linux-2.6.26.tar.bz2

tar –jxvf kernel-2.6.27.tar.bz2

#cd linux-2.6.27

```
[root@localhost sontruong]# ls
arm-2009q3                  linux-2.6.27.tar.bz2  uboot-mkimage
arm-kernel-compiler.tar.bz2  u-boot-1.3.4          uboot-mkimage.tar.bz2
linux-2.6.27                u-boot-1.3.4.tar.bz2
[root@localhost sontruong]# cd linux-2.6.27
[root@localhost linux-2.6.27]# ls
arch     crypto         fs       Kbuild       Makefile  REPORTING-BUGS  sound
block    Documentation  include  kernel       mm        samples         usr
COPYING  drivers        init     lib          net       scripts         virt
CREDITS  firmware       ipc      MAINTAINERS  README    security
[root@localhost linux-2.6.27]#
```


CuuDuongThanCong.com https://fb.com/tailieudientucntt

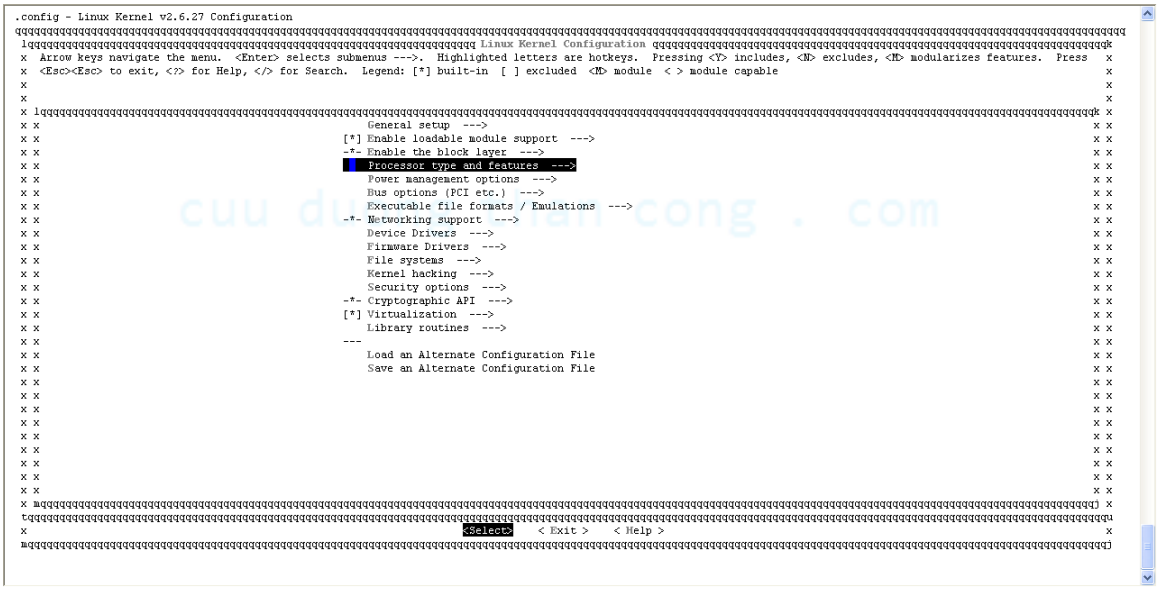
Thay đổi các cấu hình của hệ thống, sử dụng: #make menuconfig

```
.config - Linux Kernel v2.6.27 Configuration
 lqqqqqqqqqqqqqqqqqqqqqqq Linux Kernel Configuration qqqqqqqqqqqqqqqqqqqqqqk
 x  Arrow keys navigate the menu.  <Enter> selects submenus --->.        x
 x  Highlighted letters are hotkeys.  Pressing <Y> includes, <N>         x
 x  excludes, <M> modularizes features.  Press <Esc><Esc> to exit, <?>   x
 x  for Help, </> for Search.  Legend: [*] built-in  [ ] excluded        x
 x lqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqk x
 x x     General setup  --->                                             x x
 x x    [*] Enable loadable module support  --->                         x x
 x x    -*- Enable the block layer  --->                                 x x
 x x        Processor type and features  --->                            x x
 x x        Power management options  --->                               x x
 x x        Bus options (PCI etc.)  --->                                 x x
 x x        Executable file formats / Emulations  --->                   x x
 x x    -*- Networking support  --->                                     x x
 x x        Device Drivers  --->                                         x x
 x x        Firmware Drivers  --->                                       x x
 x x        File systems  --->                                           x x
 x x        Kernel hacking  --->                                         x x
 x x        Security options  --->                                       x x
 x x    -*- Cryptographic API  --->                                      x x
 x x    [*] Virtualization  --->                                         x x
 x x        Library routines  --->                                       x x
 x x    ---                                                              x x
 x x        Load an Alternate Configuration File                         x x
 x x        Save an Alternate Configuration File                         x x
```

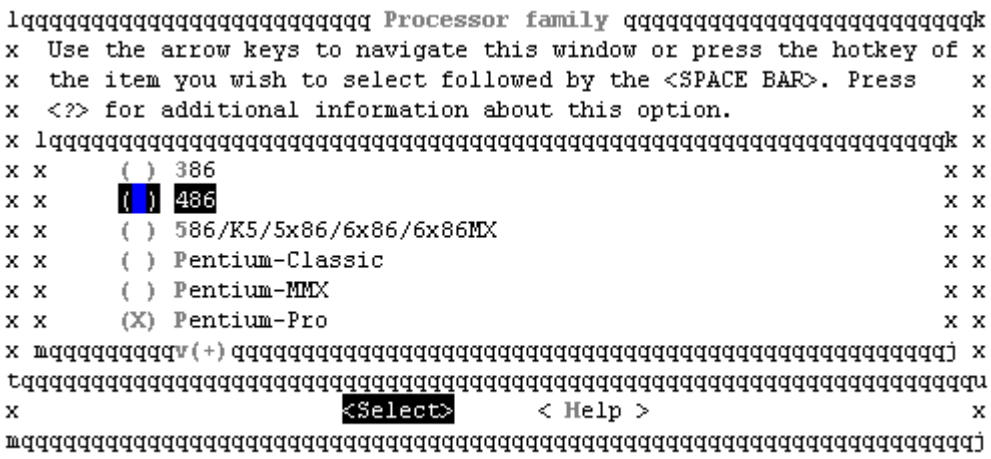
Một menu hiện ra cho phép thiết lập các tùy chọn thích hợp như bộ xử lý, bộ nhớ SDRAM, Flash, ethernet, giao tiếp các ngoại vi. Sau khi thực hiện cấu hình thích hợp chọn **save** để lưu lại.

Việc chọn các cấu hình trong cửa sổ **menuconfig** tùy thuộc vào phần cứng hệ thống mà chúng ta đang thiết kế, nó đòi hỏi nhiều kiến thức về thiết kế phần cứng. Tài liệu không đi trình bày chi tiết các tùy chọn. Người biên dịch phần mềm đòi hỏi phải nắm các thiết kế phần cứng hệ thống để cấu hình cho đúng.

Ví dụ cấu hình cho bộ xử lý.



CuuDuongThanCong.com https://fb.com/tailieudientucntt



Trong menu hệ thống chưa có loại vi xử lý tương thích với hệ thống bạn, điều này cũng dễ hiểu, Linux được thiết kế hỗ trợ nhiều phần cứng khác nhau, bạn muốn cấu hình Linux cho một hệ thống riêng của bạn đòi hỏi bạn phải có kiến thức thật tốt về hệ thống và hệ điều hành để làm được điều đó. Bạn phải tạo ra các tập tin để Linux hiểu hệ thống của bạn khi biên dịch. Đối với hệ thống nhúng sử dụng ARM9 của Atmel, nhà sản xuất đã làm sẵn cho bạn các thay đổi bằng cách tạo ra các bảng vá nhân hệ điều hành linux.

Download các bản vá

2.6.xx-at91.patch.gz , 2.6.xx-at91-exp.patch.gz

cuu duong than cong . com

Chạy các bản vá

zcat 2.6.xx-at91.patch.gz | patch -p1

zcat at91-exp. patch.gz | patch -p1

Download tập tin cấu hình at91sam926yek_defconfig

Chép tập tinh cấu hình vào thư mục **config** trong thư mục Linux. Sau đó thực thi **config**

cp at91sam926yek_defconfig .config

make ARCH=arm oldconfig

Trong quá trình thực hiện **oldconfig**, trình biên dịch sẽ yêu cầu bạn xác nhận một số thống số cho hệ thống.

Thực thi menuconfig

make ARCH=arm menuconfig

Lúc này **menuconfig** xuất hiện cho phép bạn tiến hành các cài đặt cho hệ thống.

cuu duong than cong . com

CuuDuongThanCong.com https://fb.com/tailieudientucntt

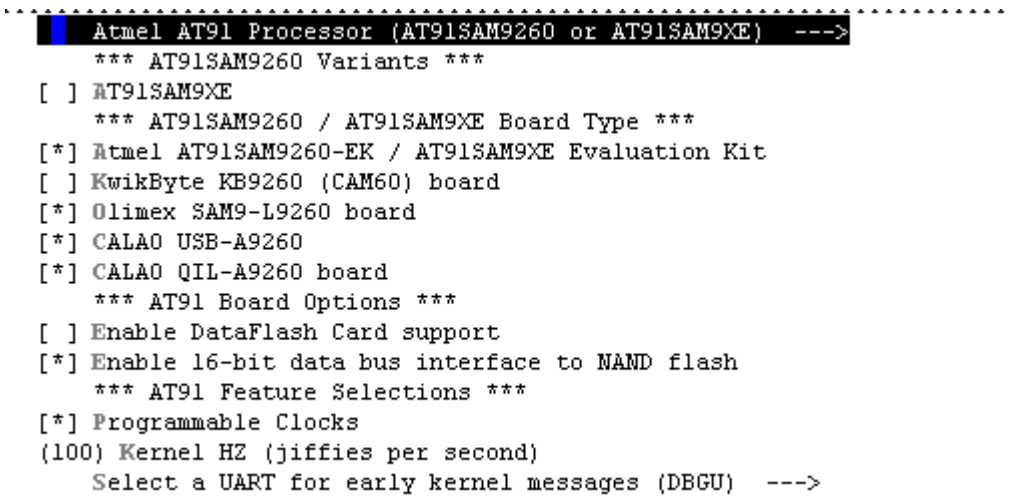
```
    General setup  --->
[*] Enable loadable module support  --->
-*- Enable the block layer  --->
    System Type  --->
    Bus support  --->
    Kernel Features  --->
    Boot options  --->
    Floating point emulation  --->
    Userspace binary formats  --->
    Power management options  --->
[*] Networking support  --->
    Device Drivers  --->
    File systems  --->
    Kernel hacking  --->
    Security options  --->
< > Cryptographic API  --->
    Library routines  --->
---
    Load an Alternate Configuration File
    Save an Alternate Configuration File
```

Trong mục chọn **System type**:

```
    ARM system type (Atmel AT91)  --->
    *** Boot options ***
    *** Power management ***
    Atmel AT91 System-on-Chip  --->
    *** Processor Type ***
[*] Support ARM926T processor
    *** Processor Features ***
[ ] Support Thumb user binaries
[ ] Disable I-Cache (I-bit)
[ ] Disable D-Cache (C-bit)
[ ] Force write through D-cache
[ ] Round robin I and D cache replacement algorithm
```

cuu duong than cong . com

CuuDuongThanCong.com https://fb.com/tailieudientucntt



Dựa vào thiết kế hệ thống, chúng ta có thể thay đổi một số thông số dựa vào việc chọn các menu trong **menuconfig**.

Biên dịch Linux image

make ARCH=arm CROSS_COMPILE=arm-none-linux-gnueabi- uImage

Sau khi biên dịch thành công, tập tin ảnh của nhân hệ điều hành được tạo ra trong thư mục **arch/ arm/ boot**.

Trong phần này tài liệu trình bày một số khái niệm về phần mềm hệ thống nhúng, các bước biên dịch **u-boot** cho hệ thống nhúng, một số thao tác trên môi trường **uboot**. Các bước thao tác để biên dịch nhân hệ điều hành. Phần **File System** chúng ta có thể tùy chọn tùy thuộc vào một số đặc điểm của hệ thống như bộ nhớ, tốc độ xử lý. Việc phát triển trên hệ thống nhúng rất đa dạng, người phát triển tùy thuộc vào mục đích và yêu cầu của hệ thống có thể thay đổi trong **u-boot**, cũng có thể thay đổi các tập tin điều khiển thiết bị trong **kernel**, hoặc có thể xây dựng các chương trình giống như các ứng dụng chạy trên hệ điều hành. Tuy nhiên cho dù bạn phát triển hệ thống nhúng ở mức độ nào thì quá trình biên dịch, các kiến thức về **u-boot**, kernel luôn là cần thiết.

cuu duong than cong . com

CuuDuongThanCong.com https://fb.com/tailieudientucntt

Ví dụ thiết kế bài tập giao diện dòng lệnh trên kit KM9260

Phần 1: viết code chương trình hello và biên dịch trên linux

1. Cài đặt trình biên dịch chéo trên linux
2. Khai báo đường dẫn cho trình biên dịch

PATH=$PATH:/root/km9260/arm-2009q3/bin/
3. Biên dịch chương trình
arm-none-linux-gnueabi-gcc -o hello hello.c

Phần 2: chép code qua kit

Bước 1:

Đặt địa chỉ trên PC(window): 192.168.1.2 netmask 255.255.255.0

default gw 192.168.1.1

Đặt địa chỉ trên kit:

ifconfig eth0 192.168.1.60 netmask 255.255.255.0

route add default gw 192.168.1.1 eth0

Trong trường hợp máy chưa thiết lập MAC ta vào U-Boot thiết lập
U-Boot> setenv ipaddr 192.168.1.35 (ip của board)
U-Boot> setenv serverip 192.168.1.34 (**trùng ip của máy tính)**
U-Boot> setenv ethaddr 00:11:22:33:44:55
U-Boot> setenv netmask 255.255.255.0
U-Boot> save

Bước 2: Nạp **hello** xuống Kit bằng lệnh TFTP
Trên máy tính **serser** mở tftpd32 và browse tới thư mục chứa tệp tin **hello**
Trên màn hình putty đang ở màn hình root của Linux của KIT SAM9260-EK ta đánh lệnh sau:
tftp -g -l / -l hello 192.168.1.2 ở đây IP của máy server
S3: Gán thuộc tính thực thi (lệnh trên putty)
chmod 777 hello
S4: Chạy chương trình trên board (lệnh trên putty)

./hello

CuuDuongThanCong.com https://fb.com/tailieudientucntt

Cách 2:

1. Cài đặt trình biên dịch chéo trên linux
2. Khai báo đường dẫn cho trình biên dịch

PATH=$PATH:/root/km9260/arm-2009q3/bin/
3. Biên dịch chương trình
arm-none-linux-gnueabi-gcc -o hello hello.c

3. Chép file sang kit

scp hello root@192.168.1.60:/home/root/baitap/

4. truy cập kit:

=> ssh 192.168.1.60
User: root; pass: 1234567
Chú ý:
nếu ssh yêu cầu add key:
=>tạo file config trong thu muc:~/.ssh/
nội dung file config:
Host 192.168.1.*
StrictHostKeyChecking no
UserKnownHostsFile=/dev/null

cuu duong than cong . com

cuu duong than cong . com

CuuDuongThanCong.com https://fb.com/tailieudientucntt

Chương 7
LẬP TRÌNH GUI SỬ DỤNG QT

Exercises Lecture An introduction to Qt

Install the Qt SDK

Visit http://qt-project.org/downloads , and download the LGPL version of Qt SDK suitable for your system. Then proceed to install the SDK. This gives you a Qt Creator icon on your desktop and in your program menu. Please refer to the platform specific tips and tricks below if you are experiencing any issues with this.

Windows

If altering the installation path of the Qt SDK, make sure to use a path without spaces. Spaces can in some situations confuse the build tool-chain.

Mac OS X

Installation should be straight forward.

Before you can develop Qt applications, you must make sure you have installed the developer tools from the DVD that came with your Mac.

Unix/Linux X11

cuu duong than cong . com

Before you can develop Qt software, you must ensure that you have a C++ toolchain installer. This usually means GCC with support for C++. This comes as a part of your distribution and you must install it to be able to use the SDK.

If your Linux distribution has a prepackaged version of Qt Creator and the Qt development libraries, it is often better to use those libraries. This course has been developed with Qt 4.6+ and Qt Creator 1.2+ in mind.

If you choose to use the installer downloaded from the web, you must make the installer executable before you can run it. You can either do this by setting the executable bit (sometimes just X-bit) of the file through your file manager, or by running chmod from a command prompt.

```
chmod u+x qt-sdk-linux-*.bin
```

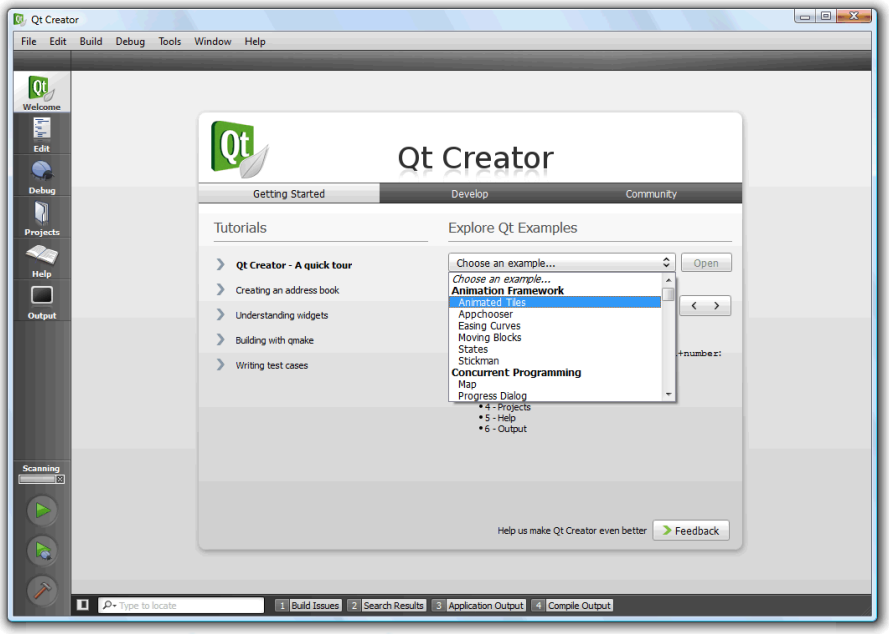
When running the installer, make sure that you have write access to the directory where you chose to install the SDK.

cuu duong than cong . com

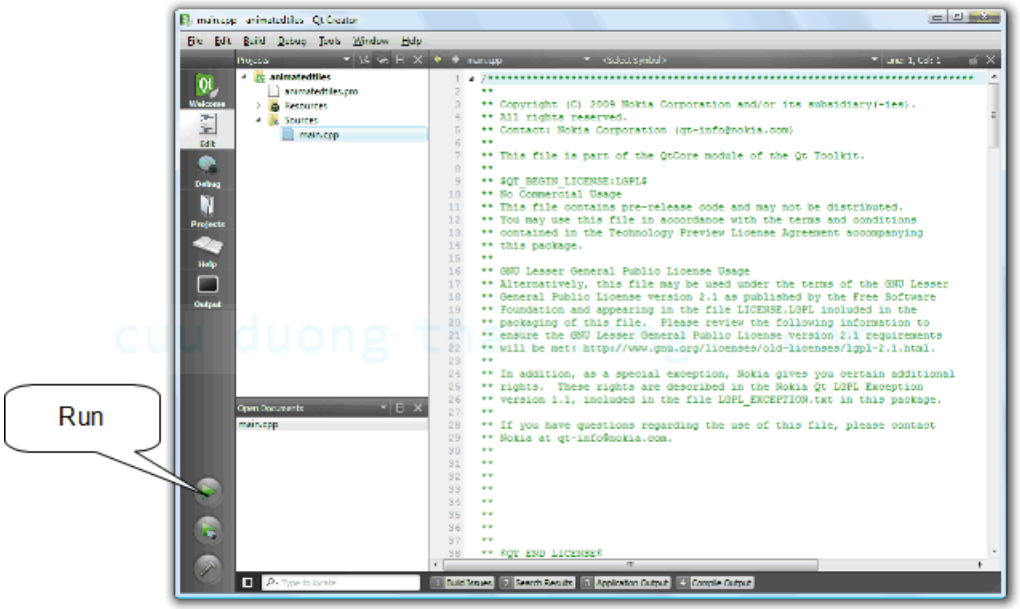
CuuDuongThanCong.com https://fb.com/tailieudientucntt

Testing the tool-chain

To ensure that the SDK has been properly installed, please try to build and debug some of the examples shipped with Qt. On the welcome screen of Qt Creator, make sure that you are on the “Getting Started” page shown below. From the list of Qt examples, pick the “Animated Tiles” example from the “Animation Framework” group.



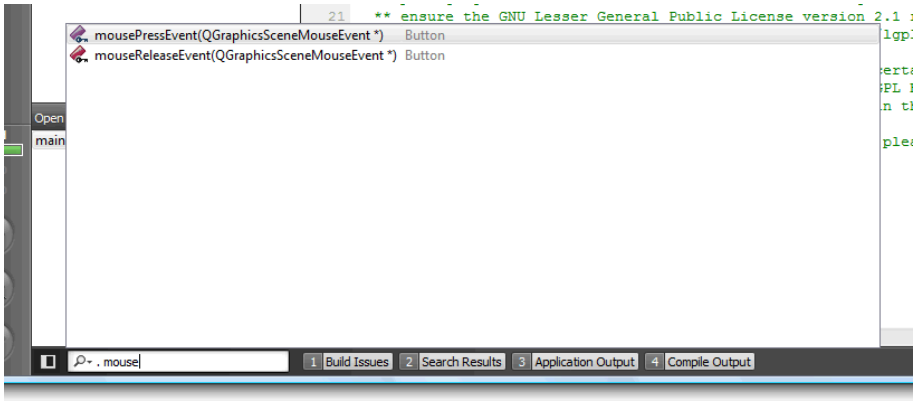
Having opened the project, ensure that you are in *edit mode* (Ctrl+2), then look for main.cpp in the Sources folder. Understanding the source code is not the purpose of this exercise, so leave that for later. Instead, try pressing the *run button* (Ctrl+R) to start the example. This will trigger Qt Creator to first build the example, then run it. When the example has been built, you should see a window with the demo running.



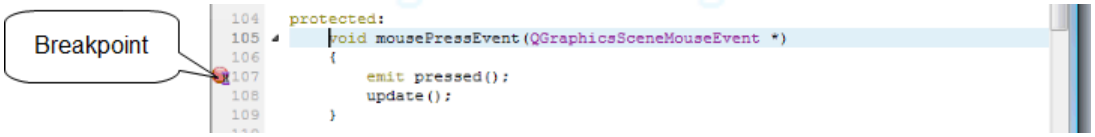
CuuDuongThanCong.com https://fb.com/tailieudientucntt

Having run the project, you have verified that the compiler and linker work. The next step is to test the debugging features.

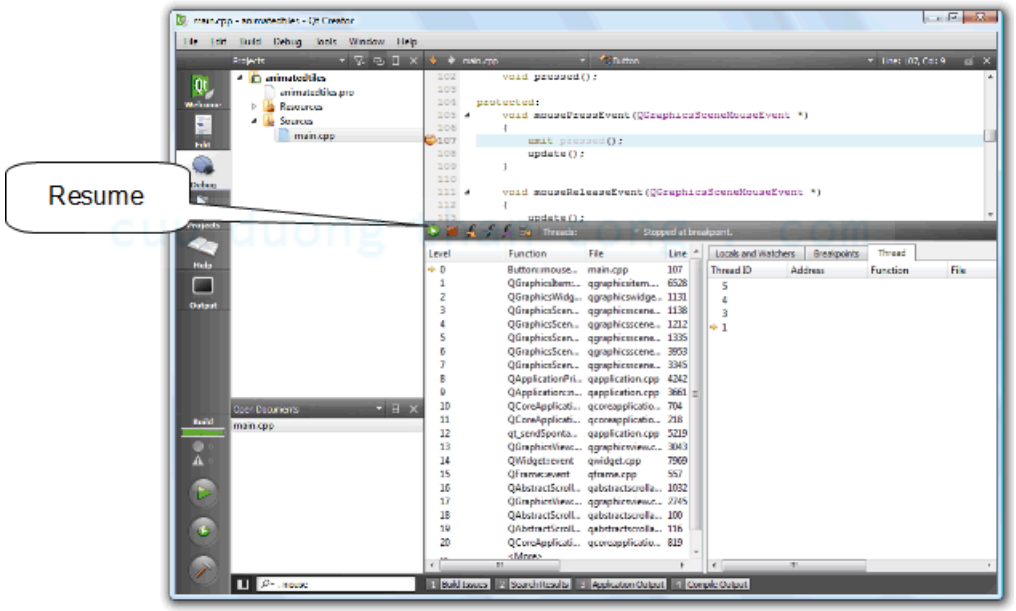
To find a point in the code to insert a break point at, we will use the *locator*. The locator lets you search within your project. It can be used to look for classes, methods and files within or outside your project. In this case, you are looking for the mousePressEvent in the main.cpp file. To locate a method, start your search with a dot followed by a space ". ". Then start typing mousePressEvent until you find the method in the list, then press *enter* to go there.



Having located the Button::mousePressEvent in the source code, you will see something like the image below. Try setting a break point as shown below (by clicking just left of the line number or right clicking on the line number and picking "Set Breakpoint" from the pop-up menu).



To start debugging click the debugging button (F5) just below the run button. This will start a debugging session. The example program will start as expected, but as soon as you click one of the buttons, the program execution will break and the editor will show the current callstack, breakpoint and so on.



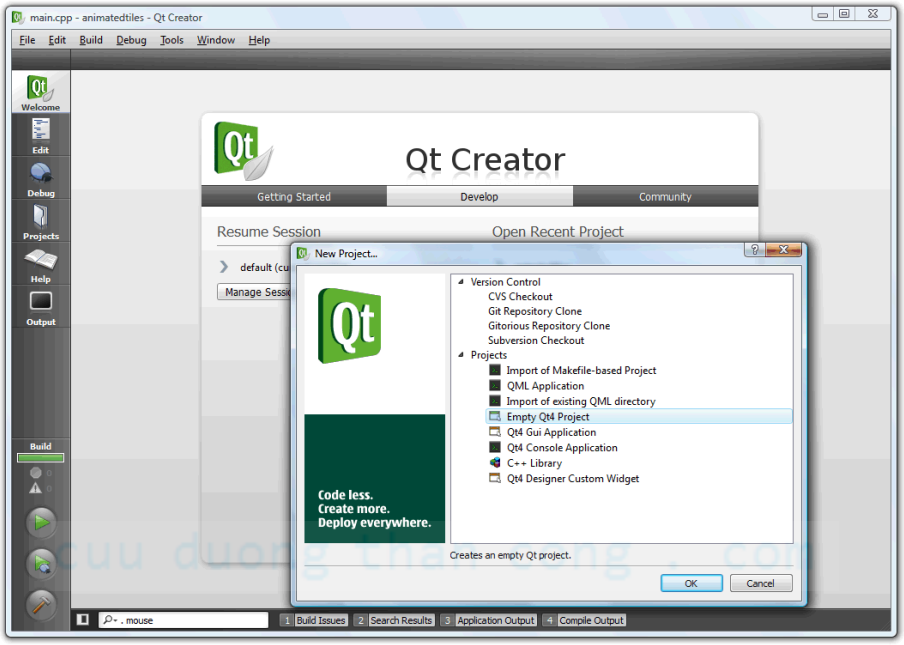
CuuDuongThanCong.com https://fb.com/tailieudientucntt

To resume execution, press the resume button on the debugging bar. Exiting the example program will end the debug session.

When you have run and debugged this example, you have verified that your development setup is in order and you are ready to continue.

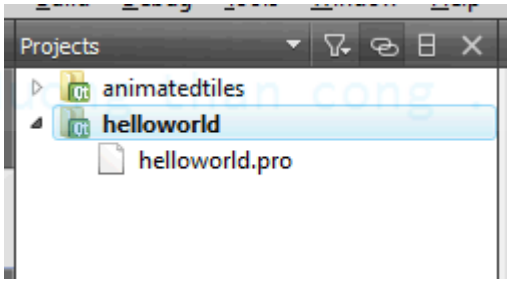
Create a Hello World project

To create a project of your own, go the the welcome screen using the top-left button (Ctrl+1), pick the page “Develop” and click “Create New Project...”.



From the dialog that pops up, create an “Empty Qt4 Project”, then specify a location for your project. Notice that Qt Creator will create a sub-directory with your project's name, so if you choose to name the project helloworld and create it in /home/user/code, your source files will end up in /home/user/code/helloworld.

If you have other projects open (from earlier exercises), make sure to either close them (right click on the project node and choose “Close Project ...” from the menu) or ensure that you are working with your project by right clicking on your project in the projects tree view and pick your project as the current *run configuration*. Your project should be marked as bold as shown below.



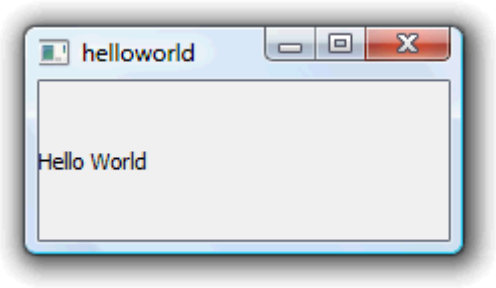
CuuDuongThanCong.com https://fb.com/tailieudientucntt

Now, right click on your project and choose “Add New...” from the menu. In the following dialogs, create a “C++ Source File” and name it main.cpp. In the file, enter the following code and save it.

```
#include <QApplication>
#include <QLabel>

int main(int argc, char **argv)
{
    QApplication a(argc, argv);
    QLabel l("Hello World");
    l.show();
    return a.exec();
}
```

Now, run the application (Ctrl+R) and try stretching and moving the resulting window about.



cuu duong than cong . com

cuu duong than cong . com

CuuDuongThanCong.com https://fb.com/tailieudientucntt

Lecture 2 – The Qt Object Model and Signal Slot Mechanism

Exploring Inheritance

Start by creating a new, empty Qt 4 project in Qt Creator. Add the following to your project:

- A C++ class, ValueObject, inheriting from QObject
- A C++ source file, main.cpp

In main.cpp, create a main function and include the ValueObject header file. In your main function, implement the following function body.

```
{
    ValueObject o;

    // Insert code here

    return 0;
}
```

Using the function QObject::inherits(const char *_className_), write qDebug statements testing if o inherits QObject and ValueObject.

> **Note!** If using Qt Creator on Windows, you might have to run your code in the debugger to see the qDebug printouts!

Try commenting out the Q_OBJECT macro from the valueobject.h header file. Discuss and explain the results. Remember to re-enable the Q_OBJECT macro after having tested this.

Now, try testing a QFile object. Does it inhert QIODevice, QDataStream, QObject or QTemporaryFile? Can you find this information in the documentation as well as through testing?

Properties

For this step, we keep the class that we created in the previous step.

Start by adding two member functions to the ValueObject class:

- setValue, setting an integer value of the object
- value, returning the set integer value of the object

These methods set and get an int value. You will have to add a private int member variable to keep that value. Do not forget to initialize the private value in the constructor.

CuuDuongThanCong.com https://fb.com/tailieudientucntt

Now, modify your main function to look like this.

```
int main(int argc, char **argv)
{
    ValueObject o;

    qDebug("Value: %d = %d", o.value(), o.property("value").toInt());
    o.setValue(42);
    qDebug("Value: %d = %d", o.value(), o.property("value").toInt());
    o.setProperty("value", 11);
    qDebug("Value: %d = %d", o.value(), o.property("value").toInt());

    return 0;
}
```

You will have to include the QVariant header file as well for the property calls to work.

Run the code, discuss and explain the results.

Now, add a Q_PROPERTY macro defining the value property with a READ and a WRITE function. Add the macro just after the Q_OBJECT macro in the class declaration.

Re-run the code, discuss and explain the different result.

Memory Management

Continuing from the previous step, we now add a qDebug statement in the constructor along with a destructor with a qDebug statement. Make sure that the destructor is virtual. The qDebug statements should inform you when an object is constructed and when it is destructed. In the destructor, add a reference to the QObject::objectName property as well.

Replace the property setting and debugging from the last step with the following line.

```
    o.setObjectName("root");
```

Running the code should now yield an output looking something like this.

```
ValueObject constructed.
ValueObject root destructed.
```

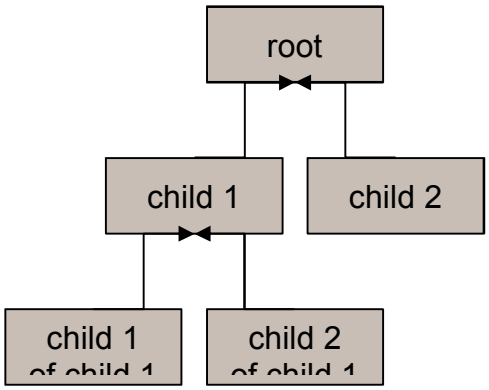
Now, add the following code just after the setting of the object name of of o.

```
    ValueObject *c1 = new ValueObject();
    c1->setObjectName("child 1");
    ValueObject *c2 = new ValueObject();
    c2->setObjectName("child 2");
    ValueObject *c1c1 = new ValueObject();
    c1c1->setObjectName("child 1 of child 1");
    ValueObject *c2c1 = new ValueObject();
    c2c1->setObjectName("child 2 of child 1");
    ValueObject *c = new ValueObject();
    c->setObjectName("child");
```

Run the code and evaluate the result. The code is leaking memory.

CuuDuongThanCong.com https://fb.com/tailieudientucntt

Now, provide a parent pointer to the constructors of c1, c2, c1c1 and c2c1 so that the ownership tree looks like the illustration below.



Finally, assign c2 as the parent of c using the QObject::setParent method.

Run the code and verify that all objects are properly destructed. Discuss and explain the order of destruction, as well as why having the root object on the stack instead of the heap simplifies the situation.

Signals and Slots

Again, building on the results from the last step, we will now add signals and slots to our ValueObject class.

Set functions are natural slots, and are also natural points for adding signals tracking the value of the property being set. Update the ValueObject class declaration with these changes.

- Move setValue to the public slots section
- Add a signal called valueChanged carrying an integer argument

Alter setValue so that it prints the new value and emits the valueChanged signal upon the value changing.

Now, replace the contents of your main function with the following code.

```
int main(int argc, char **argv)
{
    ValueObject o1;
    ValueObject o2;

    o1.setValue(1);
    o2.setValue(2);

    // Connect here

    qDebug("o1: %d, o2: %d", o1.value(), o2.value());

    o1.setValue(42);
    qDebug("o1: %d, o2: %d", o1.value(), o2.value());

    o2.setValue(11);
    qDebug("o1: %d, o2: %d", o1.value(), o2.value());

    return 0;
}
```

Running the code shows that o1 and o2 are completely independent.

CuuDuongThanCong.com https://fb.com/tailieudientucntt

Now, establish a connection from o1 to o2, making the value of o2 follow o1.

Run the code, discuss and explain the behavior.

Now establish another connection from o2 to o1, making the two ValueObject instances follow each other.

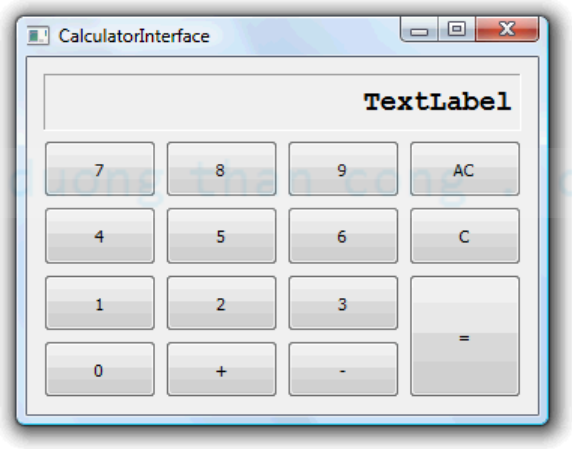
Run the code, discuss and explain the behavior.

> If the program hangs here, setting the same value over and over again, you have forgotten to add a check in your setValue. If the value isn't changed, the setValue slot should simply return.

Try to come up with situations in a user interface where the two connection relationships are useful, i.e. one object following another, versus two objects following each other.

The Signal Mapper

This step uses the *calculator* project as the base. Please open the project in Qt Creator and familiarize yourself with the components of the project.



The project is divided into three parts.

- main.cpp, containing a boilerplate main function
- The class CalculatorInterface, providing a user interface for a basic calculator
- The class Calculator, providing a trivial calculator engine implementation

In the file calculatorinterface.cpp, you will find the constructor shown below.

```
CalculatorInterface::CalculatorInterface(QWidget *parent) :
    QWidget(parent),
    ui(new Ui::CalculatorInterface),
    m_calculator(new Calculator(this))
{
    ui->setupUi(this);

    // Add code here

    m_calculator->allClear();
}
```

The user interface, made available through the ui variable, holds the following set of widgets:

- buttonZero
- buttonOne

CuuDuongThanCong.com https://fb.com/tailieudientucntt

- buttonTwo
- buttonThree
- buttonFour
- buttonFive
- buttonSix
- buttonSeven
- buttonEight
- buttonNine
- buttonClear
- buttonAllClear
- buttonAdd
- buttonSubtract
- buttonCalculate
- entryLabel

cuu duong than cong . com

cuu duong than cong . com

CuuDuongThanCong.com https://fb.com/tailieudientucntt

The Calculator class, object m_calculator, holds the following slots:

- numEntered(int) – the number provided has been entered (0-9)
- clear() - clear the current entry
- clearAll() - clear the calculator state completely
- additionMode() - switch to addition mode, calculates before switching mode
- subtractionMode() - switch to subtraction mode, calculates before switching mode
- calculate() - calculate the next result, clears the current entry

The class also provides a signal, displayChanged(QString), used to update any display showing the status of the calculator.

In the CalculatorInterface constructor, make all the connections you can see fit. Run the code and discuss the result.

Extending the Calculator

The Calculator class is far from perfect. The interested student can extend the calculator in various ways:

- Add more operations (e.g. multiplication and division)
- Add support for entering negative numbers
- Add memory operation (M+, M-, MC, MR)
- Add support for larger numbers (the current limit is the size of int)

cuu duong than cong . com

CuuDuongThanCong.com https://fb.com/tailieudientucntt

Solution Tips

Step 1 - Inheritance

Test if the object o inherits from QObject using the following line.

```
qDebug("ValueObject inherits QObject: %s",
      o.inherits("QObject")?"yes":"no");
```

Step 2 – Properties

The class declaration looks like this when the member functions and private variable have been added.

```
class ValueObject : public QObject
{
Q_OBJECT
public:
   explicit ValueObject(QObject *parent = 0);

   void setValue(int value);
   int value() const;

private:
   int m_value;
};
```

The property declaration to be added at the end of the exercise reads like this.

```
Q_PROPERTY(int value READ value WRITE setValue)
```

Step 3 – Memory Management

The constructors need to be called with the following parent pointers.

```
   ValueObject *c1 = new ValueObject(&o);
   ValueObject *c2 = new ValueObject(&o);
   ValueObject *c1c1 = new ValueObject(c1);
   ValueObject *c2c1 = new ValueObject(c1);
```

The parent of c is assigned using the following line.

```
   c->setParent(c2);
```

cuu duong than cong . com

CuuDuongThanCong.com https://fb.com/tailieudientucntt

Step 4 – Signals and Slots

The class declaration looks like this when the slot and signal have been added.

```
class ValueObject : public QObject
{
Q_OBJECT

Q_PROPERTY(int value READ value WRITE setValue)

public:
    explicit ValueObject(QObject *parent = 0);
    virtual ~ValueObject();

    int value() const;

public slots:
    void setValue(int value);

signals:
    void valueChanged(int);

private:
    int m_value;
};
```

The connections for interconnecting o1 and o2 look like this.

```
    QObject::connect(&o1, SIGNAL(valueChanged(int)),
                     &o2, SLOT(setValue(int)));
    QObject::connect(&o2, SIGNAL(valueChanged(int)),
                     &o1, SLOT(setValue(int)));
```

The implemented slot looks like this.

```
void ValueObject::setValue(int value)
{
    if(m_value == value)
        return;

    m_value = value;
    qDebug("Value set to %d", m_value);
    emit valueChanged(m_value);
}
```

cuu duong than cong . com

CuuDuongThanCong.com https://fb.com/tailieudientucntt

Hướng dẫn cài đặt QT Everywhere trên KIT FriendlyArm

1. Mục đích:

- Cài đặt Qt SDK trên máy tính Linux, cho phép phát triển ứng dụng chạy trên Desktop.

- Cài đặt thư viện Qt Everywhere (hay Qt Embedded), cho phép phát triển ứng dụng chạy trên KIT FriendlyArm (Embedd Linux).

- Hướng dẫn này thực hiện đối với phiên bản Qt 4.7.2

Giới thiệu

- Qt là một framework phát triển ứng dụng đa nền tảng, bao gồm:

+ a cross-platform class library (Thư viện các lớp hướng đối tượng)

+ integrated development tools (Các công cụ phát triển tích hợp)

+ a cross-platform IDE. (Môi trường phát triển ứng dụng)

cuu duong than cong . com

- Qt cho phép viết ứng dụng một lần và biên dịch chéo trên nhiều nền tảng hệ điều hành khác nhau mà không phải viết lại mã. Tuy nhiên, mã nguồn cần được

biên dịch trên nền tảng mà muốn ứng dụng được thực thi. Qt được phát triển bởi công ty Trolltech từ năm 1994, đến 2008 được sát nhập vào hãng Nokia và

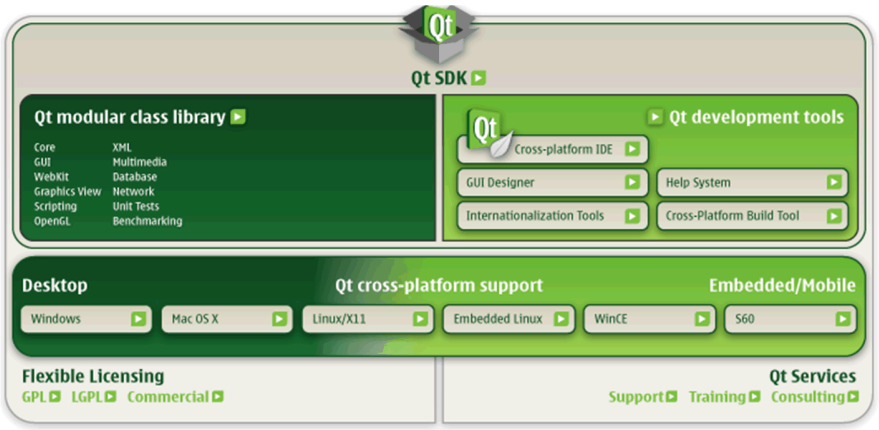
được gọi là Qt Software.

- Lập trình Qt theo chuẩn C++.

- Hình dưới chỉ ra các thành phần của bộ công cụ lập trình Qt SDK

cuu duong than cong . com

CuuDuongThanCong.com https://fb.com/tailieudientucntt



2. Cài đặt Qt:

- Bộ cài QT SDK cho Desktop

- Bộ cài QT Everywhere

Có thể phát triển ứng dụng Qt trên 3 nền tảng máy tính khác nhau: Linux, Windows, hoặc Mac. Trong hướng dẫn này chỉ tập trung vào cài đặt Qt trên máy tính

Linux X86.

2.1. Cài đặt Qt SDK trên máy tính X86 Linux (Ubuntu)

Bước 1. Chuẩn bị file cài đặt **qt-sdk-linux-x86-opensource-2010.05.1.bin**

Download từ trang chủ http://qt.nokia.com/downloads (chọn phiên bản thích hợp cho X86 Linux 32 bit), hoặc copy vào máy.

Bước 2. Di chuyển đến thư mục chứa file cài đặt, cấp quyền thực thi cho file nếu cần. Dùng lệnh:

```
$ chmod u+x qt-sdk-linux-x86-opensource-2010.05.1.bin
```

Bước 3. Thực thi file cài đặt từ dòng lệnh:

```
$ ./qt-sdk-linux-x86-opensource-2010.05.1.bin
```

Đợi quá trình cài đặt diễn ra thành công, mặc định thư mục cài đặt chứa tại $HOME/qtsdk-2010.01/qt/bin

Sau khi cài đặt xong Qt SDK, công cụ Qt Creator cho phép phát triển ứng dụng với lựa chọn mặc định biên dịch trên máy tính Linux. Để biên dịch chéo ứng dụng

CuuDuongThanCong.com https://fb.com/tailieudientucntt

thực thi trên KIT FriendlyArm (Embedded Linux) cần cài đặt Qt Everywhere

2.2. Cài đặt Qt Everywhere trên host Linux

Để ứng dụng viết bằng Qt chạy trên KIT FriendlyArm có thể sử dụng màn hình touchscreen của KIT, cần cài một thư viện hỗ trợ touchscreen gọi là tslib, trước khi

cấu hình cài đặt Qt Everywhere.

2.2.1. Cài đặt và build tslib

Trước tiên cần cài đặt một số công cụ hỗ trợ biên dịch tslib nếu máy tính chưa có sẵn.

```
sudo apt-get install autoconf

sudo apt-get install libtool
```

Download tslib:

```
$ cd ~

$ mkdir tslib-arm

$ cd tslib-arm

$ apt-get source tslib
```

Configure tslib:

```
$ cd tslib-1.0

$ ./autogen.sh

$ ./configure --prefix=$HOME/tslib_arm/ --host=arm-none-linux-gnueabi
```

Trong file config.h, sửa dòng sau (approx. line 185)

```
#define malloc rpl_malloc --> /* #define malloc rpl_malloc */
```

Sau đó, biên dịch (cross-compile) tslib:

```
$ make

$ sudo make install
```

Đợi cho quá trình biên dịch và cài đặt kết thúc thành công. (Nếu có lỗi cần kiểm tra lỗi và thực

CuuDuongThanCong.com https://fb.com/tailieudientucntt

hiện lại). Thư viện tslib đã sẵn sàng sử dụng (sẽ được cấu hình sử

dụng khi cài đặt Qt Everywhere).

2.2.2. Cài đặt Qt Everywhere trên host

Bước 1. Chuẩn bị file nén gói cài đặt **qt-everywhere-opensource-src-4.7.2.tar.gz**

Có thể download từ trang chủ http://qt.nokia.com/downloads (chọn phiên bản thích hợp), hoặc copy trên máy tính. (Để tại thư mục người dùng $HOME)

Bước 2. Giải nén file cài đặt

Mở cửa sổ lệnh, di chuyển đến thư mục chứa file cài đặt trên và tiến hành giải nén.

Dùng lệnh:

```
gunzip qt-everywhere-opensource-src-4.7.2.tar.gz

tar xf qt-everywhere-opensource-src-4.7.2.tar
```

Kết quả giải nén ra một thư mục cùng tên file tar.

cuu duong than cong . com

Bước 3. Cấu hình và biên dịch gói cài đặt

Di chuyển vào thư mục đã giải nén của gói cài đặt:

```
cd qt-everywhere-opensource-src-4.7.2
```

Thêm biến môi trường đến đường dẫn trình biên dịch Qt (Mở file ~/.bashrc, thêm biến môi trường $HOME/qt-everywhere-opensource-src-4.7.2)

Sửa file cấu hình biên dịch qmake.conf chứa tại $HOME/qt-everywhere-opensource-src-4.7.2/mkspecs/qws/linux-arm-g++/qmake.conf như sau.

```
#

# qmake configuration for building with arm-linux-g++

#

include(../../common/g++.conf)

include(../../common/linux.conf)

include(../../common/qws.conf)
```

cuu duong than cong . com

CuuDuongThanCong.com https://fb.com/tailieudientucntt

```
# modifications to g++.conf
QMAKE_CC       = arm-none-linux-gnueabi-gcc
QMAKE_CXX          = arm-none-linux-gnueabi-g++
QMAKE_LINK         = arm-none-linux-gnueabi-g++
QMAKE_LINK_SHLIB   = arm-none-linux-gnueabi-g++
# modifications to linux.conf
QMAKE_AR           = arm-none-linux-gnueabi-ar cqs
QMAKE_OBJCOPY      = arm-none-linux-gnueabi-objcopy
QMAKE_STRIP        = arm-none-linux-gnueabi-strip
QMAKE_INCDIR    += /opt/tslib/include
QMAKE_LIBDIR    += /opt/tslib/lib
Load(qt_config)
```

(***Chú ý:*** Biến môi trường có thể được thêm không chính xác hoặc không thành công, trong trường hợp đó trong file qmake.conf trên nên đặt đường dẫn tuyệt đối đến các trình biên dịch)

Sau đó, tiến hành cấu hình thư viện Qt Everywhere 4.7.2 trước khi cài đặt

```
$ cd
$ cd qt-everywhere-opensource-src-4.7.2
./configure --prefix=/opt/qte -embedded arm -xplatform qws/linux-arm-g++ -qt-mouse-tslib -little-endian -no-qt3support -fast -no-largefile -qt-sql-sqlite -nomake tools -nomake demos -nomake examples -no-webkit -no-multimedia -no-javascript-jit
```

(Quá trình cấu hình diễn ra một thời gian), chờ đến khi thông báo thành công. Tiến hành dịch thư viện:

```
$ make
```

(Quá trình make diễn ra ~hour phụ thuộc vào các option lựa chọn khi configure ở trên). Trong

CuuDuongThanCong.com https://fb.com/tailieudientucntt

trường hợp thất bại (xuất hiện thông báo lỗi), cần thực hiện lại:

make confclean, cấu hình lại với option thích hợp và make lại.

Đợi quá trình make thành công, gõ lệnh cài đặt:

```
$ sudo make install
```

Đợi quá trình cài đặt thành công, kiểm tra kết quả cài đặt tại **/opt/qte** như đã cấu hình ở trên.

2.2.3. Cài đặt Qt Everywhere trên target (KIT)

Thư viện Qt Everywhere cài trên host Linux cho phép biên dịch ứng dụng Qt để chạy trên KIT FriendlyArm. Tuy nhiên, cần copy một số file thư viện cơ bản, và font xuống KIT để ứng dụng có thể thực thi.

Thư viện Qt Everywhere (Libraries): Copy tối thiểu 3 file thư viện sau (Copy quá nhiều sẽ tốn bộ nhớ)

cuu duong than cong . com

- libQtCore.so.4
- libQtGui.so.4
- libQtNetwork.so.4

Trên KIT đặt vào thư mục **/opt/qte/lib** (nếu chưa có cần tạo thư mục này, giống như thư mục trên host)

Fonts: Copy thư mục fonts xuống KIT chứa ở **/opt/qte/lib/fonts**

Thư viện tslib: Copy toàn bộ thư mục **/opt/tslib** trên host xuống KIT tại thư mục /opt/tslib

cuu duong than cong . com

2.2.4. Thiết lập để chạy ứng dụng Qt trên KIT.

Tắt Qtopia

- Mặc định khi khởi động KIT với hệ điều hành Linux nhúng đã cài đặt, thư viện Qtopia 2.0

CuuDuongThanCong.com https://fb.com/tailieudientucntt

được khởi động. (Các ứng dụng viết để chạy trên thư viện này cần được build lại cùng Qtopia 2.0). Để chạy ứng dụng Qt trực tiếp, cần tắt Qtopia để tránh xung đột. Thực hiện như sau:

- Trên KIT, dùng **vi** mở file cấu hình **/etc/init.d/rcS,** disable dòng Qtopia.

Chỉnh sửa cấu hình để sử dụng touchscreen trên KIT

- Dùng vi mở file cấu hình **/opt/tslib/etc/ts.conf,** bỏ chú thích dòng lệnh module_raw input

- Thêm biến môi trường để sử dụng thư viện tslib, dùng vi sửa file **/etc/profile,** thêm nội dung như sau (dùng lib copy lên KIT):

```
export TSLIB_TSEVENTTYPE=INPUT
export TSLIB_ROOT=/opt/tslib
export TSLIB_TSDEVICE=/dev/input/event0
export LD_LIBRARY_PATH=$TSLIB_ROOT/lib:$LD_LIBRARY_PATH
export TSLIB_FBDEVICE=/dev/fb0
export TSLIB_PLUGINDIR=$TSLIB_ROOT/lib/ts
export TSLIB_CONSOLEDEVICE=none
export TSLIB_CONFFILE=$TSLIB_ROOT/etc/ts.conf
export POINTERCAL_FILE=/etc/pointercal
export TSLIB_CALIBFILE=/etc/pointercal
export QWS_MOUSE_PROTO=TSLIB:/dev/input/event0
```

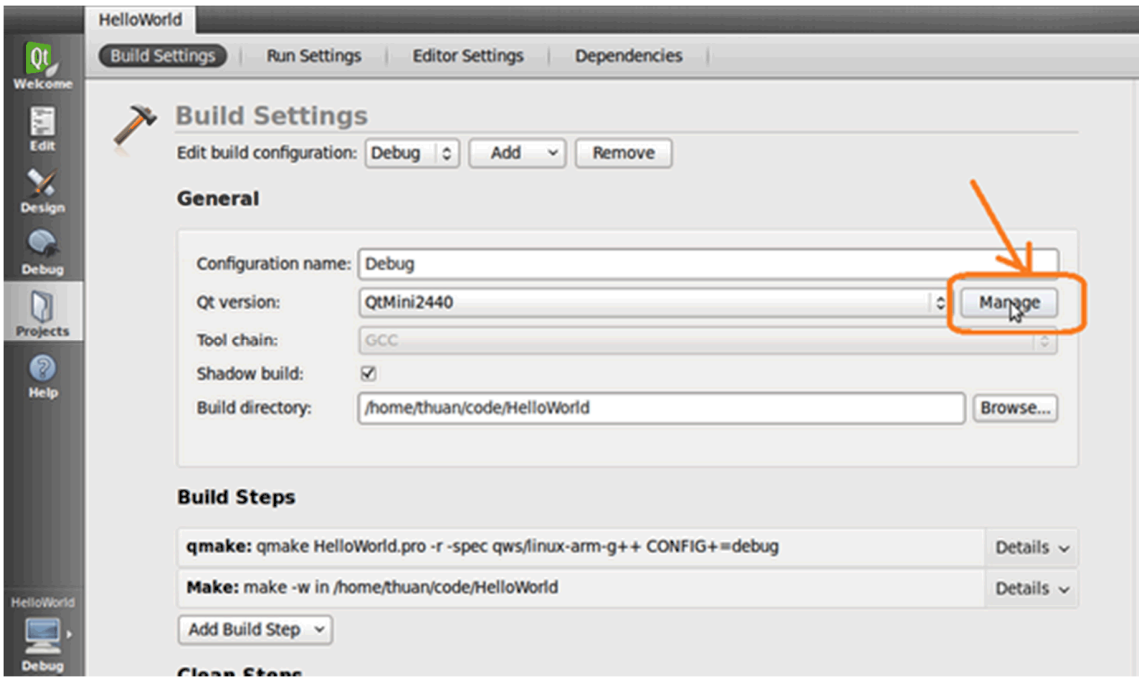
Quá trình cài đặt đến đây thành công, ứng dụng viết bằng Qt có thể được biên dịch và nạp lên KIT để thực thi.

3. Cấu hình Qt Creator để biên dịch với nền tảng Qt Embedded (Qt Everywhere)

Bước 1. Chọn tab Project, mở giao diện Build Settings tương ứng Project đang cấu hình.

CuuDuongThanCong.com https://fb.com/tailieudientucntt

Trên cửa sổ này, mở mục Manage



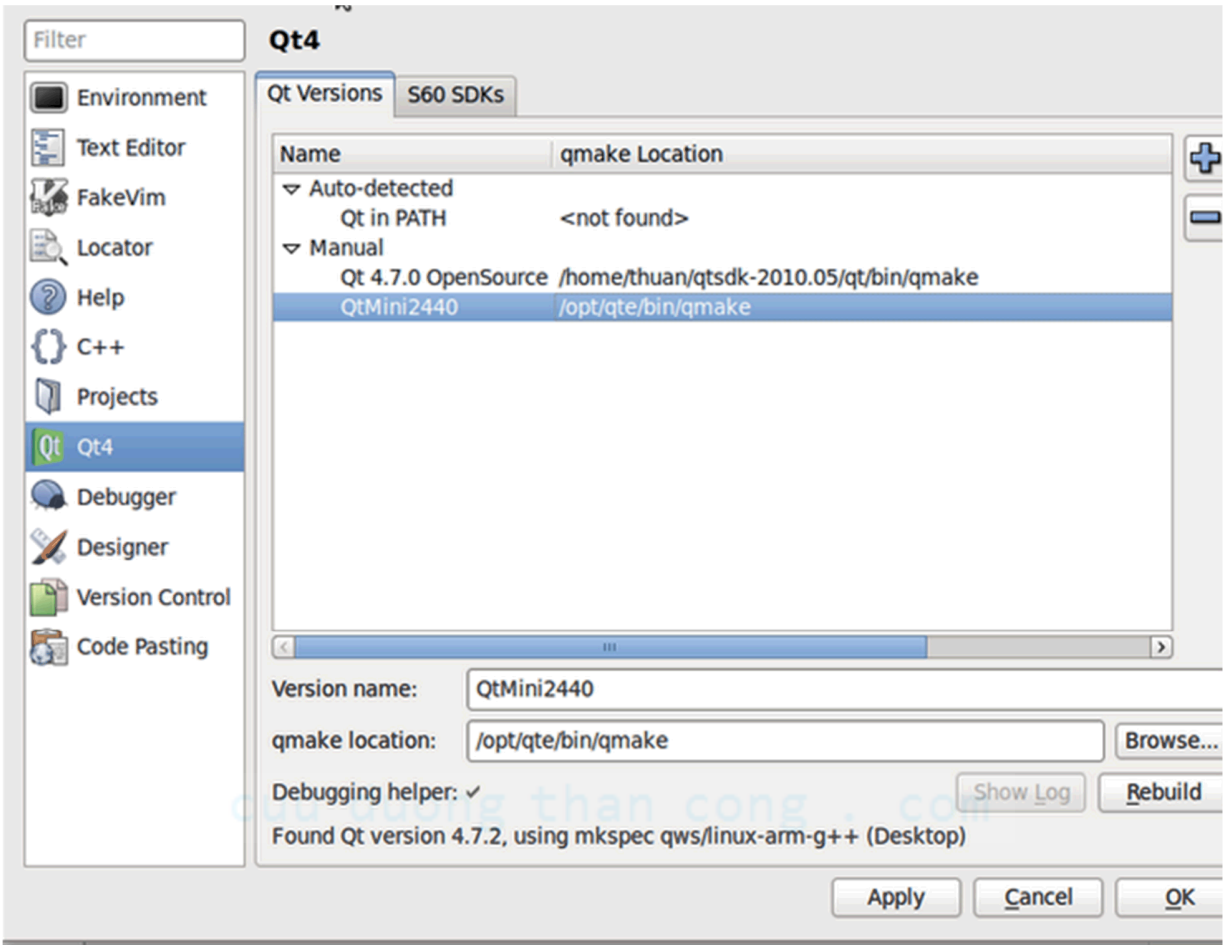
cuu duong than cong . com

Bước 2. Click nút Add (+) để thêm mới một nền tảng biên dịch chéo.

Tạo 1 cấu hình biên dịch mới, đặt tên Mini2440, trỏ tới Qmake đã cài đặt ở phần trước (nằm trong /opt/qte/bin/qmake)

cuu duong than cong . com

CuuDuongThanCong.com https://fb.com/tailieudientucntt



Bước 3. Biên dịch ứng dụng với cấu hình biên dịch đã thiết lập ở trên.

cuu duong than cong . com

CuuDuongThanCong.com https://fb.com/tailieudientucntt

Bước 1. Khởi động IDE Qt Creator, tạo một Project

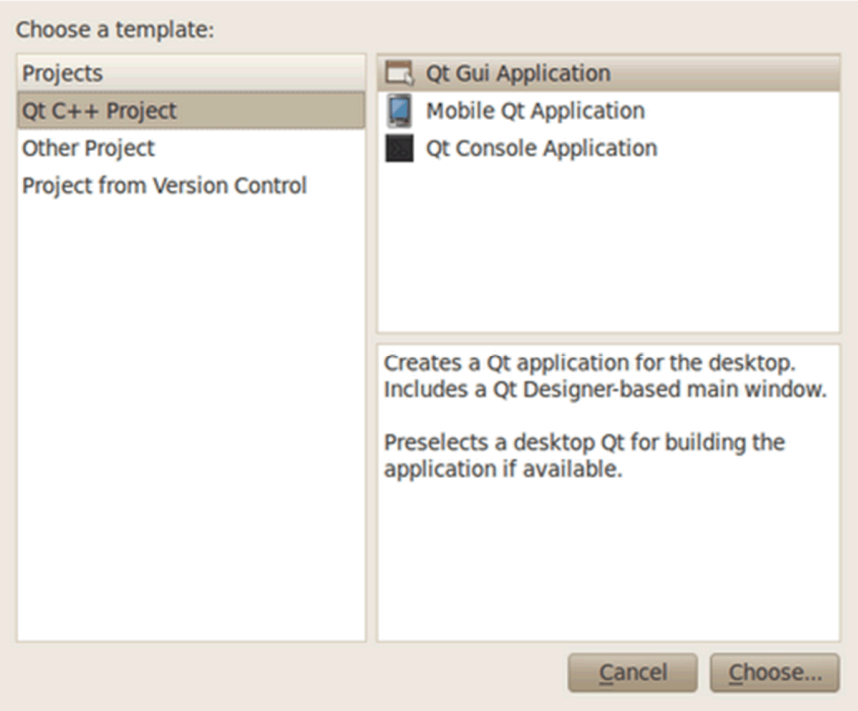
Giao diện Start Page (như hình dưới) cho phép:

- Mở Project mẫu (Choose an example)
- Xem tài liệu hướng dẫn lập trình (Tutorials)
- Mở project đã có (Open Project)
- Tạo Project mới (Create Project).

Chọn Create Project để tạo một dự án mới:

cuu duong than cong . com

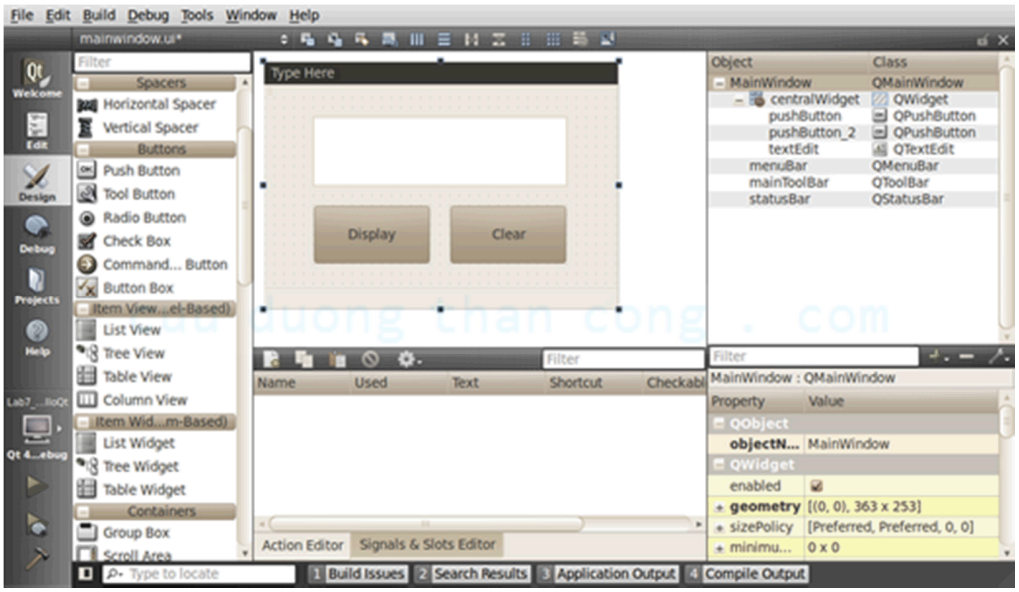
CuuDuongThanCong.com https://fb.com/tailieudientucntt



Chọn Qt C++ Project/Qt GUI Application, thực hiện tiếp các bước theo hướng dẫn Wizard của Qt Creator.

Lưu ý chọn cấu hình nền tảng biên dịch đã thiết lập để biên dịch ứng dụng chạy trên KIT FriendlyArm (hoặc biên dịch chạy trên desktop).

Bước 2. Tại Form chính thiết kế một giao diện cơ bản như sau:



CuuDuongThanCong.com https://fb.com/tailieudientucntt

- Kéo một TextEdit và 2 Pushbutton, một nút tên là Display và một nút tên là Clear.

- Chuột phải vào nút display, chọn Go to slot … ->> chọn Clicked() … ->> OK

Thêm dòng lệnh sau cho sự kiện click của pushbutton display:

```
ui->textEdit->setText(QString("Hello World!"));
```

- Chuột phải vào nút Clear, chọn Go to slot … ->> chọn Clicked() … ->> OK.

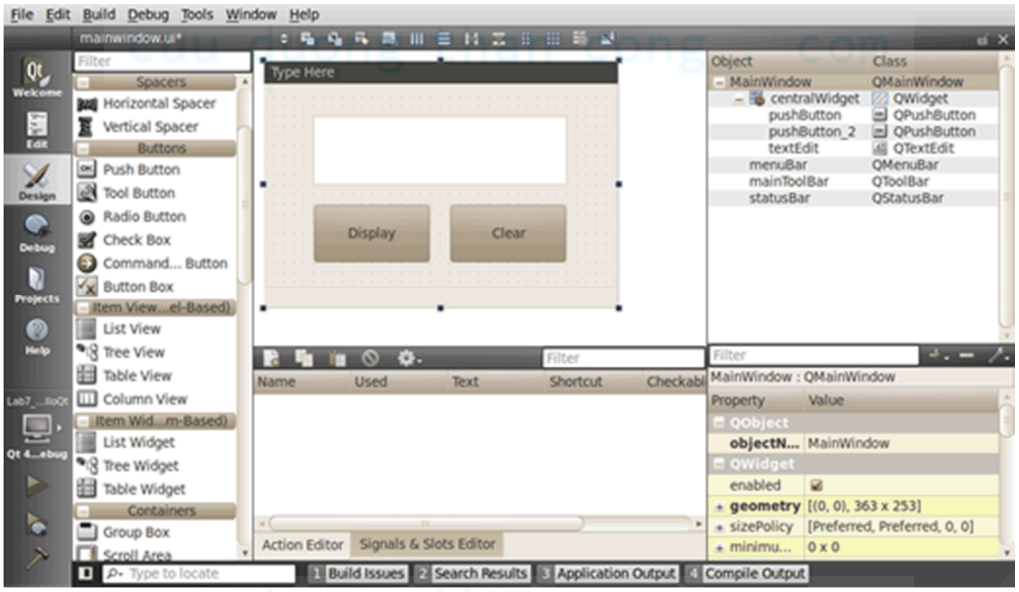
Thêm dòng lệnh sau cho sự kiện click của pushbutton Clear:

```
ui->textEdit->setText(QString(" "));
```

Bước 3. Build Project

Chú ý trong cửa sổ Build Settings, chọn cấu hình biên dịch qmake cho nền tảng Qt FriendlyArm (phiên bản tương ứng với Qt Everywhere đã cài đặt)

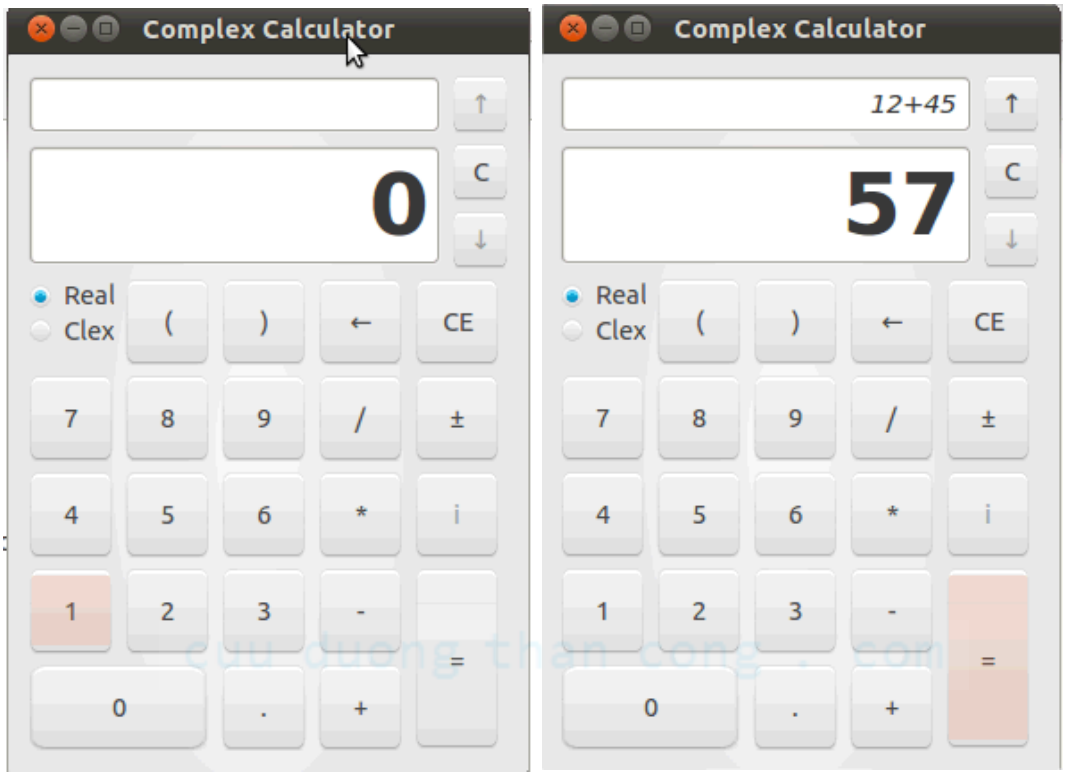
Chọn Build/Build All. Ứng dụng được build thành công, kết quả được file thực thi (trong thư mục Project) sẽ chuyển lên KIT để chạy. Ví dụ này là file HelloQt).



CuuDuongThanCong.com https://fb.com/tailieudientucntt

Bài tập: LẬP TRÌNH CHƯƠNG TRÌNH CALCULATOR SỬ DỤNG QT

1. Giao diện của hệ thống



2. Hoạt động

- Khi ta thực hiện các phép tính cộng trừ nhân chia các số thực thì ta check vào option "Real"
- Khi ta thực hiện các phép tính cộng trừ nhân chia cả số thực lẫn số phức thì ta check vào option "Clex"
- Sau khi thực hiện xong 1 phép tính thì kết quả tại khung nhập và thể hiện phép tính ở khung trên, ta có thể chỉnh sửa các phép tính ở khung ở phép trên và bấm dấu bằng cũng cho ra kết quả.
- Chương trình được viết trên Qt creator

3. Code chương trình calculator

+ calculatorwin.cpp

```
#include"calculatorwin.h"
#include<QDebug>
```


CuuDuongThanCong.com https://fb.com/tailieudientucntt

```
CalculatorWin::CalculatorWin()
    :mathString( "" ), item( "0" ), isRealMode( true )
{
    //initialize mian window
    setupUi( this );
    engine = new Engine();
    timer = new QTimer();
    mainLineEdit->setText( "0" );
    mainLineEdit->setMaxLength( 9 );
    mainLineEdit->setReadOnly( true );
    expandLineEdit->setReadOnly( true );
    realRadioButton->setChecked( true );
    complexButton->setEnabled( false );
    timer->start( 100 );

    //connect signals with slots
    connect( zeroButton, SIGNAL( clicked() ), this, SLOT( zeroButtonClicked() ) );
    connect( oneButton, SIGNAL( clicked() ), this, SLOT( oneButtonClicked() ) );
    connect( twoButton, SIGNAL( clicked() ), this, SLOT( twoButtonClicked() ) );
    connect( threeButton, SIGNAL( clicked() ), this, SLOT( threeButtonClicked() ) );
    connect( fourButton, SIGNAL( clicked() ), this, SLOT( fourButtonClicked() ) );
    connect( fiveButton, SIGNAL( clicked() ), this, SLOT( fiveButtonClicked() ) );
    connect( sixButton, SIGNAL( clicked() ), this, SLOT( sixButtonClicked() ) );
    connect( sevenButton, SIGNAL( clicked() ), this, SLOT( sevenButtonClicked() ) );
    connect( eightButton, SIGNAL( clicked() ), this, SLOT( eightButtonClicked() ) );
    connect( nineButton, SIGNAL( clicked() ), this, SLOT( nineButtonClicked() ) );
    connect( dotButton, SIGNAL( clicked() ), this, SLOT( dotButtonClicked() ) );
    connect( clearButton, SIGNAL( clicked() ), this, SLOT( clearButtonClicked() ) );
    connect( realRadioButton, SIGNAL( clicked() ), this, SLOT( setRealMode() ) );
    connect( complexRadioButton, SIGNAL( clicked() ), this, SLOT( setRealMode() ) );
    connect( complexButton, SIGNAL( clicked() ), this, SLOT( complexButtonClicked() ) );
    connect( backButton, SIGNAL( clicked() ), this, SLOT( backButtonClicked() ) );
    connect( addButton, SIGNAL( clicked() ), this, SLOT( addButtonClicked() ) );
    connect( subButton, SIGNAL( clicked() ), this, SLOT( subButtonClicked() ) );
    connect( signButton, SIGNAL( clicked() ), this, SLOT( signButtonClicked() ) );
    connect( mulButton, SIGNAL( clicked() ), this, SLOT( mulButtonClicked() ) );
    connect( divButton, SIGNAL( clicked() ), this, SLOT( divButtonClicked() ) );
    connect( frontBracketButton, SIGNAL( clicked() ), this, SLOT(
frontBracketButtonClicked() ) );
    connect( backBracketButton, SIGNAL( clicked() ), this, SLOT(
backBracketButtonClicked() ) );
    connect( equalButton, SIGNAL( clicked() ), this, SLOT( equalButtonClicked() ) );
    connect( upButton, SIGNAL( clicked() ), this, SLOT( upButtonClicked() ) );
    connect( downButton, SIGNAL( clicked() ), this, SLOT( downButtonClicked() ) );
    connect( clsHistoryButton, SIGNAL( clicked() ), this, SLOT( clsHistoryButtonClicked() ) );
```


CuuDuongThanCong.com https://fb.com/tailieudientucntt

```
    connect( timer, SIGNAL( timeout() ), this, SLOT( setUpOrDownButtonEnbled() ) ); //set a
timer to check the stack of history
}

//slots to response button clicked event
void CalculatorWin::zeroButtonClicked()
{
    if( item == "0" )
        mainLineEdit->setText( item );
    else if( ( item != "0" ) && ( item.at( item.length() - 1 ) != 'i' ) )
    {
        item = item + "0";
        mainLineEdit->setText( item );
    }
}
void CalculatorWin::oneButtonClicked()
{
    dealWithNonZero( "1" );
}
void CalculatorWin::twoButtonClicked()
{
    dealWithNonZero( "2" );
}
void CalculatorWin::threeButtonClicked()
{
    dealWithNonZero( "3" );
}
void CalculatorWin::fourButtonClicked()
{
    dealWithNonZero( "4" );
}
void CalculatorWin::fiveButtonClicked()
{
    dealWithNonZero( "5" );
}
void CalculatorWin::sixButtonClicked()
{
    dealWithNonZero( "6" );
}

void CalculatorWin::sevenButtonClicked()
{
    dealWithNonZero( "7" );
}

void CalculatorWin::eightButtonClicked()
```

cuu duong than cong . com

CuuDuongThanCong.com https://fb.com/tailieudientucntt

```
{
    dealWithNonZero( "8" );
}

void CalculatorWin::nineButtonClicked()
{
    dealWithNonZero( "9" );
}

void CalculatorWin::dotButtonClicked()
{
    if( isRealMode )
    {
        if( ( !item.contains(".") ) && ( item.at( item.length() - 1 ) != 'i' ) && ( item.at(
item.length() - 1 ) != '+' ) && ( item.at( item.length() - 1 ) != '-' ) )
        {
            item = item + ".";
            mainLineEdit->setText( item );
        }
    }
    else
    {
        if( ( item.count(".") < 2 ) && ( item.at( item.length() - 1 ) != 'i' ) && ( item.at(
item.length() - 1 ) != '+' ) && ( item.at( item.length() - 1 ) != '-' ) && ( item.at( item.length()
- 1 ) != '.' ) )
        {
            item = item + ".";
            mainLineEdit->setText( item );
        }
    }
}

void CalculatorWin::complexButtonClicked()
{
    if( !item.contains("i") )
    {
        item = item + "i";
        mainLineEdit->setText( item );
    }
}

void CalculatorWin::dealWithNonZero( QString number )    //function to deal with non-zero
number input
{
    if( item == "0" )
    {
```


CuuDuongThanCong.com https://fb.com/tailieudientucntt

```
        item = number;
        mainLineEdit->setText( item );
    }
    else if( ( item != "0" ) && ( item.at( item.length() - 1 ) != 'i' ) )
    {
        item = item + number;
        mainLineEdit->setText( item );
    }
}

void CalculatorWin::clearButtonClicked()    //slot to clean current expression
{
    mainLineEdit->setText( "0" );
    expandLineEdit->setText( "" );
    item = "0";
    mathString = "";
}

void CalculatorWin::setRealMode()   //slot to switch real mode with imag mode
{
    if( realRadioButton->isChecked() )
    {
        complexButton->setEnabled( false );
        isRealMode = true;
        item = "0";
        mainLineEdit->setText( "0" );
    }
    else
    {
        complexButton->setEnabled( true );
        isRealMode = false;
        item = "0";
        mainLineEdit->setText( "0" );
    }
}

void CalculatorWin::backButtonClicked()
{
    mainLineEdit->backspace();
    item = mainLineEdit->text();
    if( item.isEmpty() )
    {
        item = "0";
        mainLineEdit->setText( item );
    }
}
```


CuuDuongThanCong.com https://fb.com/tailieudientucntt

```
void CalculatorWin::addButtonClicked()
{
   item = mainLineEdit->text();
   if( ( !mathString.isEmpty() ) && ( mathString.at( mathString.length() - 1 ) == ')' ) )
   {
      mathString = mathString + "+";
      item = "0";
      expandLineEdit->setText( mathString );
   }
   else
   {
      if( isRealMode )
      {
         mathString = mathString + item + "+";
         item = "0";
         expandLineEdit->setText( mathString );
      }
      else
      {
         if( !item.contains( "+" ) )
         {
            if( !item.contains( "-" ) )
            {
               item = item + "+";
               mainLineEdit->setText( item );
            }
            else if( ( ( item.count( "-" ) ) == 1 ) && ( item.at(0) == '-' ) )
            {
               item = item + "+";
               mainLineEdit->setText( item );
            }
            else
            {
               mathString = mathString + "(" + item + ")" + "+";
               item = "0";
               expandLineEdit->setText( mathString );
            }
         }
         else
         {
            mathString = mathString + "(" + item + ")" + "+";
            item = "0";
            expandLineEdit->setText( mathString );
         }
      }
```

cuu duong than cong . com

CuuDuongThanCong.com https://fb.com/tailieudientucntt

```
    }
}

void CalculatorWin::subButtonClicked()
{
    item = mainLineEdit->text();
    if( ( !mathString.isEmpty() ) && ( mathString.at( mathString.length() - 1 ) == ')' ) )
    {
        mathString = mathString + "-";
        item = "0";
        expandLineEdit->setText( mathString );
    }
    else
    {
        if( isRealMode )
        {
            mathString = mathString + item + "-";
            item = "0";
            expandLineEdit->setText( mathString );
        }
        else
        {
            if( ( ( !item.contains( "-" ) ) || ( ( item.at(0) == '-' ) && ( item.count( "-" ) ) == 1 ) )
&& ( !item.contains( "+" ) ) )
            {
                item = item + "-";
                mainLineEdit->setText( item );
            }
            else
            {
                mathString = mathString + "(" + item + ")" + "-";
                item = "0";
                expandLineEdit->setText( mathString );
            }
        }
    }
}

void CalculatorWin::signButtonClicked()
{
    item = mainLineEdit->text();
    if( item != "0" )
    {
        if( item.at(0) != '-' ) //insert a '+' before the begin of string
            item.insert( 0, "+" );
        //exchange '+' with '-'
```


CuuDuongThanCong.com https://fb.com/tailieudientucntt

```
        item.replace( "+", "a" );
        item.replace( "-", "s" );
        item.replace( "a", "-" );
        item.replace( "s", "+" );
        if( item.at(0) == '+' ) //delete '+' before the begin of string
            item.remove( 0, 1 );
        mainLineEdit->setText( item );
    }
}

void CalculatorWin::mulButtonClicked()
{
    item = mainLineEdit->text();
    if( ( !mathString.isEmpty() ) && ( mathString.at( mathString.length() - 1 ) == ')' ) )
    {
        mathString = mathString + "*";
        item = "0";
        expandLineEdit->setText( mathString );
    }
    else
    {
        if( !isRealMode )
        {
            if( item.at( item.length() - 1 ) == '+' || item.at( item.length() - 1 ) == '-' )
                item.remove( item.length() - 1, 1 );
            mathString = mathString + "(" + item + ")" + "*";
            item = "0";
            expandLineEdit->setText( mathString );
        }
        else
        {
            mathString = mathString + item + "*";
            item = "0";
            expandLineEdit->setText( mathString );
        }
    }
}

void CalculatorWin::divButtonClicked()
{
    item = mainLineEdit->text();
    if( ( !mathString.isEmpty() ) && ( mathString.at( mathString.length() - 1 ) == ')' ) )
    {
        mathString = mathString + "/";
        item = "0";
        expandLineEdit->setText( mathString );
```


CuuDuongThanCong.com https://fb.com/tailieudientucntt

```
    }
    else
    {
        if( !isRealMode )
        {
            if( item.at( item.length() - 1 ) == '+' || item.at( item.length() - 1 ) == '-' )
                item.remove( item.length() - 1, 1 );
            mathString = mathString + "(" + item + ")" + "/";
            item = "0";
            expandLineEdit->setText( mathString );
        }
        else
        {
            mathString = mathString + item + "/";
            item = "0";
            expandLineEdit->setText( mathString );
        }
    }
}

void CalculatorWin::frontBracketButtonClicked()
{
    if( !mathString.isEmpty() )
    {
        if( mathString.at( mathString.length() - 1 ) == '+' || mathString.at( mathString.length() -
1 ) == '-' || mathString.at( mathString.length() - 1 ) == '*' || mathString.at(
mathString.length() - 1 ) == '/' )
        {
            mathString = mathString + "(";
            expandLineEdit->setText( mathString );
        }
    }
    else
    {
        mathString = mathString + "(";
        expandLineEdit->setText( mathString );
    }
}

void CalculatorWin::backBracketButtonClicked()
{
    item = mainLineEdit->text();
    if( mathString.count( "(" ) - mathString.count( ")" ) ) //check the number of bracket
    {
        if( mathString.at( mathString.length() - 1 ) != ')' )
        {
```


CuuDuongThanCong.com https://fb.com/tailieudientucntt

```
            if( !isRealMode )
            {
                mathString = mathString + "(" + item + ")" + ")";
                item = "0";
                expandLineEdit->setText( mathString );
            }
            else
            {
                mathString = mathString + item + ")";
                item = "0";
                expandLineEdit->setText( mathString );
            }
        }
        else
        {
            mathString = mathString + ")";
            expandLineEdit->setText( mathString );
        }
    }
}

void CalculatorWin::equalButtonClicked()
{
    item = mainLineEdit->text();
    if( mathString == "" )
    {
        mathString = item;
        expandLineEdit->setText( mathString );
        *engine = mathString;   //transfer the expression to engine
        result = engine->calculate();   //calculate the expression and return the result
    }
    else if( ( mathString.at( mathString.length() - 1 ) == ')' ) || ( ( mathString.at(
mathString.length() - 1 ) >= '0' ) && ( mathString.at( mathString.length() - 1 ) <= '9' ) ) )
    {
        correctError(); //correct the errors in expression
        *engine = mathString;   //transfer the expression to engine
        result = engine->calculate();   //calculate the expression and return the result
    }
    else
    {
        if( !isRealMode )
        {
            mathString = mathString + "(" + item + ")";
            item = "0";
            expandLineEdit->setText( mathString );
            correctError(); //correct the errors in expression
```


CuuDuongThanCong.com https://fb.com/tailieudientucntt

```
            *engine = mathString;   //transfer the expression to engine
            result = engine->calculate();   //calculate the expression and return the result
        }
        else
        {
            mathString = mathString + item;
            item = "0";
            expandLineEdit->setText( mathString );
            correctError(); //correct the errors in expression
            *engine = mathString;   //transfer the expression to engine
            result = engine->calculate();   //calculate the expression and return the result
        }
    }
    if( !engine->isError() )    //if no errors in result, display result
    {
        displayResult();
        mathStack1.push( mathString );  //save current expression in stack
        mathString = "";    //clear current expression
    }
    else
    {
        mainLineEdit->setText( engine->getMassage() ); //display error massage
        mathString = "";    //clear current expression
    }
}

void CalculatorWin::displayResult() //function to display the result
{
    QString real;
    QString imag;
    QString resultStr;

    if( result.isZero() )
    {
        resultStr = "0";
    }
    if( result.getReal() != 0.0 )
    {
        real = real.number( result.getReal(), 'g', 3 );
        resultStr = resultStr + real;
    }
    if( result.getImag() != 0.0 )
    {
        imag = imag.number( result.getImag(), 'g', 3 );
        if( imag.at( 0 ) == '-' )
            resultStr = resultStr + imag + "i";
```


CuuDuongThanCong.com https://fb.com/tailieudientucntt

```
        else
            resultStr = resultStr + "+" + imag + "i";
    }
    mainLineEdit->setText( resultStr );
}

void CalculatorWin::correctError()  //function to correct the errors in input string
{
    mathString.replace( "+-", "-" );
    mathString.replace( "+i", "+1i" );
    mathString.replace( "-i", "-1i" );
    mathString.replace( "(i", "(1i" );
}

void CalculatorWin::upButtonClicked()   //slot to load the last expression
{
    if( !mathStack1.isEmpty() )
    {
        mainLineEdit->setText( "0" );
        item = "0";
        mathString = mathStack1.top();
        mathStack1.pop();
        mathStack2.push( mathString );
        expandLineEdit->setText( mathString );
    }
}

void CalculatorWin::downButtonClicked() //slot to load the next expression
{
    if( !mathStack2.isEmpty() )
    {
        mainLineEdit->setText( "0" );
        item = "0";
        mathString = mathStack2.top();
        mathStack2.pop();
        mathStack1.push( mathString );
        expandLineEdit->setText( mathString );
    }
}

void CalculatorWin::clsHistoryButtonClicked()    //slot to clear current expression and the
history of expression
{
    mainLineEdit->setText( "0" );
    expandLineEdit->setText( "" );
    item = "0";
```


CuuDuongThanCong.com https://fb.com/tailieudientucntt

```
    mathString = "";
    while( !mathStack1.isEmpty() )
        mathStack1.pop();
    while( !mathStack2.isEmpty() )
        mathStack2.pop();
}

void CalculatorWin::setUpOrDownButtonEnbled()   //slot to set up or down button enbled
{
    if( mathStack1.isEmpty() )
    {
        upButton->setEnabled( false );
    }
    else
    {
        upButton->setEnabled( true );
    }
    if( mathStack2.isEmpty() )
    {
        downButton->setEnabled( false );
    }
    else
    {
        downButton->setEnabled( true );
    }
}

void CalculatorWin::keyPressEvent( QKeyEvent *event )   //function to response keyboard
event
{
    switch( event->key() )
    {
    case Qt::Key_0:
        zeroButton->click();
        break;
    case Qt::Key_1:
        oneButton->click();
        break;
    case Qt::Key_2:
        twoButton->click();
        break;
    case Qt::Key_3:
        threeButton->click();
        break;
    case Qt::Key_4:
        fourButton->click();
```

cuu duong than cong . com

CuuDuongThanCong.com https://fb.com/tailieudientucntt

```
    break;
case Qt::Key_5:
    fiveButton->click();
    break;
case Qt::Key_6:
    sixButton->click();
    break;
case Qt::Key_7:
    sevenButton->click();
    break;
case Qt::Key_8:
    eightButton->click();
    break;
case Qt::Key_9:
    nineButton->click();
    break;
case Qt::Key_Period:
    dotButton->click();
    break;
case Qt::Key_Plus:
    addButton->click();
    break;
case Qt::Key_Minus:
    subButton->click();
    break;
case Qt::Key_Asterisk:
    mulButton->click();
    break;
case Qt::Key_Slash:
    divButton->click();
    break;
case Qt::Key_I:
    complexButton->click();
    break;
case Qt::Key_Equal:
    equalButton->click();
    break;
case Qt::Key_Enter:
case Qt::Key_Return:
    equalButton->click();
    break;
case Qt::Key_Control:
    if( isRealMode )
        complexRadioButton->click();
    else
        realRadioButton->click();
```


CuuDuongThanCong.com https://fb.com/tailieudientucntt

```
        break;
    case Qt::Key_C:
        clsHistoryButton->click();
        break;
    case Qt::Key_Delete:
        clearButton->click();
        break;
    case Qt::Key_Backspace:
        backButton->click();
        break;
    case Qt::Key_ParenLeft:
        frontBracketButton->click();
        break;
    case Qt::Key_ParenRight:
        backBracketButton->click();
        break;
    case Qt::Key_PageUp:
        upButton->click();
        break;
    case Qt::Key_PageDown:
        downButton->click();
        break;
    default:
        break;
    }
}
```

+ Comlex.cpp

```
#include"complex.h"

Complex::Complex( double realNum, double imagNum )
    :real ( realNum ), imag ( imagNum )
{
    //the body of funtion is empty
}

const Complex Complex::operator + ( const Complex &right )  //overloading operator '+'
{
    Complex result;
    result.real = this->real + right.real;
    result.imag = this->imag + right.imag;
    return ( result );
}

const Complex Complex::operator - ( const Complex &right )  //overloading operator '-'
{
    Complex result;
```


CuuDuongThanCong.com https://fb.com/tailieudientucntt

```
    result.real = this->real - right.real;
    result.imag = this->imag - right.imag;
    return ( result );
}

const Complex Complex::operator * ( const Complex &right )  //overloading operator '*'
{
    Complex result;
    result.real = this->real * right.real - this->imag * right.imag;
    result.imag = this->real * right.imag + this->imag * right.real;
    return ( result );
}

const Complex Complex::operator / ( const Complex &right )  //overloading operator '/'
{
    Complex result;
    result.real = ( this->real * right.real + this->imag * right.imag ) / ( right.real * right.real
+ right.imag * right.imag );
    result.imag = ( this->imag * right.real - this->real * right.imag ) / ( right.real * right.real
+ right.imag * right.imag );
    return ( result );
}

const Complex &Complex::operator = ( const Complex &right ) //overloading operator '=',
achieve a complex assignment to a complex
{
    this->real = right.real;
    this->imag = right.imag;
    return ( *this );
}

const Complex &Complex::operator = ( const QString &right ) //overloading operator '=',
achieve a string assignment to a complex
{
    QChar ch;
    QString partOfRight;
    QString rightStr;
    int size = right.size();
    int indexOfi = 0;
    bool isReal = true;

    rightStr = right;

    for( int i = 0; i < size; i++ )
    {
        ch = rightStr.at(i);
```


CuuDuongThanCong.com https://fb.com/tailieudientucntt

```
        if( ch == 'i' )
        {
            isReal = false;
            indexOfi = i;
        }
    }

    if( isReal )
    {
        this->real = rightStr.toDouble();
        this->imag = 0.0;
    }
    else
    {
        for( int i = indexOfi - 1; i >= 0; i-- )
        {
            ch = rightStr.at(i);
            if( ( ch == '+' ) || ( ch == '-' ) )
            {
                partOfRight = rightStr.mid( i, indexOfi - i );
                rightStr.remove( i, indexOfi - i + 1 );
                break;
            }
        }
        if( partOfRight.isEmpty() )
        {
            partOfRight = rightStr.mid( 0, indexOfi );
            rightStr.remove( 0, indexOfi + 1 );
        }

        this->imag = partOfRight.toDouble();

        if( !rightStr.isEmpty() )
            this->real = rightStr.toDouble();
        else
            this->real = 0.0;
    }
    return( *this );
}

bool Complex::isZero()  //determine whether the number is zero
{
    if( ( this->real == 0.0 ) && ( this->imag == 0.0 ) )
        return true;
    else
        return false;
```

cuu duong than cong . com

CuuDuongThanCong.com https://fb.com/tailieudientucntt

```
}

double Complex::getReal()   //get the real part of complex
{
   return ( real );
}

double Complex::getImag()   //get the imag part of complex
{
   return ( imag );
}
```

cuu duong than cong . com

cuu duong than cong . com

CuuDuongThanCong.com https://fb.com/tailieudientucntt

Chương 8
LẬP TRÌNH DRIVER

BÀI 1

DRIVER VÀ APPLICATION
TRONG HỆ THỐNG NHÚNG

I. Khái quát về hệ thống nhúng:

Hệ thống nhúng (embedded system) được ứng dụng rất nhiều trong cuộc sống ngày nay. Theo định nghĩa, hệ thống nhúng là một hệ thống xử lý và điều khiển những tác vụ đặc trưng trong một hệ thống lớn với yêu cầu tốc độ xử lý thông tin và độ tin cậy rất cao. Nó bao gồm phần cứng và phần mềm cùng phối hợp hoạt động với nhau, tùy thuộc vào tài nguyên phần cứng mà hệ thống sẽ có phần mềm điều khiển phù hợp. Đôi khi chúng ta thường nhầm lẫn hệ thống nhúng với máy tính cá nhân. Hệ thống nhúng cũng bao gồm phần cứng (CPU, RAM, ROM, USB, ...) và phần mềm (Application, Driver, Operate System, ...). Thế nhưng khác với máy tính cá nhân, các thành phần này đã được rút gọn, thay đổi cho phù hợp với một mục đích nhất định của ứng dụng sao cho tối ưu hóa thời gian thực hiện đáp ứng yêu cầu về thời gian thực (Real-time) theo từng mức độ.

Bài này sẽ đi sâu vào tìm hiểu cấu trúc bên trong phần mềm của hệ thống nhúng nhằm mục đích hiểu được vai trò của driver và application, phân phối nhiệm vụ hoạt động cho hai lớp này sao cho đạt hiệu quả cao nhất về thời gian.

II. Cấu trúc của hệ thống nhúng:

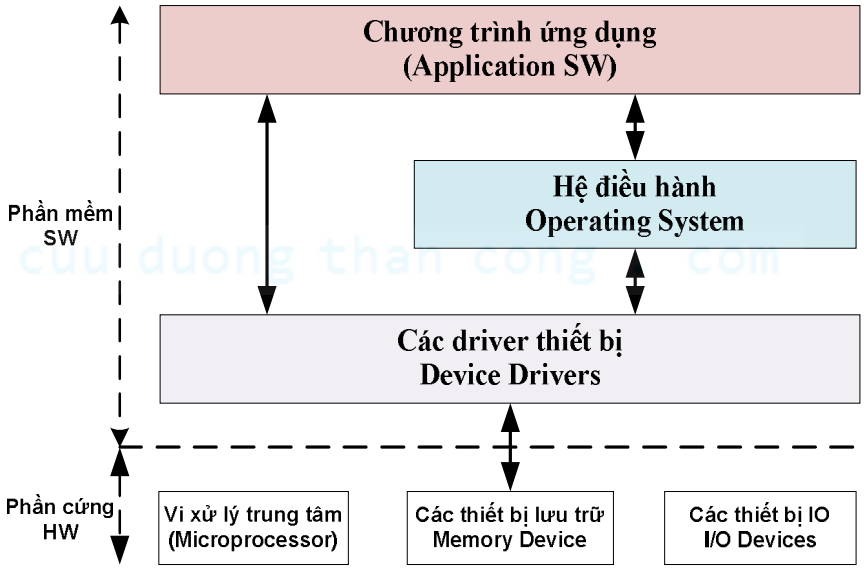
Hệ thống nhúng thông thường bao gồm những thành phần sau: Phần cứng: Bộ vi xử lý trung tâm, bộ nhớ, các thiết bị vào ra; Phần mềm: Các Driver cho thiết bị, Hệ điều hành và các chương trình ứng dụng.

Mối liên hệ giữa các thành phần được minh họa trong sơ đồ hình 3-2.

Thành phần thứ nhất trong hệ thống nhúng là phần cứng. Đây là thành phần quan trong nhất trong hệ thống. Làm nhiệm vụ thực tế hóa những dòng lệnh từ phần mềm yêu cầu. Phần cứng của hệ thống nhúng thường bao gồm những thành phần chính sau:

CuuDuongThanCong.com https://fb.com/tailieudientucntt

- Bộ xử lý trung tâm, làm nhiệm vụ tính toán thực thi các mã lệnh được yêu cầu, được xem như bộ não của toàn hệ thống. Các bộ xử lý trong hệ thống nhúng, không giống như hệ thống máy vi tính cá nhân là những con vi xử lý mạnh chuyên về xử lý dữ liệu, là những dòng vi điều khiển mạnh, được tích hợp sẵn các module ngoại vi giúp cho việc thực thi lệnh của hệ thống được thực hiện nhanh chóng hơn. Hơn nữa tập lệnh của vi điều khiển cũng trở nên gọn nhẹ hơn, ít tốn dung lượng vùng nhớ hơn phù hợp với đặc điểm của hệ thông nhúng. Với những vi điều khiển đã tích hợp sẵn những ngoại vi mạnh, đa dạng thì kích thước mạch phần cứng trong quá trình thi công sẽ giảm rất nhiều. Đây là ưu điểm của hệ thống nhúng so với các hệ thống đa nhiệm khác.



Hình 3-2- Sơ đồ cấu trúc hệ thống nhúng

- Thành phần thứ hai là các thiết bị lưu trữ: Các thiết bị lưu trữ bao gồm có RAM, NAND Flash, NOR Flash, ... mặc dù bên trong vi điều khiển đã tích hợp sẵn ROM và RAM, nhưng những vùng nhớ này chỉ là tạm thời, dung lượng của chúng rất nhỏ, giúp cho việc thực thi những lệnh cũ nhanh hơn. Để lưu trữ những mã lệnh lớn như: Kernel, Rootfs, hay Application thì đòi hỏi phải có những thiết bị lưu trữ lớn hơn. RAM làm nhiệm vụ chứa chương trình thực thi một cách tạm thời. Khi một chương trình được triệu gọi, mã lệnh của chương trình được chép từ các thiết bị lưu trữ khác vào RAM, từ đây từng câu lệnh được biên dịch sẽ lần lượt đi vào vùng nhớ cache bên trong vi xử lý để thực

CuuDuongThanCong.com https://fb.com/tailieudientucntt

thi. Các loại ROM như NAND Flash, NOR Flash, ... thường có dung lượng lớn nhất trong hệ thống nhúng, dùng để chứa những chương trình lớn (hệ điều hành, rootfs, bootstrapcode, ...) lâu dài để sử dụng trong những mục đích khác nhau khi người dùng (hệ điều hành và user) cần sử dụng đến. Chúng tương tự như ổ đĩa cứng trong máy tính cá nhân.

- Các thiết bị vào ra: Đây là những module được tích hợp sẵn bên trong vi điều khiển. Chúng có thể là ADC module, Ethenet module, USB module, ... các thiết bị này có vai trò giao tiếp giữa hệ thống với môi trường bên ngoài.

Thành phần quan trọng thứ hai trong một hệ thống nhúng là phần mềm. Phần mềm của hệ thống nhúng thay đổi theo cấu trúc phần cứng. Hệ thống chỉ hoạt động hiệu quả khi phần mềm và phần cứng có sự tương thích nhau. Đi từ thấp lên cao thông thường phần mềm hệ thống nhúng bao gồm các lớp sau: Driver thiết bị, hệ điều hành, chương trình ứng dụng.

- *Các driver thiết bị (device driver):* Đây là những phần mềm được viết sẵn để trực tiếp điều khiển phần cứng hệ thống nhúng. Mỗi một hệ thống nhúng được cấu tạo từ những phần cứng khác nhau, những vi điều khiển với những tập lệnh khác nhau, những module khác nhau của các hãng khác nhau có cơ chế giao tiếp khác nhau, device driver làm nhiệm vụ chuẩn hóa thành những thư viện chung (có mã lệnh giống nhau), phục vụ cho hệ điều hành và người viết chương trình lập trình dễ dàng hơn. Chẳng hạn, nhiều hệ thống có giao thức truy xuất dữ liệu khác nhau, nhưng device driver sẽ quy về 2 hàm duy nhất mang tên read và write để đọc và nhập thông tin cho hệ thống xử lý. Để phân biệt giữa các thiết bị với nhau, device driver sẽ cung cấp một ID duy nhất cho thiết bị đó nhằm mục đích thuận tiện cho việc quản lý. **Device driver sẽ được trình bày rất rõ trong những bài khác.

- *Hệ điều hành:* Đây cũng là một phần mềm trong hệ thống nhúng, nhiệm vụ của nó là quản lý tài nguyên hệ thống. Bao gồm quản lý tiến trình, thời gian thực, truy xuất vùng nhớ ảo và vùng nhớ vật lý, các giao thức mạng, ...

- *Chương trình ứng dụng:* Các chương trình ứng dụng là do người dùng lập trình. Thông thường trong hệ thống nhúng, công việc lập trình và biên dịch thông thường

CuuDuongThanCong.com https://fb.com/tailieudientucntt

không nằm trên chính hệ thống đó. Ngược lại thường được nằm trên một hệ thống đa nhiệm khác, quá trình này gọi là biên dịch chéo (cross-compile). Sau khi biên dịch xong, chương trình đã biên dịch được chép vào bên trong ROM lưu trữ phục vụ cho quá trình sử dụng sau này. Các chương trình sẽ sử dụng những dịch vụ bên trong hệ điều hành (tạo tiến trình, tạo tuyến, trì hoãn thời gian, ...) và những hàm được định nghĩa trong device driver (giao tiếp thiết bị đầu cuối, truy xuất IO, ...) để tác động đến phần cứng của hệ thống.

***Quyển sách này chủ yếu trình bày sâu về phần mềm hệ thống nhúng. Trong phần đầu chúng ta đã nghiên cứu sơ lược về cách lập trình ứng dụng, làm thế nào để trì hoãn thời gian, tạo tiến trình, tạo tuyến, ... Phần này sẽ đi sâu vào lớp cuối cùng trong phần mềm là Device driver.*

III. Mối quan hệ giữa Device drivers và Applications:

Application (chương trình ứng dụng) và Device drivier (Driver thiết bị) có những điểm giống và khác nhau. Tiêu chí để so sánh dựa vào nguyên lý hoạt động, vị trí, vai trò của từng loại trong hệ thống nhúng.

Application và Device driver khác nhau căn bản ở những điểm sau:

Về cách thức mỗi loại hoạt động, đa số các Application đơn nhiệm vừa và nhỏ hoạt động xuyên suốt từ câu lệnh đầu tiên cho đến câu lệnh kết thúc một cách tuần tự kể từ khi được gọi từ người sử dụng. Trong khi đó, Device driver thì hoàn toàn khác, chúng được lập trình với theo dạng từng module, nhằm mục đích phục vụ cho việc thực hiện một thao tác của Application được gọn nhẹ và dễ dàng hơn. Mỗi module có một hay nhiều chức năng riêng, được lập trình cho một thiết bị đặc trưng và được cài đặt sẵn trên hệ điều hành để sẵn sàng hoạt động khi được gọi. Sau khi được gọi, module sẽ thực thi và kết thúc ngay lập tức. Một cách khái quát, chúng ta có thể xem: Nếu Application là chương trình phục vụ người dùng, thì Device driver là chương trình phục vụ Application. Nghĩa là Application là người dùng của Device driver.

Một điểm khác biệt giữa Application và Device driver là vấn đề an toàn khi thực thi tác vụ. Nếu một Application chỉ đơn giản thực thi và kết thúc, thì công việc của Device driver phức tạp hơn nhiều. Bên cạnh việc thực thi những lệnh được lập trình nó còn phải

CuuDuongThanCong.com https://fb.com/tailieudientucntt

đảm bảo an toàn cho hệ thống khi không còn hoạt động. Nói cách khác, trước khi kết thúc, Device driver phải khôi phục trạng thái trước đó của hệ thống trả lại tài nguyên cho các Device driver khác sử dụng khi cần, tránh tình trạng xung đột phần cứng.

Một Application có thể thực thi những lệnh mà không cần định nghĩa trước đó, các lệnh này chứa trong thư viện liên kết động của hệ điều hành. Khi viết chương trình cho Application, chúng ta sẽ tiếc kiệm được thời gian, cho ra sản phẩm nhan hơn. Trong khi đó, Device driver muốn sử dụng lệnh nào thì đòi hỏi phải định nghĩa trước đó. Việc định nghĩa này được thực hiện khi chúng ta dùng khai báo #include <linux/library.h>, những thư viện này phải thực sự tồn tại, nghĩa là còn ở dạng mã lệnh C chưa biên dịch. Các thư viện này chứa trong hệ thống mã nguồn của hệ điều hành trước khi được biên dịch.

Một chương trình Application đang thực thi nếu phát sinh một lỗi thì không còn hoạt động được nữa. Trong khi đó, khi một tác vụ trong module bị lỗi, nó chỉ ảnh hưởng đến câu lệnh gọi mà thôi (nghĩa là kết quả truy xuất sẽ không đúng) các lệnh tiếp theo sau vẫn có thể tiếp tục thực thi. Thông thường lúc này chúng ta sẽ thoát khỏi chương trình bằng lệnh exit(n), để đảm bảo dữ liệu xử lý là chính xác.

Chúng ta có hai thuật ngữ mới, user space (không gian người dùng) và kernel space (không gian kernel). Không gian ở đây chúng ta nên hiểu là không gian bộ nhớ ảo, do hệ thống Linux định nghĩa và quản lý. Các chương trình ứng dụng Application được thực thi trong user space, còn những Device driver khi được biên dịch thành tập tin .ko sẽ được lưu trữ trong kernel space. Kernel space và User space liên hệ nhau thông qua hệ điều hành (operating system).

Trong khi hầu hết các lệnh trong từng tiến trình và tuyến được thực hiện tuần tự nhau, kết thúc lệnh này rồi mới thực hiện lệnh tiếp theo, trong user space; Thì các module trong device driver có thể cùng một lúc phục vụ đồng thời nhiều tiến trình, tuyến. Do đó Device driver khi lập trình phải đảm bảo giải quyết được vấn đề này tránh tình trạng xung đột vùng nhớ, phần cứng trong quá trình thực thi.

IV.Kết luận:

Chúng ta đã tìm hiểu sơ lược về cấu trúc tổng quát trong hệ thống nhúng, hiểu được vai trò chức năng của từng thành phần. Bên cạnh đó chúng ta cũng đã phân biệt được

CuuDuongThanCong.com https://fb.com/tailieudientucntt

những đặc điểm khác nhau giữa chương trình trong user space và Device Driver trong kernel space. Những kiến thức này rất quan trọng khi bước vào lập trình driver cho thiết bị. Chúng ta phải biết phân công nhiệm vụ giữa user application và kernel driver sao cho đạt hiệu quả cao nhất.

Bài tiếp theo chúng ta sẽ đi vào tìm hiểu các loại driver trong hệ thống Linux, cách nhận dạng từng loại, cũng như các thao tác cần thiết khi làm việc với driver.

cuu duong than cong . com

cuu duong than cong . com

CuuDuongThanCong.com https://fb.com/tailieudientucntt

BÀI 2

PHÂN LOẠI VÀ NHẬN DẠNG DRIVER TRONG LINUX

I. Tổng quan về Device Driver:

Một trong những mục đích quan trọng nhất khi sử dụng hệ điều hành trong hệ thống nhúng là làm sao cho người sử dụng không nhận biết được sự khác nhau giữa các loại phần cứng trong quá trình điều khiển. Nghĩa là hệ điều hành sẽ quy những thao tác điều khiển khác nhau của nhiều loại phần cứng khác nhau thành một thao tác điều khiển chung duy nhất. Ví dụ như, hệ điều hành quy định tất cả những ổ đĩa, thiết bị vào ra, thiết bị mạng đều dưới dạng tập tin và thư mục. Việc khởi động hay tắt thiết bị chỉ đơn giản là đóng hay mở tập tin (thư mục) đó còn sau khi thao tác đóng hay mở hệ điều hành làm gì đó là công việc của device driver.

Trong một hệ thống nhúng, không phải chỉ có CPU mới có thể xử lý thông tin mà tất cả những thiết bị phần cứng đều có một cơ cấu điều khiển được lập trình sẵn, đặc trưng cho từng thiết bị. Mỗi một thẻ nhớ, USB, chuột, USB Camera, … điều là những hệ thống nhúng độc lập, chúng có từng nhiệm vụ riêng, đảm trách một công việc xử lý thu thập thông tin cụ thể. Mỗi bộ điều khiển của các thiết bị đó điều chứa những thanh ghi lệnh và thanh ghi trạng thái. Và để điều khiển được thì chúng ta phải cung cấp những số nhị phân cần thiết vào thanh ghi lệnh, đọc thanh ghi trạng thái cho biết trạng thái thực hiện. Tương tự khi muốn thu thập dữ liệu, chúng ta phải cung cấp những mã cần thiết, theo những bước cần thiết do nhà sản xuất quy định. Thay vì phải làm những công việc nhàm chán đó, chúng ta sẽ giao cho device driver đảm trách. Device driver thực chất là những hàm được lập trình sẵn, nạp vào hệ điều hành. Có ngõ vào là những giao diện chung, ngõ ra là những thao tác riêng biệt điều khiển từng thiết bị của device driver đó.

Linux cung cấp cho chúng ta 3 loại device driver: Character driver, block driver, và network driver. Character driver hoạt động theo nguyên tắc truy xuất dữ liệu không có vùng nhớ đệm, nghĩa là thông tin sẽ di chuyển lập tức từ nơi gửi đến nơi nhận theo từng byte. Block driver thì khác, hoạt động theo cơ chế truy xuất dữ liệu theo vùng nhớ đệm.

CuuDuongThanCong.com https://fb.com/tailieudientucntt

Có hai vùng nhớ đệm, đệm ngõ vào và đệm ngõ ra. Dữ liệu trước khi di chuyển vào (ra) hệ thống phải chứa trong vùng nhớ đệm, cho đến khi vùng nhớ đệm đầy thì mới được phép xuất (nhập). Nghĩa là dữ liệu di chuyển theo từng khối. Network driver hoạt động theo một cách riêng dạng socket mạng, chủ yếu dùng trong truyền nhận dữ liệu từ xa giữa các máy với nhau trong mạng cục bộ hay internet bằng các giao thức mạng phổ biến.

**Trong suốt phần này chúng ta chủ yếu nghiên cứu về character driver. Mục tiêu là có thể tự mình thiết kế một character driver đơn giản;

II. Các đặc điểm của device driver trong hệ điều hành Linux:

Chúng ta đã biết như thế nào là device driver, đặc điểm của từng loại device driver. Thế nhưng các loại driver này được hệ điều hành Linux quản lý như thế nào?

Bất kỳ một thiết bị nào trong hệ điều hành Linux cũng được quản lý thông qua tập tin và thư mực hệ thống. Chúng được gọi là các tập tin thiết bị hay là các tập tin hệ thống. Những tập tin này điều chứa trong thư mục /dev. Trong thư mục /dev chúng ta thực hiện lệnh ls –l, hệ thống sẽ cho ra kết quả sau:

```
crw-rw-rw-    1 root     root       1,   3 Apr 11  2002 null
crw-------    1 root     root      10,   1 Apr 11  2002 psaux
crw-------    1 root     root       4,   1 Oct 28 03:04 tty1
crw-rw-rw-    1 root     tty        4,  64 Apr 11  2002 ttys0
crw-rw----    1 root     uucp       4,  65 Apr 11  2002 ttyS1
crw--w----    1 vcsa     tty        7,   1 Apr 11  2002 vcs1
crw--w----    1 vcsa     tty        7, 129 Apr 11  2002 vcsa1
crw-rw-rw-    1 root     root       1,   5 Apr 11  2002 zero
...
```

Cột thứ nhất cho chúng ta thông tin về loại device driver. Theo thông tin trên thì tất cả đều là character driver vì những ký tự đầu tiên điều là “c”, tương tự nếu là block driver thì ký tự đầu là “b”. Chúng ta chú ý đến cột thứ 4 và 5, tại đây có hai thông tin cách nhau bằng dấu “,” hai số này được gọi là Major và Minor. Mỗi thiết bị trong hệ điều hành đều có một số 32 bits riêng biệt để quản lý. Số này được chia thành hai thông tin, thông tin thứ nhất là Major number. Major number là số có 12 bit, dùng để phân biệt từng nhóm thiết bị với nhau, hay nói cách khác những thiết bị cùng loại sẽ có chung một số Major.

CuuDuongThanCong.com https://fb.com/tailieudientucntt

Các thiết bị cùng loại có cùng số Major được phân biệt nhau thông qua thông tin thứ hai là số Minor. Số Minor là số có chiều dài 20 bit. Với hai số Major và Minor, tổng cộng hệ điều hành có thể quản lý số thiết bị tối đa là $2^{12}*2^{20}$ tương đương với 2^{32}.

Trong lập trình driver, đôi khi chúng ta muốn thao tác với hai thông tin Major và Minor numbers. Kernel cung cấp cho chúng ta những hàm rất hữu ích để thực hiện những công việc này. Sau đây là một số hàm tiêu biểu:

```
#include <linux/types.h>
#include <linux/kdev_t.h>
int MAJOR (dev_t dev);
int MINOR (dev_t dev);
dev_t MKDEV (int major, int minor);
```

Trước khi sử dụng những hàm này, chúng ta phải khai báo thư viện phù hợp cho chúng. thư viện <linux/types.h> chứa định nghĩa kiểu dữ liệu dev_t, biến kiểu này dùng để chứa số định danh cho thiết bị. Thư viện <linux/kdev_t.h> chứa định nghĩa cho những hàm MAJOR(), MINOR(), MKDEV, …

cuu duong than cong . com

Hàm MAJOR (dev_t dev) dùng để tách số Major của thiết bị dev_t dev và lưu vào một biến kiểu int;

Hàm MINOR (dev_t dev) dùng để tách số Minor của thiết bị dev_t dev và lưu vào một biến kiểu int;

Hàm MKDEV (int major, int minor) dùng để tạo thành một số định danh thiết bị kiểu dev_t từ hai số int major, và int minor;

Đối với kernel 2.6 trở đi thì số device driver dev_t có 32 bit. Nhưng đối với những kernel đời sau đó thì dev_t có 16 bit.

cuu duong than cong . com

CuuDuongThanCong.com https://fb.com/tailieudientucntt

III. Kết luận:

Trong bài này chúng ta đã đi vào tìm hiểu một cách khái quát vai trò ý nghĩa của từng loại device driver trong hệ thống Linux, mỗi device driver đều có những ưu và nhược điểm riêng và đóng góp một phần để làm cho hệ thống chạy ổn định. Chúng ta cũng đã biết cách thức quản lý thông tin thiết bị của Linux thông qua thư mục /dev, mỗi thiết bị trong Linux đều có một số định danh, tùy vào từng hệ thống mà số này có bao nhiêu bit, số định danh có thể được tạo thành từ hai số riêng biệt Major và Minor bằng hàm MKDEV() hoặc có thể tách riêng một số định danh dev_t thành hai số Major và Minor bằng hai hàm MAJOR() và MINOR (). Những hàm này rất quan trong trong lập trình driver.

Trong giới hạn về thời gian, quyển sách này chỉ trình bày cho các bạn cách lập trình một character driver. Trên cơ sở đó các bạn sẽ tự mình tìm hiểu cách lập trình cho các loại driver khác. Bài sau chúng ta sẽ tìm hiểu sâu hơn về character driver, cấu trúc dữ liệu bên trong, các hàm thao tác khởi tạo character device, …

cuu duong than cong . com

cuu duong than cong . com

CuuDuongThanCong.com https://fb.com/tailieudientucntt

BÀI 3

CHARACTER DEVICE DRIVER

I. Tổng quan character device driver:

Character device driver là một trong 3 loại driver trong hệ thống Linux. Đây là driver dễ và phổ biến nhất trong các ứng dụng giao tiếp vừa và nhỏ đối với lập trình nhúng. Character driver và các loại driver khác đều được hệ điều hành quản lý dưới dạng tập tin và thư mục. Hệ điều hành sử dụng các hàm truy xuất tập tin chuẩn để giao tiếp trao đổi thông tin giữa người lập trình và thiết bị do driver điều khiển. Chẳng hạn những hàm như `read, write, open, release, close, …` được dùng chung cho tất cả các character driver, những hàm này còn được gọi là giao diện điều khiển giữa hệ điều hành (được người lập trình ra lệnh) và device driver (được hệ điều hành ra lệnh). Hoạt động bên trong giao diện này là những thao tác của từng device driver đặc trưng cho từng thiết bị đó. Công việc lập trình các thao tác này gọi là lập trình driver.

Một character driver muốn cài đặt và hoạt động bình thường thì phải trải qua nhiều bước lập trình. Đầu tiên là đăng ký số định danh cho driver, số định danh là số mà hệ điều hành linux cung cấp cho mỗi driver để quản lý. Tiếp theo, mô tả tập lệnh mà driver hỗ trợ, chúng ta có thể xem tập lệnh là những thao tác hoạt động bên trong của driver dùng để điều khiển một thiết bị vật lý. Sau khi đã mô tả tập lệnh, chúng ta sẽ liên kết các tập lệnh này với các giao diện chuẩn mà hệ điều hành linux hỗ trợ, nhằm mục đích giao tiếp giữa hệ điều hành và các thiết bị ngoại vị vật lý mà driver điều khiển. Tiếp theo chúng ta định nghĩa liên kết các giao diện này với cấu trúc mô tả tập tin khi thiết bị được mở. Cuối cùng chúng ta thực hiện cài đặt driver thiết bị vào hệ thống thư mục tập tin linux, thông thường nằm trong thư mục /dev.

Trong phần này chúng ta sẽ tìm hiểu một cách chi tiết các bước lập trình driver đã nêu.

II. Số định danh character driver:

Thế nào là số định danh, đặc diểm và vai trò của số định danh của character driver cũng hoàn toàn tương tự như device driver khác mà chúng ta đã nghiên cứu rất kỹ trong

CuuDuongThanCong.com https://fb.com/tailieudientucntt

bài trước. Chúng ta cũng đã biết những hàm rất quan trọng để thao tác với số định danh này. Ở đây không nhắc lại mà thêm vào đó là làm thế nào để tạo lập một số định danh cho thiết bị nào đó mà không sinh ra lỗi.

1. *Xác định số định danh hợp lệ cho thiết bị mới theo cách thông thường:*

Một trong những công việc quan trong đầu tiên cần phải làm trong driver là xác định số định danh cho thiết bị. Có hai thông tin cần xác định là số *Major* và số *Minor*.

Trước hết chúng ta phải biết số định danh nào còn trống trong hệ thống chưa được các thiết bị khác sử dụng. Thông tin về số định danh được linux sử dụng chứa trong tập tin *Documentation/devices.txt*. Tập tin này chứa số định danh, tên thiết bị, thời gian tạo lập, loại thiết bị, … đã được linux sử dụng hay sẽ dùng cho những mục đích đặc biệt nào đó. Đọc nội dung trong tập tin này, chúng ta sẽ tìm được số định danh phù hợp, và công việc tiếp theo là đăng ký số định danh đó vào linux.

Linux kernel cung cấp cho chúng ta một hàm dùng để đăng ký số định danh cho thiết bị, hàm đó là:

```
#include <linux/fs.h>
int register_chrdev_region (dev_t first, unsigned int count,
char *name);
```

Để sử dụng được hàm, chúng ta phải khai báo thư viện `<linux/fs.h>`. Tham số thứ nhất `dev_t first` là số định danh thiết bị đầu tiên muốn đăng ký với số Major là số hợp lệ chưa được sử dụng, Minor thông thường cho bằng 0. Tham số thứ hai `unsigned int count` là số thiết bị muốn đăng ký, chẳng hạn muốn đăng ký 1 thiết bị thì ta nhập 1, lúc này chỉ có một thiết bị mang số định danh là `dev_t first` được đăng ký. Tham số thứ ba `char *name` là tên thiết bị muốn đăng ký.

Hàm `register_chrdev_region ()` trả về giá trị kiểu int là 0 nếu quá trình đăng ký thành công. Và trả về số mã lỗi âm khi quá trình đăng ký không thành công.

Tất cả những thông tin khi đăng ký thành công sẽ được hệ điều hành chứa trong tập tin `/proc/devices` và `sysfs` khi quá trình cài đặt thiết bị kết thúc.

Cách đăng ký số định danh trên có một nhược điểm lớn là chỉ áp dụng khi người đăng ký đồng thời là người lập trình nên driver đó vì thế họ sẽ biết rõ số định danh nào là còn

CuuDuongThanCong.com https://fb.com/tailieudientucntt

trống. Khi driver được sử dụng trên những máy tính khác, thì số định danh được chọn có thể bị trùng với các driver khác. Vì thế việc lựa chọn một số định danh động là cần thiết. Vì số định danh động sẽ không trùng với bất kỳ số định danh nào tồn đang tồn tại trong hệ thống.

Ví dụ, nếu muốn đăng ký một character driver có tên là "lcd_dev", số lượng là 1, số Major đầu tiên là 2, chúng ta tiến hành khai báo hàm như sau:

```
/*Khai báo biến lưu trữ mã lỗi trả về của hàm*/
int res;
/*Thực hiện đăng ký thiết bị cho hệ thống*/
res = register_chrdev_region (2, 1, "lcd_dev");
if (res < 0) {
   printk ("Register device error!");
   exit (1);
}
```

2. Xác định số định danh cho thiết bị theo cách ngẫu nhiên:

Linux cung cấp cho chúng ta một hàm đăng ký số định danh động cho driver thiết bị mới.

```
#include <linux/fs.h>
int alloc_chrdev_region (dev_t *dev, unsigned int firstminor,
unsigned int count, char *name);
```

Cũng tương tự như hàm `register_chrdev_region ()`, hàm `alloc_chrdev_region ()` cũng làm nhiệm vụ đăng ký định danh cho một thiết bị mới. Nhưng có một điểm khác biệt là số Major trong định danh không còn cố định nữa, số này do hệ thống linux tự động cấp vì thế sẽ không trùng với bất kỳ số định danh nào khác đã tồn tại.

Tham số thứ nhất của hàm, `dev_t *dev`, là con trỏ kiểu `dev_t` dùng để lưu trữ số định danh đầu tiên trong khoảng định danh được cấp nếu hàm thực hiện thành công;

Tham số thứ hai, `unsigned int first minor`, là số Minor đầu tiên của khoảng định danh muốn cấp;

CuuDuongThanCong.com https://fb.com/tailieudientucntt

Tham số thứ ba, `unsigned int count`, là số lượng định danh muốn cấp, tính từ số Major được cấp động và số Minor `unsigned int first minor`;

Tham số thứ tư, `char *name`, là tên của driver thiết bị muốn đăng ký.

Ví dụ khi muốn đăng ký thiết bị tên "`lcd_dev`", số Minor đầu tiên là 0, số thiết bị muốn đăng ký là 1, số định danh khi tạo ra được lưu vào biến dev_t dev_id. Lúc này hàm `alloc_chrdev_region ()` khai báo như sau:

```
/*Khai báo biến dev_t để lưu giá trị định danh đầu tiên trả về của hàm*/
dev_t dev_id;
/*Khai báo biến lưu mã lỗi trả về của hệ thống*/
int res;
/*Thực hiện đăng ký thiết bị với định danh động*/
res = alloc_chrdev_region (&dev_id, 0, 1, "lcd_dev");
/*Kiểm tra mã lỗi trả về*/
if ( res < 0) {
   printk ("Allocate devices error!");
   return res;
}
```

Tuy nhiên viêc dăng ký số định danh động cho thiết bị đôi khi cũng có nhiều bất lợi. Giả sử số định danh của thiết bị cần được sử dụng cho những mục đích khác, vì thế số định danh luôn thay đổi khi mỗi lần cài đặt driver sẽ sinh ra lỗi trong quá trình thực thi lệnh. Để kết hợp ưu điểm của 2 phương pháp, chúng ta sẽ đăng ký driver thiết bị theo cách sau:

```
/*Khai báo các biến cần thiết*/
int lcd_major;  //Biến lưu trữ số Major
int lcd_minor; //Biến lưu trữ số Minor
dev_t dev_id;  //Biến lưu trữ số định danh thiết bị
int result;  //Biến lưu mã lỗi
/*Nếu số Major hợp lệ, đã tồn tại*/
if (lcd_major) {
```


CuuDuongThanCong.com https://fb.com/tailieudientucntt

```
        dev = MKDEV(lcd_major, lcd_minor); //Tạo số định danh
/*Đăng ký thiết bị với số định danh cố định*/
        result = register_chrdev_region (dev, lcd_nr_devs,
        "lcd_dev");
} else {
/*Nếu số Major chưa tồn tại, thực hiện tìm kiếm số Major động*/
result = alloc_chrdev_region(&dev_id, lcd_minor, lcd_nr_devs,
"lcd_dev");
/*Cập nhật lại số Major động cần sử dụng trong những lần sau*/
lcd_major = MAJOR (dev_id);
}
/*Kiểm tra kết quả thực thi của hai lệnh trên*/
if (result < 0) {
    printk(KERN_WARNING "lcd: can't get major %d\n",
lcd_major);
    return result;
}
```

Như vậy ta có thể cập nhật lại số định danh động khi vừa tạo ra để sử dụng cho những chương trình liên quan bằng kỹ thuật như trong đoạn mã lệnh trên.

Character driver bao gồm có nhiều thành phần, đăng ký số định danh chỉ là một trong những thành phần đó. Bên cạnh số định danh character driver còn có những bộ phận như: Cấu trúc dữ liệu (data structure) được gọi là file_operation, cấu trúc này chứa những tập lệnh được người lập trình driver định nghĩa; Cấu trúc mô tả tập tin (file) chứa những thông tin cơ bản của tập tin thiết bị; Cấu trúc tập tin chứa thông tin quản lý tập tin thiết bị trong hệ thống linux.

Phần tiếp theo chúng ta sẽ tìm hiểu cách gán các hành vi cho character device driver thông qua việc thao tác với file_operations.

III. Cấu trúc lệnh của character driver:

Cấu trúc lệnh của character driver (file_operations) là một cấu trúc dùng để liên kết những hàm chứa các thao tác của driver điều khiển thiết bị với những hàm chuẩn trong hệ

CuuDuongThanCong.com https://fb.com/tailieudientucntt

điều hành giúp giao tiếp giữa người lập trình ứng dụng với thiết bị vật lý. Cấu trúc *file_operation* được định nghĩa trong thư viện `<linux/fs.h>`. Mỗi một tập tin thiết bị được mở trong hệ điều hành linux đều được hệ điều hành dành cho một vùng nhớ mô tả cấu trúc tập tin, trong cấu trúc tập tin có rất nhiều thông tin liên quan phục vụ cho việc thao tác với tập tin đó (chúng ta sẽ nghiên cứu kỹ trong phần sau). Một trong những thông tin này là *file_operations*, dùng mô tả những hàm mà driver thiết bị đang được mở hỗ trợ. Có thể nói một cách khác mỗi tập tin thiết bị trong hệ thống linux tương tự như một vật thể và *file_operation* là những công dụng của vật thể đó.

Cấu trúc *file_operations* là một thành phần trong cấu trúc *file_structure* khi tập tin thiết bị được mở. Mỗi thành phần trong file_*operations* bao gồm những lệnh căn bản theo chuẩn do hệ điều hành định nghĩa, nhưng những lệnh này chưa được định nghĩa thao tác cụ thể, đây là nhiệm vụ của người lập trình driver. Chúng ta phải liên kết những thao tác muốn lập trình với những dạng hàm chuẩn này.

Sau đây chúng ta sẽ tìm hiểu một số những thành phần quan trong trong cấu trúc *file_structure*:

```
struct module *owner
```

Đây không phải là một lệnh trong driver mà chỉ là con trỏ cho biết tên driver nào quản lý những lệnh được liên kết. Thông tin này được thiết lập thông qua macro THIS_MODULE định nghĩa trong thư viện `<linux/module.h>`.

```
ssize_t (*read) (struct file *, char __user *, size_t, loff_t
*);
```

Hàm chuẩn này dùng để yêu cầu nhận dữ liệu từ thiết bị vật lý. Nhận dữ liệu như thế nào sẽ do người lập trình quyết định, phù hợp với quy định của từng thiết bị. Tham số thứ nhất, `struct file *`, là con trỏ đến cấu trúc tập tin đang mở trong hệ điều hành, dùng để phân biệt thiết bị này với thiết bị khác. Tham số thứ hai, `char __user *`, là con trỏ được khai báo trong user space, chứa thông tin đọc được từ thiết bị. Tham số thứ ba, size_t là kích thước dữ liệu muốn đọc (tính bằng byte). Tham số thứ tư, `loff_t *`, là con trỏ chỉ vị trí dữ liệu trong thiết bị cần đọc về, nếu để trống thì mặc định là vị trí đầu tiên. Hàm có giá trị trả về là kích thước dữ liệu đọc về thành công.

CuuDuongThanCong.com https://fb.com/tailieudientucntt

```
ssize_t (*write) (struct file *, const char __user *, size_t,
loff_t *);
```

Hàm này dùng để ghi thông tin của người dùng vào thiết bị vật lý. Các thao tác ghi cụ thể sẽ do người lập trình quyết định tùy theo từng thiết bị phần cứng. Tham số thứ nhất, `struct file *`, là con trỏ đến cấu trúc tập tin đang mở trong hệ điều hành, dùng để phân biệt thiết bị này với thiết bị khác khi sử dụng nhiều thiết bị. Tham số thứ hai, `char __user *`, là con trỏ được khai báo trong user space, chứa thông tin muốn ghi từ người sử dụng. Tham số thứ ba, `size_t` là kích thước dữ liệu muốn ghi (tính bằng byte). Tham số thứ tư, `loff_t *`, là con trỏ chỉ địa chỉ dữ liệu trong thiết bị cần ghi thông tin, nếu để trống thì mặc định là vị trí đầu tiên. Hàm có giá trị trả về là kích thước dữ liệu ghi thành công.

```
int (*open) (struct inode *, struct file *);
```

Đây là hàm luôn được thực thi khi thao tác với driver. Hàm được gọi khi ta sử dụng lệnh mở tập tin driver thiết bị sử dụng. Chúng ta không cần thiết phải lập trình thao tác cho lệnh này. Trong cấu trúc lệnh có thể đặt giá trị `NULL`, như vậy khi đó driver sẽ không được cảnh báo khi thiết bị được mở. Mặc dù không quan trọng nhưng chúng ta nên khai báo lệnh open_device trong chương trình để sử dụng mã lỗi trả về khi cần thiết.

```
int (*release) (struct inode *, struct file *);
```

Hàm chuẩn này dược thực thi khi driver thiết bị không còn sử dụng, thoát khỏi hệ thống linux. Cũng tương tự như hàm open, hàm release có thể không cần khai báo trong cấu trúc tập lệnh file_operation. Tuy nhiên để thuận lợi trong quá trình lập trình, chúng ta nên khai báo hàm release_device trong driver để có thể trả về mã lỗi nếu cần thiết.

```
int (*ioctl) (struct inode *, struct file *, unsigned int,
unsigned long);
```

ioctl là một hàm rất mạnh trong cấu trúc tập lệnh `file_operations`. Hàm này có thể tích hợp nhiều hàm khác do người lập trình driver định nghĩa. Những hàm khác nhau được phân biệt thông qua các tham số của hàm ioctl. Tham số thứ nhất, `struct inode *`, là cấu trúc tập tin trong hệ thống thư mục linux (chúng ta sẽ nghiên cứu trong phần sau). Tham số thứ hai, `struct file *`, là cấu trúc tập tin đang mở trong hệ thống

CuuDuongThanCong.com https://fb.com/tailieudientucntt

linux. Tham số thứ ba, `unsigned int`, là số `unsigned int` phân biệt những lệnh khác nhau, có thể gọi đây là số định danh lệnh. tham số thứ ba, dạng `unsigned long`, là tham số của hàm tương ứng với số định danh lệnh. Chúng ta sẽ nghiên cứu sâu các sử dụng hàm trong những bài sau.

Sau đây là một ví dụ cho thấy cách gán chức năng cho các hàm sử dụng trong tập lệnh của character device driver.

```
/*Khai báo cấu trúc lệnh cho driver*/
struct file_operations lcd_fops = {
/*Tên của module sở hữu tập lệnh này*/
.owner =    THIS_MODULE,
/*Gán lệnh đọc lcd_read vào hàm chuẩn read*/
.read =     lcd _read,
/*Gán hàm ghi dữ liệu vào hàm chuẩn write*/
.write =    lcd _write,
/*Gán hàm lcd_ioctl vào hàm chuẩn ioctl*/
.ioctl =    lcd _ioctl,
/*Gán hàm khởi động thiết bị vào hàm chuẩn, có thể đặt giá trị NULL*/
.open =     lcd _open,
/*Gán hàm thoát thiết bị vào hàm chuẩn, có thể đặt giá trị  NULL*/
.release =  lcd _release,
};
```

Tiếp theo chúng ta sẽ nghiên cứu cấu trúc khác lớn hơn trong character device driver. Cấu trúc này chứa những thông tin thao tác tập tin cần thiết khi tập tin đang mở trong đó có cấu trúc tập lệnh *file_operations*.

IV. Cấu trúc mô tả tập tin của character driver:

Cấu trúc mô tả tập tin (file_structure), định nghĩa trong thư viện `<linux/fs.h>` là cấu trúc quan trọng thứ hai trong *character device driver*. Cấu trúc này không xuất hiện trong hệ thống thư mục tập tin của Linux. Mà chỉ xuất hiện khi tập tin được mở, sử dụng

CuuDuongThanCong.com https://fb.com/tailieudientucntt

trong hệ thống. Khi một tập tin được mở, linux sẽ cung cấp một không gian vùng nhớ lưu trữ những thông tin quan trọng phục vụ cho quá trình lập trình sử dụng tập tin. Những thông tin đó là:

```
mode_t f_mode;
```

Thông tin này quy định chế độ truy xuất tập tin thiết bị. Một tập tin khi được mở trong hệ thống sẽ có thể chỉ được phép đọc, chỉ được phép ghi, hay cả hai bằng cách sử dụng các bit cờ `FMODE_READ` và `FMODE_WRITE`. Chúng ta nên kiểm tra chế độ truy xuất của tập tin thiết bị khi sử dụng hàm ioclt hay open. Nhưng khi sử dụng hàm read và write thì không cần thiết. Vì trước khi thực thi các hàm này hệ thống sẽ tự động kiểm tra các cờ hợp lệ hay không, nếu không hệ thống sẽ bỏ qua không thực thi.

```
loff_t f_pos;
```

Là thông tin lưu vị trí truy cập tập tin, phục vụ cho thao tác `read` và `write`. Đây là số có 64 bits, khả năng truy xuất rất rộng. Người lập trình driver có thể tham khảo thông tin này để biết vị trí hiện tại của con trỏ truy cập tập tin. Tuy nhiên nên hạn chế thay đổi thông tin này. Để thay đổi thông tin này, chúng ta có thể thay đổi trực tiếp bằng cách thay đổi tham số `filp -> f_pos` hoặc có thể sử dụng những hàm chuẩn trong linux.

```
unsigned int f_flags;
```

Đây là những cờ thể hiện chế độ truy cập tập tin, bao gồm những giá trị có thể như, `O_RDONLY`, `O_NONBLOCK`, `O_SYNC` trong những thông tin này thì `O_NONBLOCK` được sử dụng nhiều nhất để kiểm tra lệnh thực hiện có phải là lệnh truy xuất theo block hay không. Còn những thông tin truy xuất khác thông thường được kiểm tra thông qua `f_mode`. Các định nghĩa cho giá trị bit cờ chứa trong thư viện `<linux/fcnt.h>`

```
struct file_operations *f_op;
```

Thông tin này chứa định nghĩa các tập lệnh tương ứng của từng tập tin thiết bị. Thông tin này đã được giải thích rõ trong phần trên.

```
void *private_data;
```

Đây là con trỏ đến vùng nhớ dành riêng cho người sử dụng driver. Vùng nhớ này được xóa khi tập tin được mở, nhưng vẫn tồn tại khi tập tin được đóng, vì thế chúng ta phải tiến hành giải phóng vùng nhớ này trước khi thoát.

CuuDuongThanCong.com https://fb.com/tailieudientucntt

```
struct dentry *f_dentry;
```

Chứa thông tin về tập tin nguồn được mở, mỗi tập tin được mở trong hệ thống linux đều bắt nguồn từ một tập tin nào đó lưu trong bộ nhớ. Người viết driver thường dùng thông tin này hơn là thông tin về i_node của thiết bị để quản lý vị trí tập tin thiết bị được mở trong hệ thống.

Trong thực tế một tập tin tổng quát được mở trong hệ thống có thể có nhiều hơn những thông tin nêu trên. Nhưng đối với tập tin driver thì những thông tin đó không cần thiết. Tất cả những driver điều thao tác trên cơ sở những cấu trúc tập tin được xây dựng sẵn.

V. Cấu trúc tập tin của character driver:

Cấu trúc tập tin (*inode structure*) được kernel sử dụng để đặc trưng cho một tập tin driver thiết bị. Cấu trúc này hoàn toàn khác với cấu trúc file structure được giải thích trong phần trước, điều này có nghĩa là có thể có nhiều *file structure* biểu thị cấu trúc tập tin đang mở nhưng tất cả những *file structure* này điều có nguồn gốc từ một *inode structure* duy nhất.

Kernel dùng cấu trúc file structure này để biểu diễn một tập tin thiết bị trong cấu trúc hệ thống của mình (hay nói cụ thể hơn là cấu trúc cây thư mục). Chúng ta có thể mở tập tin này với nhiều chế độ truy xuất khác nhau, mỗi chế độ truy xuất sẽ tương đương với một cấu trúc *file structure*. Cấu trúc *inode structure* chứa rất nhiều thông tin về tập tin thiết bị, trong công việc lập trình driver chúng ta chỉ quan tâm đến những thông tin sau đây:

```
dev_t i_rdev;
```

Mỗi một cấu trúc inode structure đại diện cho một tập tin thiết bị, thông tin này trong inode structure chứa số định danh thiết bị mà chúng ta đã tạo trong phần trước.

```
struct cdev *i_cdev;
```

`struct cdev` là kiểu cấu trúc lưu trữ thông tin của một tập tin lưu trữ trong kernel. Và thông tin `i_cdev` là con trỏ đến cấu trúc này.

Linux cung cấp cho chúng ta hai hàm chuẩn để tìm số định danh Major và Minor của thiết bị biểu thị bằng inode structure. Hai hàm đó là:

CuuDuongThanCong.com https://fb.com/tailieudientucntt

```
unsigned int iminor(struct inode *inode);
unsigned int imajor(struct inode *inode);
```

Chúng ta nên dùng hàm này để lấy thông tin về số định danh thiết bị bên cạnh việc truy cập trực tiếp thông tin `dev_t i_rdev` trong cấu trúc inode structure.

VI. Cài đặt character device driver vào hệ thống Linux:

Sau khi đăng ký số định danh thiết bị, thiết lập liên kết với *file_operations*, gán *file_operations* với *file_structure*, và định nghĩa một *inode_structure*, chúng ta đã hoàn thành cơ bản một *character device driver*. Công việc cuối cùng là tiến hành cài đặt *character driver* này vào hệ điều hành và sử dụng. Phần này sẽ trình bày cho chúng ta các bước để cài đặt những cấu trúc trên vào *kernel* trở thành một *character driver* hoàn chỉnh.

Những hàm được giới thiệu sau đây được định nghĩa trong thư viện `<linux/cdev.h>`.

Trước khi cài đặt thông tin vào kernel chúng ta phải khai báo cho kernel dành ra một không gian vùng nhớ riêng, chuẩn bị cho quá trình cài đặt. Có hai cách thực hiện công việc này.

- ***Cách 1:*** Khai báo cấu trúc trước khi định nghĩa thông tin.

```
struct cdev *my_cdev = cdev_alloc();
my_cdev->ops = &my_fops;
...
(Tương tự cho những trường khác)
```

Đầu tiên chúng ta khai báo con trỏ cấu trúc dạng `struc cdev` để chứa con trỏ struct cdev trống do hàm `cdev_alloc()` trả về. Tiếp theo, định nghĩa từng thông tin liên quan cho cấu trúc vừa tạo ra. Chẳng hạn, trong câu lệnh trên, chúng ta cập nhật con trỏ cấu trúc lệnh cho cdev này bằng lệnh gán cơ bản `my_cdev->ops = &my_fops;` trong đó `&my_fops` là cấu trúc lệnh đã được tạo thành từ trước được gán vào trường ops của cấu trúc `my_cdev`.

- ***Cách 2:*** Thực hiện định nghĩa thông tin trước khi khai báo cho kernel.

```
struct cdev *my_cdev;
```


CuuDuongThanCong.com https://fb.com/tailieudientucntt

```
my_cdev->ops = &my_ops;
...
(Tiếp theo cho những trường khác)
cdev_init ( my_cdev, my_ops);
```

Như vậy, sau khi định nghĩa xong các trường cần thiết cho cấu trúc struct cdev, chúng ta tiến hành gọi hàm `cdev_init ();` để thông báo cho kernel dành ra một vùng nhớ riêng lưu trữ cdev mới vừa tạo ra.

Cấu trúc của hàm `cdev_init ()` như sau:

```
void cdev_init(struct cdev *cdev, struct file_operations *fops);
```

Với `struct cdev *cdev` là con trỏ cấu trúc lưu thông tin driver đã được khai báo, `struct file_operations *fops` là cấu trúc tập lệnh của driver.

Sau khi khai báo cho kernel dành một vùng nhớ lưu trữ cấu trúc driver, công việc cuối cùng là triệu gọi hàm `cdev_add ()` để cài đặt cấu trúc này vào kernel. Cấu trúc của hàm cdev_add như sau:

cuu duong than cong . com

```
int cdev_add(struct cdev *dev, dev_t num, unsigned int count);
```

Trong đó, `struct cdev *dev` là con trỏ cấu trúc driver cần cài đặt. `dev_t num` là số định danh thiết bị đầu tiên muốn cài đặt vào hệ thống, số định danh này được xác định phù hợp với hệ thống thông qua các hàm đặc trưng. Tham số cuối cùng, `unsigned int count`, là số thiết bị muốn cài đặt vào kernel.

Hàm này sẽ trả về giá trị 0 nếu quá trình cài đặt driver vào kernel thành công. Ngược lại sẽ trả về mã lỗi âm. Kernel chỉ có thể gọi được những hàm trong driver khi nó được cài đặt thành công vào hệ thống.

Đôi khi chúng ta muốn tháo bỏ cấu trúc device driver ra khỏi hệ thống, linux cung cấp cho chúng ta hàm `cdev_del()` để thực hiện công việc này. Cấu trúc của hàm như sau:

cuu duong than cong . com

```
void cdev_del(struct cdev *dev);
```

Với `cdev *dev` là cấu trúc driver muốn tháo bỏ, khi driver được tháo bỏ, thì kernel không thể sử dụng những hàm định nghĩa trong kernel được nữa.

VII. Tổng kết:

CuuDuongThanCong.com https://fb.com/tailieudientucntt

Trên đây chúng ta đã tìm hiểu rất kỹ những thành phần cấu tạo nên một character driver do hệ thống linux định nghĩa và quản lý. Chúng ta cũng đã biết cách kết nối những trường có liên quan trong từng cấu trúc với những hàm được định nghĩa và cách cài đặt những cấu trúc đó vào kernel.

Để tìm hiểu hơn về nguyên tắc giao tiếp giữa driver và user, trong bài sau chúng ta sẽ bàn về các giao diện chuẩn trong cấu trúc lệnh của character device driver, đó là các hàm `read`(), `write`() và `ioctl`();

cuu duong than cong . com

cuu duong than cong . com

CuuDuongThanCong.com https://fb.com/tailieudientucntt

BÀI 4

CÁC GIAO DIỆN HÀM TRONG DRIVER

I. Tổng quan về giao diện trong cấu trúc lệnh file_operations:

Cấu trúc lệnh *file_operations* là một trong 3 cấu trúc quan trọng của *character device driver*, bao gồm một số những hàm chuẩn giúp giao tiếp giữa chương trình ứng dụng trong *user* và *driver* trong kernel. Chương trình ứng dụng chứa những yêu cầu thực thi từ người dùng chạy trong môi trường *user(user space)*. *Driver* chứa những hàm chức năng điều khiển thiết bị vật lý được lập trình sẵn chạy trong môi trường *kernel(kernel spcae)*. Để những yêu cầu từ *user* được nhận ra và điều khiển được phần thiết bị vật lý thì cần phải có một giao diện điều khiển. Những giao diện điều khiển này thực chất là những hàm `read()`, `write()`, và `ioctl()`, ... được linux định nghĩa sẵn, bản thân nó không có chức năng cụ thể, chức năng cụ thể được định nghĩa bởi người lập trình *driver*. Chẳng hạn, hàm `read()` có chức năng tổng quát là đọc thông tin từ driver đến một vùng nhớ đệm trong user (của một chương trình ứng dụng đang chạy), nhưng *driver* muốn lấy được thông tin cung cấp cho *user* thì phải qua nhiều thao tác điều khiển các thanh ghi lệnh thanh ghi dữ liệu của thiết bị vật lý được viết bởi người lập trình *driver,* dữ liệu thu được không thể truyền qua *user* một cách trực tiếp mà phải thông qua các hàm hỗ trợ trong giao diện khác như `copy_to_user()`.

Trong phần này, chúng ta sẽ tìm hiểu nguyên lý hoạt động của những giao diện này, làm cơ sở để viết hoàn chỉnh một *character driver* trong những bài sau.

II. Giao diện `read()` & `write()`:

Do hàm `read()` và `write()` nói chung đều có cùng một nhiệm vụ là trao đổi thông tin qua lại giữa user (*application*) và kernel (*driver*). Nếu hàm `read()` dùng để chép dữ liệu từ *driver* qua *applcation*, thì ngược lại hàm `write()` dùng để chép dữ liệu từ *application* sang *driver*. Hơn nữa hai hàm này đều có những tham số tương tự nhau. Vì thế chúng ta sẽ nghiên cứu hai hàm này cùng lúc.

1. Cấu trúc lệnh:

CuuDuongThanCong.com https://fb.com/tailieudientucntt

```
ssize_t read(struct file *filp, char __user *buff, size_t
count, loff_t *offp);
ssize_t write(struct file *filp, const char __user *buff,
size_t count, loff_t *offp);
```

2. Giải thích:

Trong cả hai hàm trên, `filp` là con trỏ tập tin thiết bị muốn truy xuất dữ liệu; `count` là kích thước muốn truy xuất tính bằng đơn vị byte. `buff` là con trỏ đến ô nhớ khai báo trong *application* để lưu dữ liệu đọc về trong trường hợp là hàm `read()`, trong trường hợp là hàm `write()` thì chính là con trỏ đến vùng nhớ rỗng cần ghi dữ liệu đến. Cuối cùng `offp` là con trỏ dạng *long offset* lưu vị trí con trỏ hiện tại của tập tin đang được truy cập. Sau đây chúng ta sẽ tìm hiểu kỹ hơn từng tham số trong hàm.

Con trỏ `const char __user *buff` trong hàm `read(...)` và `write(...)` đều là những con trỏ trỏ đến một vùng nhớ trong user space do người lập trình khai báo để lưu dữ liệu truy xuất từ *driver* thiết bị. Hơn nữa *driver* hoạt động trong môi trường *kernel*. Giữa *user space* và *kernel space* có những nguyên tắc quản lý bộ nhớ khác nhau. Do đó, *driver*, hoạt động trong môi trường *kernel*, không thể giao tiếp với con trỏ này, hoạt động trong môi trường *user*, một cách trực tiếp mà phải thông qua một số hàm đặc biệt.

Những hàm giao tiếp này được linux định nghĩa trong thư viện `<asm/uaccess.h>`. Các hàm này là:

- `unsigned long copy_to_user (void __user *to, const void *from, unsigned long count);` Công dụng của hàm là chép dữ liệu từ *kernel space* sang *user space*; Trong đó, `void __user *to` là con trỏ trỏ đến vùng nhớ trong *user space* lưu dữ liệu chép từ *kernel space*; `const void *from` là con trỏ chứa dữ liệu trong *kernel space* muốn chép qua *user space*; `unsigned long count` là số bytes dữ liệu muốn chép.
- `unsigned long copy_from_user (void *to, const void __user *from, unsigned long count);` Công dụng của hàm là chép dữ liệu từ *user space* sang *kernel space*; Trong đó, `void *to` là con trỏ trỏ đến vùng nhớ trong *kernel space*

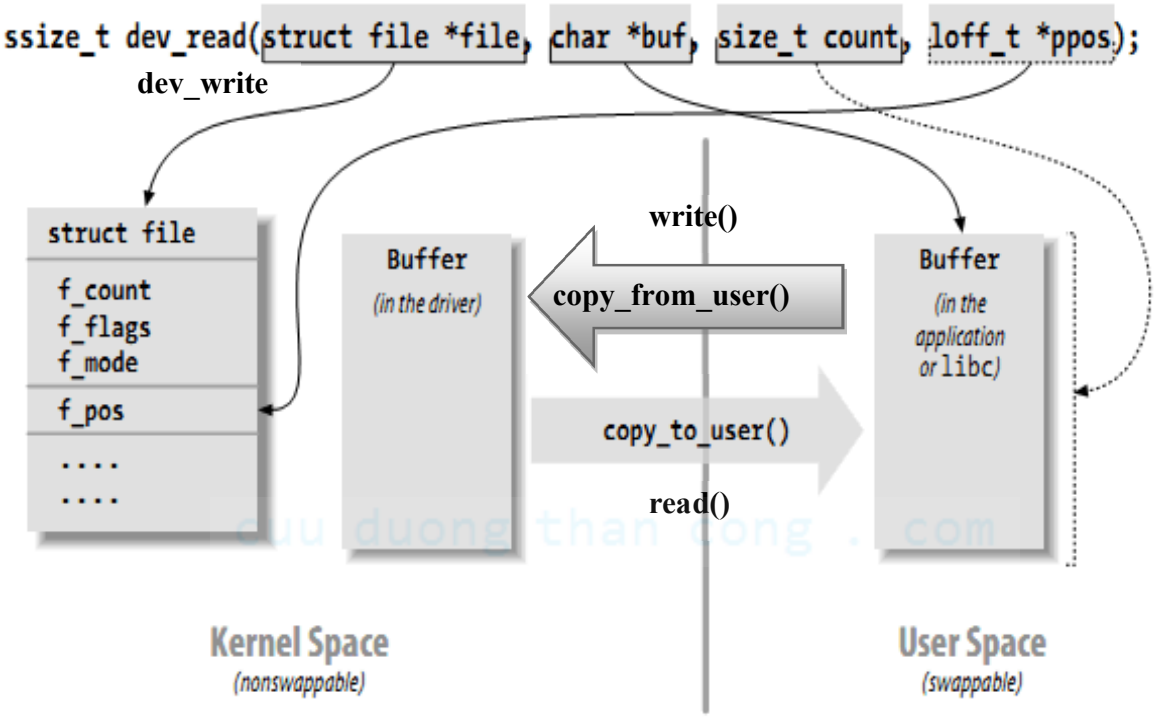
CuuDuongThanCong.com https://fb.com/tailieudientucntt

lưu dữ liệu chép từ *user space*; `const void __user *from` là con trỏ trỏ chứa dữ liệu trong *user space* muốn chép qua *kernel space*; `unsigned long count` là số bytes dữ liệu muốn chép.

Hai hàm đều có giá trị trả về kiểu `unsigned long`. Trước khi truyền dữ liệu chúng kiểm tra giá trị con trỏ vùng nhớ có hợp lệ hay không. Nếu không hợp lệ, thì quá trình truyền dữ liệu không thể thực hiện và hàm sẽ trả về mã lỗi `-EFAULT`. Nếu trong quá trình truyền dữ liệu, giá trị con trỏ cập nhật bị lỗi, thì quá trình truyền dừng ngay tại thời điểm đó, và như thế chỉ một phần dữ liệu được chép. Phần dữ liệu truyền thành công sẽ được thông báo trong giá trị trả về dưới dạng số byte. Chúng ta căn cứ vào giá trị này để truyền lại thông tin nếu cần thiết. Nếu không có lỗi xảy ra, hàm sẽ trả về giá trị 0.

Chúng ta sẽ minh họa nguyên tắc hoạt động của hàm `read()` và `write()` thông qua sơ đồ hình 3-3:

cuu duong than cong . com



ệm
gọi
ực
ay

`copy_from_user()` để giao tiếp trao đổi thông tin giữa hai vùng đệm này. Đồng

CuuDuongThanCong.com https://fb.com/tailieudientucntt

thời cập nhật những thông tin cần thiết phục vụ theo dõi quá trình truyền dữ liệu. Công việc này được thực hiện bởi người lập trình driver.

III. Giao diện ioctl():

Trên đây chúng ta đã tìm hiểu hai giao diện `read()` và `write()` dùng trao đổi dữ liệu qua lại giữa *user space* và *kernel space*, hay nói đúng hơn là giữa *driver* thiết bị vật lý và chương trình ứng dụng. Thế nhưng trong thực tế, ngoài việc trao đổi dữ liệu, một *driver* thiết bị còn phải thực hiện nhiều công việc khác ví dụ: cập nhật các thông số giao tiếp (tốc độ *baund*, số bits dữ liệu truyền nhận, ...) của thiết bị vật lý, điều khiển trực tiếp các thanh ghi lệnh phần cứng, đóng mở cổng giao tiếp, ... Linux cung cấp cho một giao diện chuẩn khác phục vụ cho tất cả các thao tác điều khiển trên, đó là giao diện `ioctl()`.

Như vậy, `ioclt()` có thể thực hiện những chức năng của hàm `read()` và `write()`, thế nhưng chỉ hiệu quả trong trường hợp số lượng dữ liệu cần truyền không cao, mang tính chất rời rạc. Trong trường hợp dữ liệu truyền theo từng khối, liên tục thì sử dụng hàm `read()` và `write()` là một lựa chọn khôn ngoan.

1. Cấu trúc lệnh:

Hàm `ioctl()` có cấu trúc như sau:

```
#include <linux/ioctl.h>
int (*ioctl) (struct inode *inode, struct file *filp, unsigned int cmd, unsigned long arg);
```

2. Giải thích:

Như trên đã phân tích, tất cả những giao diện trong *character device driver* đều không có những chức năng cụ thể. Những tham số lệnh chỉ mang tính tổng quát, người lập trình *driver* phải hiểu rõ ý nghĩa từng tham số sau đó sẽ định nghĩa cụ thể chức năng từng tham số này để đưa vào sử dụng tùy theo yêu cầu trong thực tế. Sau đây chúng ta sẽ tìm hiểu các tham số trong hàm ioctl.

- `struct inode *inode` và `struct file *filp` là con trỏ inode structure và *file structure* tương ứng với từng thiết bị được mở trong chương trình. Hai tham số này đều chứa trong số mô tả tập tin (*file descriptor*) gọi tắt là *fd*, được trả về khi thực hiện mở

CuuDuongThanCong.com https://fb.com/tailieudientucntt

thiết bị thành công bởi hàm `open()`. *(**Theo nguyên tắc, trước khi thực hiện giao tiếp với các thiết bị trong hệ thống linux, chúng ta phải truy cập đến cấu trúc inode của tập tin thiết bị trên bộ nhớ, kích hoạt khởi động thiết bị bằng hàm `open()`, hàm này sẽ quy định chế độ truy xuất tập tin thiết bị này, sau khi kích hoạt thành công hàm `open()` sẽ trả về số mô tả tập tin fd, mang tất cả thông tin của thiết bị đang được mở trong hệ thống).*

- `unsigned int cmd`, là số unsigned int do người lập trình driver quy định. Mỗi lệnh trong ioctl được phân biệt nhau thông qua số cmd này.

Số `unsigned int cmd` có 16 bits được chia thành 2 phần. Phần đầu tiên có 8 bits MSBs được gọi là số "*magic*" dùng phân biệt lệnh của thiết bị này với thiết bị khác. Phần thứ hai có 8 bit LSBs dùng để phân biệt những lệnh khác nhau của cùng một thiết bị. Số định danh lệnh này tương tự như số định danh thiết bị, phải được định nghĩa duy nhất trong hệ thống linux. Để biết được số định danh lệnh nào còn trống, chúng ta sẽ tìm trong tập tin *Documentation/ioctl-number.txt*. Tập tin này chứa thông tin về cách sử dụng hàm `ioctl()` và thông tin về số định danh lệnh đã được sử dụng trong hệ thống linux. Nhiệm vụ của chúng ta là đối chiếu so sánh để loại trừ những số định danh này, tìm ra số định danh trống cho thiết bị của mình. Sau khi tìm ra khoảng số định danh lệnh cho mình, chúng ta nên ghi rõ khoảng định danh đó vào tập tin *ioctl-number.txt* để sau này thuận tiện cho viêc lập trình driver mới.

Do hàm `ioctl()` tồn tại trong cả hai môi trường, giao diện trong *user space* và những thao tác lệnh hoạt động trong *kernel space* nên các số định danh lệnh này phải quy định chung trong cả hai môi trường. Đối với các chương trình *application* và *driver* có sử dụng giao diện `ioctl`, việc đầu tiên trong đoạn mã chương trình là định nghĩa số định danh lệnh, mỗi số định danh lệnh có một chế độ truy xuất dữ liệu khác nhau. Những chế độ này được giải thích trong tập tin tài liệu *Documentation/ioctl-number.txt*.

Số định danh thiết bị được định nghĩa theo dạng thức sau:

*/*Định nghĩa số magic cho số định danh lệnh là 'B' trong bảng mã ascii là 66 thập phân*/*

CuuDuongThanCong.com https://fb.com/tailieudientucntt

*/*Phần định nghĩa số định danh thiết bị được đặt đầu mỗi chương trình hoặc bên trong một <tập tin.h> thư viện*/*

```
#define IOC_GPIODEV_MAGIC  'B'
```

*/*Định nghĩa lệnh thứ nhất tên GPIO_GET, với mã số lệnh trong thiết bị là 10, chế độ truy xuất là _IO*/*

```
#define GPIO_GET        _IO(IOC_GPIODEV_MAGIC, 10)
#define GPIO_SET        _IO(IOC_GPIODEV_MAGIC, 11)
#define GPIO_CLEAR      _IO(IOC_GPIODEV_MAGIC, 12)
#define GPIO_DIR_IN     _IO(IOC_GPIODEV_MAGIC, 13)
#define GPIO_DIR_OUT    _IO(IOC_GPIODEV_MAGIC, 14)
```

Cấu trúc định nghĩa số định danh lệnh có dạng:

`<chế độ truy xuất lệnh>(<số magic thiết bị>, <số mã lệnh thiết bị>)`

- `<chế độ truy xuất lệnh>`: `ioctl` có 4 trạng thái lệnh: (Quy định bởi 2 bits trạng thái)

cuu duong than cong . com

`_IO` lệnh `ioctl` không có tham số;

`_IOW` lệnh `ioctl` chứa tham số dùng để nhận dữ liệu từ *user* (tương dương chức năng của hàm `copy_from_user()`);

`_ IOR` lệnh `ioctl` chứa tham số dùng để gửi dữ liệu sang *user* (tương đương chức năng của hàm `copy_to_user()`);

`_IOWR` lệnh `ioctl` chứa tham số dùng cho hai nhiệm vụ, đọc và ghi dữ liệu của *user*.

- `<số magic thiết bị>`: Là số đặc trưng cho từng thiết bị do người dùng định nghĩa.
- `<số mã lệnh thiết bị>`: Là số đặc trưng cho từng lệnh trong mỗi thiết bị.

cuu duong than cong . com

Căn cứ vào số định danh lệnh này, người lập trình *driver* sẽ lựa chọn lệnh nào sẽ được thực thi khi *user* gọi hàm `ioctl`. Công việc lựa chọn lệnh được thực hiện bằng cấu trúc `switch...case...` như sau:

Đây là đoạn chương trình mẫu cho hàm `ioctl()` được khai báo sử dụng trong driver `gpio_dev` mang tên `gpio_ioctl`;

CuuDuongThanCong.com https://fb.com/tailieudientucntt

```
   static int
   gpio_ioctl(struct inode * inode, struct file * file, unsigned
int cmd, unsigned long arg)
   {
      int retval = 0;
   /*Thực hiện lựa chọn lệnh thực thi bằng lệnh switch với tham số  lựa chọn là số định
   danh lệnh cmd*/
      switch (cmd)
      {
      case GPIO_GET:
      /*Hàm thao tác cho lệnh GPIO_GET*/
         break;
      case GPIO_SET:
      /*Hàm thao tác cho lệnh GPIO_SET*/
         break;
      case GPIO_CLEAR:
      /*Hàm thao tác cho lệnh GPIO_CLEAR*/
         break;
      case GPIO_DIR_IN:
      /*Hàm thao tác cho lệnh GPIO_DIR_IN*/
         break;
      case GPIO_DIR_OUT:
       /*Hàm thao tác cho lệnh GPIO_DIR_OUT*/
         break;
      default:
      /*Trả về mã lỗi trong trường hợp không có lệnh thực thi phù  hợp*/
         retval = -EINVAL;
         break;
      }
      return retval;
   }
```


CuuDuongThanCong.com https://fb.com/tailieudientucntt

- `unsigned long arg`, là tham số lệnh tương ứng với số định danh lệnh, tham số này có vai trò là bộ nhớ đệm chứa thông tin từ user hay thông tin đến user tùy theo chế độ của lệnh được định nghĩa ở đầu chương trình driver. Trong trường hợp lệnh không có tham số, chúng ta không cần thiết phải sử dụng tham số này trong user, chương trình trong ioctl vẫn thực thi mà không có lỗi. Trong trường hợp hàm cần nhiều tham số, chúng ta khai báo sử dụng tham số này như là một con trỏ dữ liệu.

IV. Tổng kết:

Như vậy với những giao diện chính trong character device như `read()`, `write()` và `ioctl()`, `open()`, ... chúng ta có thể giao tiếp giữa *user* và *kernel* rất thuận lợi. Điều này cũng có nghĩa rằng: Giữa người sử dụng và thiết bị vật lý có thể giao tiếp trao đổi thông tin với nhau dễ dàng thông qua hoạt động của driver với sự hỗ trợ của giao diện. Bên cạnh những giao diện trên, *linux* còn hỗ trợ rất nhiều giao diện khác, ứng dụng của những giao diện này không nằm trong các driver được nghiên cứu trong giáo trình cho nên chúng sẽ không được đề cập đến. Đối với những driver lớn, sẽ có rất nhiều giao diện khác nhau xuất hiện, nhiệm vụ của chúng ta là tìm hiểu nguyên lý hoạt động của giao diện đó dựa vào những kiến thức đã học. Đây cũng chính là mục đích cuối cùng mà nhóm biên tập muốn thực hiện trong cuốn giáo trình này.

Trước khi bước vào viết một driver hoàn chỉnh, chúng ta sẽ tìm hiểu các cấu trúc để viết một driver đơn giản trong bài sau.

cuu duong than cong . com

CuuDuongThanCong.com https://fb.com/tailieudientucntt

BÀI 5

TRÌNH TỰ VIẾT CHARACTER DEVICE DRIVER

Lập trình driver là một trong những công việc quan trọng mà một người lập trình hệ thống nhúng cần phải nắm vững. Để một chương trình ứng dụng hoạt động tối ưu, người lập trình phải biết phân công nhiệm vụ giữa *driver* và *application* sao cho hợp lý. Trong một ứng dụng điều khiển, nếu *driver* đảm trách nhiều chức năng thì chương trình trong *application* sẽ trở nên đơn giản, nhưng bù lại tài nguyên phần cứng sẽ dễ bị xung đột, không thuận lợi cho các *driver* khác dùng chung tài nguyên. Ngược lại nếu *driver* chỉ thực hiện những công việc rất đơn giản, thì chương trình trong *application* trở nên phức tạp, việc chuyển qua lại giữa các tiến trình trở nên khó khăn. Việc phân chia nhiệm vụ này đòi hỏi phải có nhiều kinh nghiệm lập trình thực tế với hệ thống nhúng mới có thể đem lại hiệu quả tối ưu cho hệ thống.

Với những kiến thức tổng quát đã trình bày trong những phần trước về *driver* mà chủ yếu là *character device driver*, chúng ta đã có những khái niệm ban đầu về vai trò của *driver* trong hệ thống nhúng, cách thức điều khiển thiết bị vật lý, các giao diện giao tiếp thông tin giữa *user space* và *kernel space*, ... trong phần này chúng ta sẽ đi vào các bước cụ thể để viết hoàn chỉnh một *character device driver*.

Công việc phát triển một ứng dụng điều khiển mới thường tiến hành theo những bước sau:

- Tìm hiểu nhu cầu điều khiển ngoài thực tế: Từ hoạt động thực tiễn hay đơn đặt hàng xác định yêu cầu, thời gian thực hiện, giá cả chi phí, ...
- Phân tích yêu cầu điều khiển: Từ những yêu cầu giới hạn về thời gian, chi phí, chất lượng, người lập trình tìm ra phương pháp tối ưu hóa hệ thống, lựa chọn công nghệ phù hợp, ...; Lập danh sách các yêu cầu cần thực hiện trong hệ thống điều khiển;
- Phân công nhiệm vụ điều khiển giữa *application* và *driver* để hệ thống hoạt động tối ưu;

CuuDuongThanCong.com https://fb.com/tailieudientucntt

- Lập trình *driver* theo những yêu cầu được lập;
- Lập trình *application* sử dụng *driver* hoàn tất chương trình điều khiển;
- Chạy kiểm tra độ tin cậy hệ thống;
- Giao cho khách hàng sử dụng, bào trì sửa chữa khắc phục lỗi khi thực thi;

Chúng ta đang tập trung nghiên cứu bước lập trình *driver*, khi đã có những yêu cầu cụ thể. Trong bước này, có rất nhiều cách thực hiện, tuy nhiên nhìn chung đều có những thao tác căn bản sau:

1. Viết lưu đồ hoặc máy trạng thái thực thi cho dirver;
2. Lập trình mã lệnh;
3. Biên dịch *driver*;
4. Cài đặt vào hệ thống linux;
5. Viết chương trình ứng dụng kiểm tra hoạt động driver;
6. Nếu đạt yêu cầu gán cố định vào cấu trúc *kernel* linux, biên dịch lại nhân để *driver* hoạt động lâu dài trong hệ thống; Nếu không tiến hành chỉnh sửa, kiểm tra cho đến khi hoàn thiện;

Trong đó các bước 1, 5, 6 chúng ta đã có dịp nghiên cứu trong phần I lập trình nhúng căn bản hoặc trong các tài liệu chuyên ngành khác. Phần này chúng ta sẽ tìm hiểu chi tiết các bước 2, 3, và 4;

I. Lập trình mã lệnh trong *character driver*:

Có rất nhiều cách viết *character driver* nhưng cũng giống như chương trình ứng dụng *application*, *character driver* cũng có những cấu trúc chung, chuẩn, từ đó sẽ tùy biến theo từng yêu cầu cụ thể. Ở đây, chúng ta sẽ tìm hiểu hai dạng cấu trúc: Cấu trúc dạng 1, và cấu trúc dạng 2;

Cấu trúc dạng 1: Bao gồm những thao tác cơ bản nhất, nhưng đa số những thao tác kiểm tra lỗi, lấy số định danh lệnh, số định danh thiết bị, ... người lập trình driver phải tự mình thực hiện. Có một ưu điểm là người lập trình có thể tùy biến theo yêu cầu của ứng dụng, có khả năng phát triển driver. Nhưng bù lại, thời gian thực hiện hoàn chỉnh *driver* để có thể chạy trong linux lâu hơn.

CuuDuongThanCong.com https://fb.com/tailieudientucntt

Cấu trúc dạng 2: Những thao tác cơ bản trong việc thành lập các thông số cho driver như: Lấy số định danh lệnh, thiết bị, khai báo cài đặt driver vào hệ thống, ... được gói gọn trong một câu lệnh duy nhất. Ưu điểm là thời gian thực hiện hoàn chỉnh một *driver* nhanh hơn, đơn giản hơn, các dịch vụ kiểm tra lỗi được hỗ trợ sẵn, giúp tránh xảy ra lỗi trong quá trình sử dụng. Thế nhưng việc tùy biến của người dùng nằm trong một giới hạn cho phép.

Thông thường chúng ta nên dùng cấu trúc dạng 2 cho những *driver* đơn giản, và những khả năng trong cấu trúc này hầu hết cũng tương tự như trong cấu trúc dạng 1, cũng đủ cho chúng ta thực hiện tất cả những thao tác điều khiển khác nhau.

Sau đây chúng ta sẽ tìm hiểu chi tiết từng cấu trúc để có cái nhìn cụ thể hơn về những ưu và nhược điểm nêu trên. Các chương trình *driver* cũng có cấu trúc tương tự như các chương trình trong C, chỉ khác là chúng có thêm những hàm chuẩn định nghĩa riêng cho việc giao tiếp điều khiển giữa *user space* và *kernel space* và còn nhiều đặc điểm khác nữa, sẽ được chúng ta đề cập trong mỗi phần cấu trúc;

cuu duong than cong . com

1. Cấu trúc chương trình character device driver dạng 1:

*/*Bước 1: Khai báo thư viện cho các hàm dùng trong chương trình, thư viện dùng trong kernel space khác với thư viện dùng trong user space. Thư viện trong driver là những hàm, biến, hằng số, ... được định nghĩa sẵn và hầu hết được lưu trong thư mục linux/ trong cấu trúc mã nguồn của linux. Do đó khi khai báo, chúng ta phải chỉ rõ đường dẫn đến thư mục này */*

```
#include <linux/module.h> //Thư viện thứ nhất trong thư mục linux
#include <linux/fs.h>
#include <linux/cdev.h>
#include <asm/uaccess.h>
......
```

cuu duong than cong . com

*/*Bước 2: Định nghĩa những hằng số cần dùng trong chương trình driver*/*

```
#define ...
...
```


CuuDuongThanCong.com https://fb.com/tailieudientucntt

```
/*Bước 3: Khai báo số định danh thiết bị (Device number); Cấu trúc (file structure); Cấu trúc (file operations); */
int                 device_devno; //Khai báo biến lưu số định danh thiết bị
struct cdev         device_cdev; //Khai báo biến lưu cấu trúc tập tin
struct file_operations      device_fops; //Khai báo biến lưu cấu trúc lệnh
/*Bước 4: Khai báo những biến, cấu trúc cần dùng trong chương trình driver*/
unsigned int variable_0;     //Những biến này được định nghĩa tương tự trong
unsigned long variable_1;//mã lệnh C, và C++
struct clk *device_clock_0;//
....
/*Bước 5: Khai báo, định nghĩa những hàm con, chương trình con được sử dụng trong driver*/
   /*Chương trình con thứ nhất, có hai tham số parameter_0 và parameter_1 cùng kiểu int*/
void sub_program_0 (int parameter_0, int parameter_1) {
   /*Những lệnh thao tác cho chương trình con*/
   lệnh_0;
   lệnh_1;
   lệnh_2;
}
   /*Hàm thứ nhất, trả về kiểu int, hai tham số con trỏ parameter_0 và parameter_1 kiểu int*/
int func_0 (int *parameter_0, int *parameter_1) {
   /*Các thao tác lệnh cho hàm*/
   lệnh_0;
   lệnh_1;
   lệnh_2;
```


CuuDuongThanCong.com https://fb.com/tailieudientucntt

```
}
```
*/*Bước 6: Định nghĩa hàm init cho driver, hàm init là hàm được thực thi đầu tiên khi thực hiện cài đặt driver vào hệ thống linux bằng lệnh shell insmod driver_name.ko*/*

*/*Hàm init có một vai trò quan trọng trong lập trình driver. Ban đầu hàm sẽ gọi thực thi các hàm xác định số định danh thiết bị; Cập nhật các thông số của cấu trúc tập tin, cấu trúc lệnh; Đăng ký các cấu trúc vào hệ thống linux; Khởi tạo các thông số điều khiển ban đầu cho các thiết bị ngoại vi mà driver điều khiển*/*

*/*Hàm init có kiểu dữ liệu trả về dạng int, static trong trường hợp này nghĩa là hàm init chỉ dùng riêng cho driver sở hữu*/*

```
static int __init at91adc_init (void) {
   /*Khai báo biến lưu giá trị trả về của hàm*/
   int ret;
   /*Xác định số định danh động cho thiết bị, tránh trường hợp bị trùng khi cài đặt với các driver thiết bị khác đã có trong hệ thống*/
   //Chỉ đến địa chỉ biến lưu số định danh đầu tiên tìm được;
   ret = alloc_chrdev_region ( &device_devno
                               0,    //Số Minor đầu tiên yêu cầu;
                               1,    //Số thiết bị yêu cầu;
                               "device_name" );//Tên thiết bị muốn đăng ký;
   /*Kiểm tra mã lỗi khi thực thi hàm alloc_chrdev_region()*/
   if (ret < 0) {
         //In thông báo lỗi khi thực hiện xác định số định danh thiết bị;
         printk (KERN_INFO  "Device_name: Device number
         allocate fail");
         ret = -ENODEV; //Trả về mã lỗi cho biến ret;
         return ret; //Trả về mã lỗi cho hàm init;
   }
   /*Khởi tạo thiết bị với số định danh đã xác định, yêu cầu hệ thống dành một vùng nhớ lưu driver thiết bị sắp được tạo ra;*/
```


CuuDuongThanCong.com https://fb.com/tailieudientucntt

```
    cdev_init ( &device_cdev,       //Trỏ đến cấu trúc tập tin đã được định
    nghĩa;
              device_devno,   //Số định danh thiết bị đã được xác định;
              1 );            //Số thiết bị muốn đăng ký vào hệ thống;
    /*Chập nhật những thông tin cần thiết cho cấu trúc tập tin vừa được hệ thống
    dành ra*/
    device_cdev.owner = THIS_MODULE; /*Cập nhật người sở hữu cấu trúc
    tập tin*/
    device_cdev.ops = &device_fops; /*Cập nhật cấu trúc lệnh đã được
    định nghĩa*/
    /*Đăng ký thiết bị vào hệ thống */
    ret = cdev_add ( &device_cdev, //Trỏ đến cấu trúc tập tin đã được khởi
    tạo;
                  device_devno, //Số định danh thiết bị đã được xác định;
                  1 ); //Số thiết bị muốn đăng ký;
    /*Kiểm tra lỗi trong quá trình đăng ký thiết bị vào hệ thống*/
    if (ret < 0)
          {
          //In thông báo khi lỗi xuất hiện
          printk(KERN_INFO "Device: Device number allocation
          failed\n");
          ret = -ENODEV; //Trả về mã lỗi cho biến;
          return ret;//Trả về mã lỗi cho hàm;
          }
    /*Thực hiện các công việc khởi tạo thiết bị vật lý do driver điều khiển*/
    ...
    //Trả về mã lỗi 0 khi quá trình thực thi hàm init không có lỗi;
    return 0;
}
```


CuuDuongThanCong.com https://fb.com/tailieudientucntt

*/*Bước 7: Khai báo và định nghĩa hàm exit, hàm exit là hàm được thực hiện ngay trước khi driver được tháo gỡ khỏi hệ thống bằng lệnh shell rmmod device.ko; Những tác vụ bên trong hàm exit được lập trình nhằm khôi phục lại trạng thái hệ thống trước khi cài đặt driver. Chẳng hạn như giải phóng vùng nhớ, timer, vô hiệu hóa các nguồn phát sinh ngắt, ...*/*

```
static void __exit device_exit (void) {
   /*Các lệnh giải phóng vùng nhớ sử dụng trong driver*/
   ...
   /*Các lệnh giải phóng nguồn ngắt, timer, ...*/
   ...
   /*Tháo thiết bị ra khỏi hệ thống bằng hàm*/
   unregister_chrdev_region(     device_devno, /*Số định danh đầu tiên nhóm thiết bị;*/
                                 1); //Số thiết bị cần tháo gỡ;
   /*In thông báo driver thiết bị đã được tháo ra khỏi hệ thống*/
   printk(KERN_INFO "device: Unloaded module\n");
}
```

*/*Bước 8: Khai báo hàm open, đây là hàm được thực thi ngay sau khi lệnh open trong user space thực thi. Những lệnh chứa trong hàm này thông thường dùng cập nhật dữ liệu ban đầu cho chương trình driver, khởi tạo môi trường hoạt động ban đầu để hệ thống làm việc ổn định*/*

```
static int device_open (struct inode *inode, struct file *filp)
{
   /*Nơi lập trình các lệnh khởi tạo hệ thống*/
   return 0;
}
```

*/*Bước 9: Khai báo và định nghĩa hàm release, đây là hàm được thực thi ngay trước khi driver bị đóng bởi hàm close trong user space*/*

CuuDuongThanCong.com https://fb.com/tailieudientucntt

```
static int device_release (struct inode *inode, struct file
*filp)
{
   /*Nơi thực thi những lệnh trước khi driver bị đóng, không còn sử dụng*/
   return 0;
}
```

*/*Bước 10: Khai báo các hàm read(), write(), ioctl() trong hệ thống khi cần sử dụng*/*

*/*Hàm read() đọc thông tin từ driver qua chương trình trong user*/*

```
static ssize_t device_read (struct file *filp, char __iomem
*buf, size_t bufsize, loff_t *f_pos) {
```

*/*Định nghĩa các tác vụ hoạt động trong hàm read, để truy cập lấy thông tin từ thiết bị vật lý*/*

```
...
}
```

cuu duong than cong . com

*/*Hàm write() ghi thông tin từ user qua driver*/*

```
static ssize_t device_write (struct file *filp, char __iomem
*buf, size_t bufsize, loff_t *f_pos) {
```

*/*Định nghĩa các tác vụ hoạt động trong hàm write(), để truy cập lấy thông tin từ chương trình ứng dụng trong user*/*

```
...
}
```

*/*Hàm ioctl định nghĩa các trạng thái lệnh thực thi theo số định danh lệnh từ chương trình ứng dụng bên user*/*

```
static int
gpio_ioctl (struct inode * inode, struct file * file, unsigned
int cmd, unsigned long arg) {
```

cuu duong than cong . com

*/*Khai báo biến lưu mã lỗi trả về cho hàm giao diện ioctl()*/*

```
   int retval = 0;
```

*/*Chia trường hợp thực thi lệnh*/*

CuuDuongThanCong.com https://fb.com/tailieudientucntt

```
    switch (cmd) {
         case LENH_0:
    /*Lệnh 0 được thực thi*/
        break;
         case LENH_1:
    /*Lệnh 1 được thực thi*/
        break;
         default:
    /*Có thể thông báo, không có lệnh nào để thực thi*/
        retval = -EINVAL;
        break;
         }
         return retval;
}
```

cuu duong than cong . com

```
/*Bước 11: Gán các con trỏ hàm giao diện chuẩn vào cấu trúc lệnh file structure*/
struct file_operations gpio_fops = {
        .read           = device_read,
        .write    = device_write,
        .ioctl    = device_ioctl,
        .open           = device_open,
        .release  = device_release,
};
```

cuu duong than cong . com

```
/*Bước 12: Gán các hàm exit và init vào hệ thống driver*/
module_init (device_init); /*Hàm device_init() thực hiện khi driver được cài đặt*/
module_exit  (device_exit); /*Hàm device_exit() thực hiện khi driver được gỡ bỏ*/
/*Bước 13: Khai báo các thông tin liên quan đến driver*/
```


CuuDuongThanCong.com https://fb.com/tailieudientucntt

```
MODULE_AUTHOR("Tên người viết driver");
MODULE_DESCRIPTION("Mô tả nhiệm vụ của driver");
MODULE_LICENSE("GPL"); //Bản quyền của driver là GPL
```

2. *Cấu trúc chương trình character device driver dạng 2:*

Cấu trúc chương trình *character device driver* dạng 2 cũng tương tự như dạng 1, nhưng điểm khác biệt là kỹ thuật tạo và đăng ký thiết bị mới vào hệ thống. Để thuận tiện cho việc tham khảo chúng tôi không ngại nhắc lại những bước tương tự và giải thích những điều mới khi tiếp xúc với câu lệnh.

```
/*Bước 1: Khai báo những thư viện cần thiết cho các lệnh dùng trong chương trình driver*/
#include <linux/module.h>
#include <linux/errno.h>
#include <linux/init.h>
#include <asm/uaccess.h>
#include <asm/atomic.h>
#include <linux/miscdevice.h>
/*Bước 2: Định nghĩa những hằng số cần dùng trong chương trình driver*/
#define ...
...
/*Bước 3: Khai báo số định danh thiết bị, các biến phục vụ đồng bộ hóa dữ liệu*/
/*Biến lưu số major của thiết bị*/
static int dev_major;
/*Khai báo biến atomic_t dùng đồng bộ hóa tài nguyên driver thiết bị*/
/*Cách đồng bộ hóa tài nguyên của thiết bị:
Trước khi mở thao tác với driver thiết bị, kỹ thuật atomic sẽ kiểm tra xem thiết bị đã được mở hay chưa, nếu thiết bị đã được mở (nhận biết bằng số counter) thì không cho phép truy cập đến thiết bị này, thông báo lỗi cho người sử dụng. Ngược lại, cho phép mở thiết bị, thực thi những câu lệnh tiếp theo trong hệ thống. Cụ thể là:
- Ban đầu cho biến kiểu atomic_t bằng 1, nghĩa là thiết bị chưa được mở;
```


CuuDuongThanCong.com https://fb.com/tailieudientucntt

- *Khi thiết bị được mở, chúng ta giảm biến kiểu atomic_t một đơn vị, lúc này giá trị biến kiểu atomic_t bằng 0. Trong quá trình mở, hay đóng đều phải có sự so sánh biến atomic_t.*
- *Chỉ đóng driver thiết bị khi biến kiểu atomic_t bằng 1. Chỉ mở khi biến kiểu atomic_t bằng 0.*
- *Đóng thiết bị, thực hiện tăng biến kiểu atomic_t. Mở thiết bị, thực hiện giảm biến kiểu atomic_t.*/*

```
static atomic_t device_open_cnt = ATOMIC_INIT(1);
```

*/*Bước 4: Khai báo các biến dùng trong chương trình driver*/*

```
...
```

*/*Bước 5: Khai báo và định nghĩa các hàm, chương trình con cần sử dụng trong quá trình lập trình driver*/*

```
...
```

*/*Bước 6: Khai báo và định nghĩa hàm open, là hàm được thực thi khi thực hiện lệnh open() trong user application*/*

```
static int device_open (struct inode *inode, struct file
*file) {
   /*Khai báo biến lưu mã lỗi trả về*/
   int result = 0;
   /*Khai báo biến lưu số minor của thiết bị, đồng thời cập nhật số minor của thiết bị
   trong hệ thống*/
   unsigned int device_minor = MINOR (inode->i_rdev);
   /*Thực hiện kiểm tra biến atomic_t trong quá trình mở thiết bị nhiều lần*/
   if (!atomic_dec_and_test(&device_open_cnt)) {
      /*Tăng biến kiểu atomic_t*/
      tomic_inc(&device_open_cnt);
      /*In ra thông báo thiết bị muốn mở đang bận*/
      printk(KERN_ERR DRVNAME ": Device with minor ID %d
      already in use\n", dev_minor);
      /*Trả về mã lỗi bận*/
```


CuuDuongThanCong.com https://fb.com/tailieudientucntt

```
        return -EBUSY;
    }
/*Thực hiện những công việc tiếp theo nếu mở tập tin thiết bị thành công*/
...
/*Trả về mã lỗi cuối cùng nếu quá trình thực thi không bị lỗi*/
return result;
}
/*Bước 7: Khai báo và định nghĩa hàm close(), được thực hiện khi thực thi hàm close
trong user application*/
static int device_close (struct inode *inode, struct file
*file) {
    /*Thực hiện đồng bộ hóa counter trước khi tăng*/
    smp_mb_before_atomic_inc();
    /*Tăng biến đếm atomic khi đóng tập tin thiết bị*/
    atomic_inc(&device_open_cnt);
    /*Trả về mã lỗi 0 cho hàm close()*/
    return 0;
}
/*Bước 8: Khai báo và cập nhật cấu trúc lệnh file_operations cho thiết bị*/
struct file_operations device_fops = {
    /*Gán hàm device_read của thiết bị vào giao diện chuẩn, nếu sử dụng*/
    .read = device_read,
    /*Gán hàm device_write của thiết bị vào giao diện chuẩn, nếu sử dụng*/
    .write = device_write,
    /*Gán hàm device_open của thiết bị vào giao diện chuẩn open*/
    .open = device_open,
    /*Gán hàm device_release của thiết bị vào giao diện chuẩn release*/
    .release = device_release,
};
/*Bước 9: Khai báo và cập nhật cấu trúc tập tin thiết bị theo kiểu miscdevice*/
```


CuuDuongThanCong.com https://fb.com/tailieudientucntt

```
static struct misdevice device_dev = {
    /*Thực hiện kỹ thuật lấy số minor động cho thiết bị bằng hằng số định nghĩa sẵn
    trong thư viện linux/miscdevice.h*/
    .minor = MISC_DYNAMIC_MINOR,
    /*Cập nhật tên cho thiết bị*/
    .name = "device",
    /*Cập nhật cấu trúc lệnh đã được định nghĩa*/
    .fops = device_fops,
};
/*Bước 10: Khai báo định nghĩa hàm init thực hiện khi cài đặt driver vào hệ thống*/
/*Tương tự như trong dạng 1, tuy nhiên với kỹ thuật mới này, trong hàm init chúng ta
không cần phải lấy số định danh thiết bị, khởi tạo vùng nhớ cho driver, cài đặt driver
vào hệ thống. Tất cả những công việc này được thực hiện bởi một hàm duy nhất đó là
misc_register()*/
static int __init device_mod_init(void) {
    /*Những lệnh cần thực hiện khi cài đặt driver vào hệ thống*/
    ...
    /*Thực hiện đăng ký driver vào hệ thống, kết hợp trả về mã lỗi*/
    return    misc_register(&devive_dev);
}
/*Bước 11: Khai báo và định nghĩa hàm exit()*/
static void __exit device_mod_exit (void) {
    /*Thực hiện những công việc khôi phục hệ thống trước khi driver bị tháo gỡ*/
    ...
    /*Tháo gỡ thiết bị mang tên là device_dev*/
    misc_deregister (&device_dev);
}
/*Bước 12: Gắn các hàm exit và init vào hệ thống driver*/
module_init (device_mod_init);
module_exit (device_mod_exit);
```


CuuDuongThanCong.com https://fb.com/tailieudientucntt

*/*Bước 13: Cập nhật các thông tin cá nhân cho driver*/*

```
MODULE_LICENSE("Bản quyền driver thông thường là GPL");
MODULE_AUTHOR("Tên người viết driver");
MODULE_DESCRIPTION("Mô tả khái quát thông tin về driver");
```

II. Biên dịch driver:

Sau khi lập trình *dirver*, công việc tiếp theo là biên dịch *driver* biến tập tin mã nguồn C thành tập tin ngôn ngữ máy *driver* trước khi cài đặt vào hệ điều hành. Sau đây là các bước biên dịch driver.

Biên dịch driver có 2 dạng, đầu tiên biên dịch driver khi tập tin mã nguồn driver thuộc một bộ phận trong cấu trúc mã nguồn mở của *kernel*, khi đó *driver* được biên dịch cùng lúc với *kernel*, cách này chỉ áp dụng khi *driver* đã hoạt động ổn định, hơn nữa chúng ta không cần cài đặt lại *driver* khi khởi động lại hệ thống. Phương pháp thứ hai là biên dịch driver khi nó nằm ngoài cấu trúc mã nguồn mở của *kernel*, *driver* có thể được biên dịch trong khi *kernel* đang chạy, ưu điểm của phương pháp này là thời gian thực hiện nhanh, thích hợp cho việc thử nghiệm *driver* mới, thế nhưng mỗi lần hệ thống khởi động lại, driver sẽ bị mất do đó phải cài đặt lại khi khởi động. Khi *driver* đã hoạt động ổn định chúng ta mới biên dịch *driver* theo cách 1.

Biên dịch *driver* hoàn toàn khác với biên dịch chương trình ứng dụng. Chương trình ứng dụng có những thư viện chuẩn trong hệ thống *linux*, nên khi biên dịch ta chỉ việc chép tập tin chương trình vào trong một thư mục bất kỳ trong cấu trúc *root file system* và gọi lệnh biên dịch. Nhưng đối với *driver*, những thư viện sử dụng không nằm sẵn trong hệ thống, mà nằm trong cấu trúc mã nguồn mở của *kernel*. Vì thế trước khi biên dịch *driver* chúng ta phải giải nén tập tin mã nguồn mở của *kernel* vào cấu trúc *root file system*. Sau đó tạo tập tin `Makefile` để dùng lệnh `make` trong *shell* biên dịch *driver*. Cấu trúc Makefile đã được hướng dẫn kỹ trong phần lập trình hệ thống nhúng căn bản. Trong phần này chúng ta chỉ tạo ra `Makefile` với nội dung cần thiết để có thể biên dịch được *driver*.

Biên dịch *driver* được tiến hành theo các bước sau:

CuuDuongThanCong.com https://fb.com/tailieudientucntt

1. Chép tập tin *driver* mã nguồn C vào thư mục nào đó trong cấu trúc *root file system.*
2. Tạo tập tin có tên `Makefile` nằm trong cùng thư mục với tập tin *driver* mã nguồn C. Tập tin `Makefile` có nội dung như sau:

   Thông báo cho trình biên dịch biết loại chip mà driver sẽ cài đặt/

   ```
   export ARCH=arm
   ```

   */*Khai báo chương trình biên chéo là những tập tin có tên đầu tiên là arm-none-linux-gnueabi-...*/*

   ```
   export CROSS_COMPILE=arm-none-linux-gnueabi-
   ```

   */*Tại đây tập tin mã nguồn driver sẽ được biên dịch thành tập tin .ko, có thể cài đặt vào linux*/*

   ```
   obj-m += <tên tập tin driver mã nguồn C>.o
   ```

   */*Tùy chọn all, thực hiện chuyển đến cấu trúc mã nguồn mở của kernel, tại đây driver sẽ được biên dịch thông qua lệnh modules */*

   ```
   all:
   make -C <đường dẫn đến thư mục chứa cấu trúc mã nguồn mở của kernel> M=$(PWD) modules
   ```

   */*Tùy chọn clean, thực hiện chuyển đến cấu trúc mã nguồn mở của kernel, thực hiện xóa những tập tin .o, .ko được tạo thành trong lần biên dịch trước*/*

   ```
   clean:
   make -C / đường dẫn đến thư mục chứa cấu trúc mã nguồn mở của kernel> M=$(PWD) clean
   ```

3. Tại thư mục chứa mã nguồn *driver*, dùng lệnh *shell*: `make clean all`

   Lúc này hệ thống linux sẽ xóa những tập tin có đuôi .o, .ko, ... tạo thành trong những lần biên dịch trước.Tiếp theo, biên dịch tập tin mã nguồn driver thành tập tin .ko, tập tin này có thể được cài dặt vào hệ thống linux thông qua những thao tác sẽ được hướng dẫn trong phần sau.

III. Cài đặt driver vào hệ thống linux:

CuuDuongThanCong.com https://fb.com/tailieudientucntt

Trong phần I, chúng ta đã trình bày hai cấu trúc chung để viết hoàn chỉnh một *character device driver*, mỗi cấu trúc đều có những ưu và nhược điểm riêng. Những ưu và nhược điểm này còn được biểu hiện rõ trong việc cài đặt *driver* vào hệ thống.

Sau khi đã biên dịch thành công *driver*, xuất hiện tập tin .ko trong thư mục chứa tập tin mã nguồn, chúng ta dùng những câu lệnh trong *shell* để hoàn tất công đoạn cuối cùng đưa driver vào hoạt động trong hệ thống, tiến hành kiểm tra chức năng. Tùy theo kỹ thuật áp dụng lập trình *driver* mà sẽ có những thao tác cài đặt khác nhau:

1. ***Cài đặt driver khi áp dụng cấu trúc dạng 1:***

   Tiến hành theo các bước sau:

- Di chuyển đến thư mục chứa tập tin .ko vừa biên dịch xong;
- Tại dòng lệnh shell, thực thi: `insmod <tên driver>.ko`;
- Vào tập tin */proc/devices* tìm tên thiết bị vừa cài đặt vào hệ thống, xác định số *Major* và số *Minor* động của thiết bị;
- Sử dụng câu lệnh trong shell: `mknod /dev/<tên thiết bị> c <Số Major> <Số Minor>`, tạo inode trong thư mục */dev/*, làm tập tin thiết bị sử dụng cho những chương trình ứng dụng;

2. ***Cài đặt driver khi áp dụng cấu trúc dạng 2:***

- Di chuyển đến thư mục chứa tập tin .ko vùa biên dịch xong;
- Tại dòng lệnh *shell*, thực thi lệnh: `insmod <tên driver>.ko`;

  Khi đó, cấu trúc inode tự động được tạo ra trong thư mục */dev/* liên kết với số định danh thiết bị lưu trong tập tin */proc/devices* mà không cần phải thông qua những câu lệnh shell khác như trong cách 1. Như vậy, với cấu trúc dạng 2 thời gian cài đặt driver vào hệ thống được rút gọn đáng kể.

IV. Tổng kết:

Từ những kiến thức lý thuyết nền tảng trong những bài trước, chúng ta đã rút ra được những bước tổng quát để lập trình hoàn chỉnh một character *device driver*. Có nhiều cách để viết ra một *character driver*, trong bài này chỉ đề cập 2 cách căn bản làm nền tảng cho các bạn nghiên cứu thêm những cách khác hiệu quả hơn ngoài thực tế.

CuuDuongThanCong.com https://fb.com/tailieudientucntt

Trong bài sau, chúng ta sẽ thực hành viết một *character device driver* mang tên là *helloworld*. *Driver* này sẽ áp dụng tất cả những kỹ thuật đã được học. Nếu cần thiết, các bạn có thể xem lại lý thuyết cũ trước khi qua bài sau để nắm được những vấn đề cốt lõi trong lập trình *driver*.

cuu duong than cong . com

cuu duong than cong . com

CuuDuongThanCong.com https://fb.com/tailieudientucntt

BÀI 6

HELLO WORLD DRIVER

I. Mở đầu:

Đến đây, chúng ta đã có được những kiến thức gần như đầy đủ để tự mình viết một *character device driver*. Các bạn đã biết thế nào là *character driver*, số định danh lệnh, số định danh thiết bị, cấu trúc *inode*, *file structure*,...cũng như những lệnh khởi tạo và cập nhật chúng. Với những yêu cầu của từng lệnh, chúng ta đã rút ra được hai cấu trúc chung khi muốn viết một *driver* hoàn chỉnh, đã được tìm hiểu trong bài trước. Tùy vào từng cấu trúc mà có các bước cài đặt *driver* khác nhau. Thế nhưng, chúng ta chỉ mới dừng lại các thao tác trên giấy, chưa thực sự lập trình ra được một *driver* nào. Trong bài này, chúng ta sẽ viết một dự án có tên *helloworld* nhằm mục đích thực tế hóa những thao tác lệnh đã được trình bày trong phần driver.

Với mục đích là đem những thao tác lệnh đã được học vào thực tế chương trình, dự án này bao gồm có hai thành phần cần hoàn thành đó là driver và user application. Driver trong dự án sẽ áp dụng 3 giao diện chính dùng để trao đổi thông tin dữ liệu và điều khiển qua lại giữa hai lớp user và kernel, đó là các giao diện hàm read(), write() và ioctl() cùng với các hàm đóng mở driver như open() hay close(). Chương trình ứng dụng (Application) sẽ áp dụng những hàm về truy cập tập tin, ... để truy xuất những thông tin trong *driver*, hiển thị cho người dùng. Ngoài ra, *driver* và *application* đều có nhiệm vụ xuất thông tin có liên quan để người lập trình biết môi trường nào đang thực thi.

Theo những yêu cầu trên, đầu tiên chúng ta sẽ phân công tác vụ cho *driver* và *application* trong mục II sau đó lập trình theo những tác vụ đã phân công trong mục III. Người lập trình sẽ biên dịch, cài đặt chương trình vào hệ thống linux, thực thi và quan sát kết quả. Từ đó giải thích rút ra nhận xét.

II. Phân công tác vụ giữa *driver* và *application*:

helloworld là một dự án bao quát tất cả những kỹ thuật lập trình *driver* được nghiên cứu trong những nội dung trước. Nhiệm vụ chính là dùng hàm `printf()` và `printk()` xuất thông tin ra màn hình hiển thị theo một quy luật phù hợp, từ đó người lập trình có

CuuDuongThanCong.com https://fb.com/tailieudientucntt

thể hiểu nguyên lý hoạt động của từng hàm sử dụng để áp dụng vào trường hợp khác. *Driver* và *Application* đều có nhiệm vụ riêng, tùy vào từng giao diện sử dụng mà yêu cầu của từng ví dụ sẽ khác nhau, sao cho toát lên được ý nghĩa cốt lõi của giao diện hàm. Sau đây là chi tiết từng yêu cầu của dự án *helloworld*.

1. Driver:

a. Giao diện read():

Giao diện `read()` được sử dụng để chép thông tin từ *kernel space* sang *user space*. Thông tin được lưu trong *driver* là số khởi tạo ban đầu khi *driver* được mở chứa trong một biến cục bộ. Tại thời điểm gọi giao diện `read()`, thông tin này sẽ được chép qua *user space*, chứa trong biến khai báo trong *user application* thông qua hàm `copy_to_user()`.

Sau khi chép thông tin cho *user*, *driver* thông báo cho người lập trình biết quá trình chép thành công hay không.

b. Giao diện write():

Giao diện `write()` dùng để chép thông tin từ *user space* qua *kernel space*. Thông tin từ *user* là dữ liệu kiểu số do người dùng nhập vào chuyển qua *kernel* thông qua giao diện `write()` lưu vào một biến trong *kernel*, biến này cũng là nơi lưu thông tin được chuyển sang *user* khi giao diện `read()` được gọi.

Sau khi nhận dữ liệu từ *user*, *driver* thông báo cho người lập trình quá trình chép thành công. Ngược lại sẽ trả về mã lỗi.

c. Giao diện ioctl():

`ioctl()` là một giao diện hàm đa chức năng, nghĩa là chỉ cần một dạng câu lệnh mà có thể thực hiện được tất cả những chức năng khác. Để thể hiện được những chức năng này của hàm, chúng ta lập trình một ví dụ sau:

Xây dựng giao diện `ioctl()` thành một hàm toán học, thực hiện các phép toán cộng, trừ, nhân, chia. Các tham số được truyền từ *user*, sau khi tính toán xong, kết quả được gửi ngược lại *user*. Các phép toán được lựa chọn thông qua các số định danh lệnh.

2. Application:

CuuDuongThanCong.com https://fb.com/tailieudientucntt

Chương trình trong *user*, sử dụng kỹ thuật tùy chọn trong hàm `main()` để kiểm tra tất cả những chức năng do chương trình trong *driver* hỗ trợ. Mỗi chức năng sẽ được thực hiện do người sử dụng lựa chọn.

Trước khi đi vào viết chương trình cụ thể, chúng ta sẽ tìm hiểu hệ điều hành linux thực hiện tùy chọn trong hàm `main()` như thế nào:

Hàm `main()` là hàm được thực hiện đầu tiên khi chương trình ứng dụng được gọi thực thi. Từ hàm này, người lập trình sẽ tiến hành tất cả những chức năng như tạo lập tiến trình, tuyến, gọi các chương trình con, ... Thông thường hàm `main()` được khai báo như sau:

```
void main(void) {
/*Các lệnh do người lập trình định nghĩa*/
}
```

Với cách khai báo này thì hàm `main()` không có tham số cũng như không có dữ liệu trả về.

Ngoài ra linux còn hỗ trợ cho người lập trình cách khai báo hàm `main()` khác có dạng như sau:

```
int main (int argc, char **argv) {
   /*Các lệnh do người lập trình định nghĩa*/
}
```

Với cách lập trình này hàm `main()` có thể được người sử dụng cung cấp các tham số trong khi gọi thực thi. Cú pháp cung cấp tham số trong câu lệnh `shell` như sau:

```
./<tên chương trình> <tham số 1> <tham số 2> <...> <tham số n>
```

Hàm `main()` có hai tham số:

- Tham số thứ nhất `int argc` là số `int` lưu số lượng tham số khai báo trong câu lệnh *shell* trên, bao gồm cả tham số đầu tiên là tên chương trình chứa hàm `main()`.
- Tham số thứ hai `char **argv` là mảng con trỏ lưu nội dung từng tham số nhập trong câu lệnh gọi chương trình thực thi trong *shell*;

CuuDuongThanCong.com https://fb.com/tailieudientucntt

Như vậy tên chương trình chứa hàm `main()` sẽ thuộc về tham số đầu tiên có nội dung lưu trong `argv[0]`. Tương tự `<tham số 1>` là tham số thứ hai chứa trong `argv[1]`, ...Tương tự cho các tham số khác. Tổng quát nếu có n tham số trong câu lệnh shell thì trong hàm main có: `argc = n+1; argv[0], argv[1], argv[2], ..., argv[n]` lưu nội dung của từng tham số.

Trong chương trình ứng dụng *user application* của dự án *helloworld*, hàm `main()` được khai báo có dạng tùy chọn như trên. Khi người dùng nhập: `"add", "sub", "mul", "div", "read", "write"`, thì câu lệnh tương ứng sẽ thực hiện gọi các giao diện hàm cần thiết thực hiện chức năng cộng, trừ , nhân, chia, đọc và ghi được lập trình trong *dirver*. Thao tác cụ thể sẽ được chúng tôi chú thích trong từng dòng lệnh của *dirver* và *application*.

III. Chương trình driver và application:

1. ***Chương trình driver:*** *Chương trình driver mang tên helloworld_dev.c*

(Mã lệnh và giải thích chương trình driver chứa trong CD đính kèm)

2. ***Chương trình application:*** *Chương trình application mang tên helloworld_app.c*

(Mã lệnh và giải thích chương trình application chứa trong CD đính kèm)

3. **Thực thi chương trình:**

Các bước biên dịch và kết quả thực thi dự án chứa trong CD đính kèm

IV. Tổng kết:

Như vậy chúng ta đã hoàn thành công việc viết hoàn chỉnh một *driver* đơn giản dựa vào các bước mẫu trong bài học trước. Các bạn cũng đã hiểu nguyên lý hoạt động của hàm *main* có tham số, cách lựa chọn tham số hoạt động trong từng trường hợp cụ thể theo yêu cầu.

Với những thao tác trên, các bạn có thể tự mình viết những *dirver* đơn giản trong xử lý tính toán, xuất nhập thông báo, ... Thế nhưng, trong thực tế, *driver* không phải chỉ dùng trong việc truy xuất những ký tự thông báo. Nhiệm vụ chính của nó là điều khiển các thiết bị phần cứng thông qua các hàm giao tiếp với các cổng vào ra, truy xuất thanh

CuuDuongThanCong.com https://fb.com/tailieudientucntt

ghi lệnh, dữ liệu, ...của thiết bị, thu thập thông tin lưu vào vùng nhớ đệm trong *driver* chờ chương trình trong *user* truy xuất. Để có thể giao tiếp với các thiết bị phần cứng thông qua các cổng vào ra, trong bài sau chúng ta sẽ nghiên cứu các hàm giao tiếp gpio do *linux* hỗ trợ sẵn.

cuu duong than cong . com

cuu duong than cong . com

CuuDuongThanCong.com https://fb.com/tailieudientucntt

BÀI 7

CÁC HÀM HỖ TRỢ GPIO

I. Tổng quan về GPIO:

GPIO, viết tắt của cụm từ General Purpose *Input/Output*, là một thư viện phần mềm điều khiển các cổng vào ra tích hợp trên vi điều khiển hay các ngoại vi IO liên kết với vi điều khiển đó. Hầu hết các vi điều khiển đều hỗ trợ thư viện này, giúp cho việc lập trình các cổng vào ra trở nên thuận tiện hơn. Các tập lệnh vào ra và điều khiển, cách quy định số chân, ... hầu hết tương tự nhau so với các loại vi khiển khác nhau. Điều này làm tăng tính linh hoạt, giảm thời gian xây dựng hệ thống.

Theo như quy định chuẩn, mỗi một chân IO trên vi điều khiển sẽ tương ứng với một số GPIO của thư viện này. Số GPIO được quy định như sau: Đối với vi điều khiển ARM9260, số cổng vào ra là 3x32 cổng, tương ứng với 3 ports, đó là các Port A, Port B, và Port C. Mỗi chân quy định trong GPIO theo quy luật sau:

BASEx32+PIN;

cuu duong than cong . com

- Trong đó BASE là số cơ sở của Port. Port A có cơ sở là 1, Port B là 2, Port C là 3. PIN là số thứ tự của từng chân trong Port. Chân 0 có giá trị PIN là 32, 1 là 33, ... Ví dụ, chân thứ 2 của Port A có số GPIO là 33; Chân thứ 2 của Port B có số GPIO là 65, ... tương tự cho các chân còn lại trên vi điều khiển. Đối với các vi điều khiển khác có số Port lớn hơn ta chỉ việc tuân theo quy luật trên để tìm số GPIO phù hợp.

GPIO cho các loại vi điều khiển khác nhau điều có chung những tính chất:

- Mỗi chân trong GPIO đều có thể có hai chế độ *input* và *output*, tùy vào loại vi điều khiển mà GPIO đang sử dụng.
- Trong chế độ *input*, các chân GPIO có thể lập trình để trở thành nguồn ngắt hệ thống.

cuu duong than cong . com

- Và nhiều chức năng khác nữa, trong quyển sách này chúng ta chỉ tìm hiểu những chức năng phổ biến nhất phục vụ giao tiếp với các chân IO trong các chương trình ứng dụng.

CuuDuongThanCong.com https://fb.com/tailieudientucntt

Một trong những thao tác đầu tiên để đưa GPIO hoạt động trong hệ thống là xác định GPIO cần dùng bằng hàm:

```
gpio_request(); //Yêu cầu truy xuất chân GPIO
```

Tiếp đến, chúng ta cấu hình chân GPIO là ngõ vào hay ngõ ra bằng hai hàm:

```
int gpio_direction_input(unsigned gpio);
int gpio_direction_output(unsigned gpio, int value);
```

Công việc cuối cùng là đưa dữ liệu đến chân GPIO, nếu là ngõ ra; hoặc đọc dữ liệu từ chân GPIO, nếu là ngõ vào; ta sử dụng hai hàm sau:

```
int gpio_get_value(unsigned gpio); //Đọc dữ liệu;
void gpio_set_value(unsigned gpio, int value); //Xuất dữ liệu;
```

Ngoài ra còn có nhiều hàm chức năng khác sẽ được trình bày trong mục sau.

***Có nhiều cách thao tác với gpio. Hoặc thao tác với giao diện thiết bị trong cấu trúc root file system (đây là những dirver đã được lập trình sẵn), việc điều khiển sẽ là thao tác với tập tin và thư mục (lập trình trong user application). Hoặc dùng trực tiếp những lệnh trong mã nguồn kernel, nghĩa là người sử dụng tạo riêng cho mình một driver sử dụng trực tiếp các hàm giao tiếp với gpio sau đó mới viết chương trình ứng dụng điều khiển IO theo những giao diện trong driver hỗ trợ. Nhằm mục đích thuận tiện cho viêc điều khiển IO, phần lập trình nhúng nâng cao chỉ trình bay cách thứ hai, điều khiển trực tiếp IO thông qua thư viện gpio.h trong kernel.*

II. Các hàm chính trong GPIO:

1. Hàm gpio_is_valid():

Cú pháp hàm như sau:

```
#include <linux/gpio.h>
int gpio_is_valid (int number);
```

Do thư viện *gpio* dùng chung cho nhiều loại vi điều khiển khác nhau, nên số lượng chân IO của từng loại cũng khác nhau. Cần thiết phải cung cấp một hàm kiểm tra sự tồn tại của chân trong hệ thống. Hàm `gpio_is_valid()` làm nhiệm vụ này. Hàm có tham số là `int number` là số chân *gpio* muốn kiểm tra. Hàm trả về giá trị 0 nếu số chân *gpio*

CuuDuongThanCong.com https://fb.com/tailieudientucntt

cung cấp là hợp lệ, nghĩa là chân *gpio* này tồn tại trong hệ thống mà *gpio* đang hỗ trợ. Ngược lại, nếu chân *gpio* không hợp lệ, hàm trả về mã lỗi âm "`-EINVAL`".

Ví dụ, nếu muốn kiểm tra chân gpio 32 hợp lệ hay không, ta dùng đoạn mã lệnh sau:

```
/*Khai báo biến lưu mã lỗi trả về*/
int ret;
/*Gọi hàm kiểm tra gpio*/
ret = gpio_is_valid(32);
/*Kiểm tra mã lỗi trả về hệ thống*/
if (ret < 0) {
   /*Nếu xảy ra lỗi, in thông báo cho người dùng*/
   printk ("This gpio pin is not valid\n");
   /*Trả về mã lỗi cho hàm gọi*/
   return ret;
}
```

Công việc kiểm tra được thực hiện đầu tiên khi muốn thao tác với một chân nào đó trong gpio. Nếu hợp lệ các lệnh tiếp theo sẽ được thực hiện. Ngược lại, in ra thông báo cho người dùng và thoát ra khỏi driver.

2. Hàm gpio_request():

Cú pháp của hàm này như sau:

```
#include <linux/gpio.h>
int gpio_request ( unsigned gpio, const char *lable);
```

Sau khi kiểm tra tính hợp lệ của chân *gpio*, công việc tiếp theo là khai báo sử dụng chân gpio đó. *Linux kernel* cung cấp cho chúng ta hàm `gpio_request()` để thực hiện công việc này. Hàm `gpio_request` có hai tham số. Tham số thứ nhất `unsigned gpio` là chân gpio muốn khai báo sử dụng; Tham số thứ hai `const char *lable` là tên muốn đặt cho *gpio*, phục vụ cho quá trình thao tác với chân gpio dễ dàng hơn. Tham số này có thể để trống bằng cách dùng hằng số rỗng NULL.

Sau đây là đoạn chương trình mẫu minh họa cách sử dụng hàm `gpio_request()`:

```
/*Những đoạn lệnh trước*/
```


CuuDuongThanCong.com https://fb.com/tailieudientucntt

```
...
/*Khai báo biến lưu về mã lỗi cho hàm gọi*/
int ret;
/*Gọi hàm gpio_request (), yêu cầu sử dụng chân gpio 32 với tên là "EXPL"*/
ret = gpio_request (32, "EXPL");
/*Kiểm tra mã lỗi trả về*/
if (ret) {
   /*Nếu xảy ra lỗi thì in ra thông bào cho người sử dụng*/
printk (KERN_WARNING "EXPL: unable to request gpio 32");
   /*Trả về mã lỗi cho hàm gọi*/
return ret;
}
/*Nếu không có lỗi, thực thi những đoạn lệnh khác*/
...
```

3. *Hàm gpio_free():*

Cú pháp của hàm này như sau:

```
#include <linux/gpio.h>
void gpio_free (unsigned gpio);
```

Hàm `gpio_free()` có chức năng ngược lại với hàm `gpio_request()`. Hàm `gpio_free()` dùng giải phóng chân *gpio* nào đó không cần sử dụng cho hệ thống. Hàm này chỉ có một tham số trả về `unsigned gpio` là số *gpio* của chân muốn giải phóng. Hàm không có dữ liệu trả về.

Sau đây là đoạn chương trình ví dụ cách sử dụng hàm gpio_free:

```
/*Đoạn chương trình giải phóng chân gpio 32 ra khỏi driver*/
gpio_free(32);
```

4. *Hàm gpio_direction_input():*

Cú pháp của hàm này như sau:

```
#include <linux/gpio.h>
int gpio_direction_input (unsigned gpio);
```


CuuDuongThanCong.com https://fb.com/tailieudientucntt

Hàm `gpio_direction_input` dùng để cài đặt chế độ giao tiếp cho chân *gpio* là *input*. Nghĩa là chân gpio này sẽ nhận dữ liệu từ bên ngoài lưu vào vùng nhớ đệm bên trong. Hàm `gpio_direction_input` có tham số `unsigned gpio` là chân *gpio* muốn cài đặt chế độ *input*. Hàm trả về giá trị 0 nếu quá trình cài đặt thành công. Ngược lại, sẽ trả về mã lỗi âm.

Sau đây là một ví dụ cho hàm `gpio_direction_input()`:

```
/*Hàm cài đặt chân gpio 32 ở chế độ ngõ vào*/
/*Khai báo biến lưu giá trị mã lỗi trả về cho hàm gọi*/
int ret;
/*Cài đặt chân 32 chế độ ngõ vào*/
ret = gpio_direction_input (32);
/*Kiểm tra lỗi thực thi */
if (ret) {
   /*Nếu có lỗi in ra thông báo cho người dùng*/
   printk (KERN_WARNING "Unable to set input mode");
   /*Trả về mã lỗi cho hàm gọi*/
   return ret;
}
...
```

5. *Hàm gpio_direction_output():*

Cú pháp của hàm này như sau:

```
#include <linux/gpio.h>
int gpio_direction_output (unsigned gpio, int value);
```

Hàm `gpio_direction_output` dùng để cài đặt chế độ giao tiếp cho chân *gpio* là *output*. Chân *gpio* sẽ làm nhiệm vụ xuất dữ liệu bên trong chương trình ra ngoài. Hàm `gpio_direction_output()` có hai tham số. Tham số thứ nhất, `unsigned gpio`, là chân gpio muốn cài đặt. Tham số thứ hai, `int value`, là giá trị ban đầu của *gpio* khi nó

CuuDuongThanCong.com https://fb.com/tailieudientucntt

là ngõ ra. Hàm trả về giá trị 0 nếu quá trình cài đặt thành công. Ngược lại, sẽ trả về mã lỗi âm.

Sau đây là một ví dụ cho hàm `gpio_direction_output()`:

```
/*Hàm cài đặt chân gpio 32 ở chế độ ngõ ra, giá trị ban đầu là 1*/
/*Khai báo biến lưu giá trị mã lỗi trả về cho hàm gọi*/
int ret;
/*Cài đặt chân 32 chế độ ngõ vào*/
ret = gpio_direction_output (32, 1);
/*Kiểm tra lỗi thực thi */
if (ret) {
   /*Nếu có lỗi in ra thông báo cho người dùng*/
   printk (KERN_WARNING "Unable to set gpio 32 into output mode");
   /*Trả về mã lỗi cho hàm gọi*/
   return ret;
}
```

6. *Hàm gpio_get_value():*

Cú pháp của hàm này như sau:

```
#include <linux/gpio.h>
int gpio_get_value (unsigned gpio);
```

Khi chân `unsigned gpio` là ngõ vào, hàm `gpio_get_value()` sẽ lấy giá trị tín hiệu của chân này. Hàm có giá trị trả về dạng *int*, bằng 0 nếu ngõ vào mức thấp, khác 0 nếu ngõ vào mức cao.

Sau đây là một ví dụ đọc vào giá trị chân *gpio* 32, kiểm tra và in thông tin ra màn hình:

```
...
if (gpio_get_value(32)) {
   printk ("The input is high value\n");
} else {
   printk ("The input is low value\n");
```


CuuDuongThanCong.com https://fb.com/tailieudientucntt

```
}
...
```

7. *Hàm gpio_set_value():*

Cú pháp của hàm này như sau:

```
#include <linux/gpio.h>
void gpio_set_value (unsigned gpio, int value);
```

Ngược lại với hàm `gpio_get_value()`, hàm `gpio_set_value()` có chức năng xuất dữ liệu cho một chân `unsigned gpio` ngõ ra. Với dữ liệu chứa trong tham số `int value`. Bằng 0 nếu muốn xuất ra mức thấp, bằng 1 nếu muốn xuất ra mức cao. Hàm `gpio_set_value()` không có dữ liệu trả về.

Sau đây là một ví dụ xuất mức cao ra chân gpio 32:

```
/*Các lệnh khởi tạo gpio 32 là ngõ ra*/
...
/*Xuất mức cao ra chân gpio 32 */
gpio_set_value (32, 1);
/*Các lệnh xử lý tiếp theo*/
...
```

cuu duong than cong . com

cuu duong than cong . com

CuuDuongThanCong.com https://fb.com/tailieudientucntt

8. *Hàm gpio_to_irq():*

Cú pháp của hàm này như sau:

```
#include <linux/gpio.h>
int gpio_to_irq (unsigned gpio);
```

Đôi khi chúng ta muốn cài đặt chế độ ngắt cho một chân nào đó trong *gpio* dùng để thu nhận thông tin đồng bộ từ phần cứng. Hàm `gpio_to_irq()` thực hiện chức năng này. Hàm có tham số `unsigned gpio` là chân *gpio* muốn cài đặt chế độ ngắt. Hàm có giá trị trả về là số IRQ nếu quá trình cài đặt chế độ ngắt thành công. Ngược lại sẽ trả về mã lỗi âm.

Số IRQ được sử dụng cho hàm `request_irq()` khởi tạo ngắt cho hệ thống, hàm sẽ gán số IRQ với hàm xử lý ngắt. Hàm này sẽ thực hiện những thao tác do người lập trình quy định khi xuất hiện ngắt từ tín hiệu ngắt có số định danh ngắt là IRQ. Bên cạnh đó hàm `request_irq()` còn quy định chế độ ngắt cho chân *gpio* là ngắt theo cạnh hay ngắt theo mức, ...

cuu duong than cong . com

Sau đây là đoạn chương trình ví dụ khởi tạo ngắt từ chân gpio.

```
/*Đoạn chương trình khởi tạo ngắt cho chân gpio 70, PC6*/
/*Khai báo biến lưu mã lỗi trả về hàm gọi*/
int ret;
/*Yêu cầu chân gpio, với định danh là IRQ*/
  ret = gpio_request (70, "IRQ");
/*Kiểm tra mã lỗi trả về*/
    if (ret) {
/*Thông báo cho người lập trình có lỗi xảy ra*/
        printk (KERN_ALERT "Unable to request PC6\n");
    }
/*Cài đặt chế độ kéo lên cho chân gpio*/
    at91_set_GPIO_periph (70, 1);
/*Cài đặt chân gpio là input*/
    at91_set_gpio_input  (70, 1);
```

cuu duong than cong . com

CuuDuongThanCong.com https://fb.com/tailieudientucntt

```
/*Cài đặt chân gpio có chế độ chống lỗi*/
    at91_set_deglitch    (70, 1);
/*Cài đặt gpio thành nguồn ngắt, lưu về số định danh ngắt irq*/
  irq = gpio_to_irq (70);   /* Get IRQ number */
/*Thông báo chân gpio đã khởi tạo ngắt */
    printk (KERN_ALERT "IRQ = %d\n", irq);
/*Khai báo ngắt chân gpio cho hệ thống*/
    ret = request_irq (irq, gpio_irq_handler,
                       IRQF_TRIGGER_RISING                    |
IRQF_TRIGGER_FALLING,
                       "MyIRQ", NULL);
/*Kiểm tra mã lỗi trả về khi khai báo ngắt*/
    if (ret)  {
/*Thông báo cho người sử dụng số định danh ngắt không còn trống*/
     printk (KERN_ALERT "IRQ %d is not free\n", irq);
/*Trả về mã lỗi cho hàm gọi */
          return ret;
    }
```

Như vậy để tránh lỗi trong quá trình khởi tạo ngắt cho chân gpio, trước tiên chúng ta phải cài đặt những thông số cần thiết cho gpio đó, chẳng hạn như chân gpio phải hợp lệ và ở chế độ ngõ vào.

cuu duong than cong . com

CuuDuongThanCong.com https://fb.com/tailieudientucntt

III. Kết luận:

Trên đây, chúng ta đã sử dụng được những hàm điều khiển các chân gpio ngoại vi để thực hiện chức năng cụ thể của yêu cầu ứng dụng. Kết hợp với những kiến thức trong những bài trước: Giao diện điều khiển liên kết *user space* với *kernel space*, các hàm trì hoãn thời gian trong *user space*, các kỹ thuật lập trình C, ... chúng ta có thể viết được hầu hết tất cả những ứng dụng có liên quan đến các cổng vào ra: chẳng hạn như điều khiển LED, điều khiển LCD, ...

gpio không chỉ điều khiển các cổng vào ra trên vi điều khiển, theo yêu cầu trong thực tế, đa số các kit nhúng, số cổng vào ra rất hạn chế, đòi hỏi phải có các chân io mở rộng từ những linh kiện phụ trợ khác, vì thế *gpio* còn hỗ trợ thêm các hàm điều khiển các chân io mở rộng. Vấn đề này sẽ được đề cập trong những tài liệu khác không thuộc phạm vi của giáo trình này.

Trước khi di vào ứng dụng các hàm thao tác với *gpio* trong điều khiển LED đơn, LCD, ... bài tiếp theo sẽ tìm hiểu thêm về cách trì hoãn thời gian trong *kernel*, với cách trì hoãn thời gian này, chúng ta không cần dùng đến các hàm trì hoãn khác trong user space vì lý do yêu cầu đồng bộ hóa phần cứng hệ thống, hay một số trường hợp chúng ta muốn sử dụng delay trong driver gắn liền với những thao tác điều khiển.

cuu duong than cong . com

CuuDuongThanCong.com https://fb.com/tailieudientucntt

BÀI 8

THAO TÁC THỜI GIAN TRONG KERNEL

I. Sơ lược về thời gian trong kernel:

Thời gian trong hệ điều hành linux nói riêng và các hệ điều hành khác nói chung đều rất quan trọng. Trong hệ điều hành, những hoạt động đều dựa theo sự tác động mang tính chất chu kỳ. Chẳng hạn như hoạt động chia khe thời gian thực thi, chuyển qua lại giữa các tiến trình với nhau, đồng bộ hóa hoạt động giữa các thiết bị phần cứng có thời gian truy xuất không giống nhau, và nhiều hoạt động quan trọng khác nữa.

Khi làm việc với hệ thống linux, chúng ta cần phân biệt hai khái niệm thời gian: Thời gian tuyệt đối và thời gian tương đối. Thời gian tuyệt đối được hiểu như thời gian thực của hệ thống, là các thông tin như ngày-tháng-năm-giờ-phút-giây và các đơn vị thời gian khác nhỏ hơn, phục vụ cho người sử dụng. Thời gian tương đối được hiểu như những khoảng thời gian được cập nhật không cố định, mang tính chất chu kỳ, mốc thời gian không cố định và thông thường không biết trước. Ví dụ thời gian tuyệt đối là thời điểm trong một ngày, có mốc thời gian tính từ ngày 1 tháng 1 năm 1970 quy định trong các thiết bị thời gian thực. Thời gian tương đối là khoảng thời gian tính từ thời điểm xảy ra một sự kiện nào đó, chẳng hạn định thời tác động một khoảng thời gian sau khi hệ thống có lỗi hoặc cập nhật thông số hiện tại của hệ thống sau mỗi một thời khoảng cố định.

Để có thể kiểm tra, quản lý và thao tác với thời gian một cách chính xác, hệ điều hành phải dựa vào thiết bị thời gian được tích hợp trên hầu hết các vi xử lý đó là *timer*. *Timer* được sử dụng bởi hệ điều hành được gọi là *timer* hệ thống, hệ điều hành cài đặt các thông số thích hợp cho *timer*, quy định khoảng thời gian sinh ra ngắt, khoảng thời gian giữa hai lần xảy ra ngắt liên tiếp được gọi bởi thuật ngữ *tick*, giá trị của *tick* do người sử dụng *driver* quy định. Khi xảy ra ngắt, hệ điều hành linux sẽ cập nhật giá trị của biến *jiffies*, chuyển tiến trình, cập nhật thời gian trong ngày, ...

Trong phần lập trình *user application*, chúng ta đã tìm hiểu những hàm thao tác với thời gian *user space*. Đây là những hàm được hỗ trợ sẵn bởi hệ điều hành, nghĩa là chúng được lập trình để hoạt động ổn định không gây ảnh hưởng đến các tiến trình khác chạy

CuuDuongThanCong.com https://fb.com/tailieudientucntt

đồng thời. Trong *kernel*, những hàm thao tác với thời gian hoạt động bên ngoài tầm kiểm soát của hệ điều hành, sử dụng trực tiếp tài nguyên phần cứng của hệ thống, cụ thể là thời gian hoạt động của vi xử lý trung tâm trong vi điều khiển. Vì thế nếu sử dụng không phù hợp thì hệ điều hành sẽ hoạt động không ổn định, xảy ra những lỗi về thời gian thực trong khi thực hiện tác vụ. Nhưng thời gian thực hiện của các hàm này sẽ nhanh hơn, vì không qua trung gian là hệ điều hành, phù hợp với yêu cầu điều khiển của *driver* là nhanh chóng và chính xác.

Trong bài này, đầu tiên chúng ta sẽ tìm hiểu nguyên tắc quản lý thời gian trong *kernel*, sau đó là các hàm tương tác, xử lý thời gian thực, cách sử dụng định thời trong *timer*, và cuối cùng là một số kỹ thuật trì hoãn thời gian trong *kernel*.

II. Đơn vị thời gian trong *kernel*:

Để quản lý chính xác thời gian, hệ điều hành sử dụng bộ định thời *timer* tích hợp sẵn trên vi điều khiển mà nó hoạt động. Bộ định thời (*timer*) được đặt cho một giá trị cố định sao cho có thể sinh ra ngắt theo chu kỳ cho trước. Khoảng thời gian của một chu kỳ ngắt được gọi là *tick*. Trong một giây có khoảng *N ticks* được hoàn thành. Hay nói cách khác, tốc độ tick là *N Hz*. Giá trị *N* được định nghĩa bởi người sử dụng trước khi biên dịch *kernel*. Thông thường một số hệ điều hành có giá trị mặc định là *N = 100* và đây cũng là giá trị mặc định của hệ điều hành chạy trên kit *KM9260*.

Giá trị của HZ được định nghĩa như sau:

```
# define USER_HZ 100    /* User interfaces are in "ticks"
*/
```

và

```
# define HZ 100     /*Internal kernel timer frequency*/
```

trong tập tin `\arch\arm\include\asm\param.h`. Chúng ta thấy, giá trị mặc định của HZ là 100. Có nghĩa là *timer* hệ thống được cài đặt sao cho trong một giây có khoảng 100 lần ngắt xảy ra. Hay nói cách khác, chu kỳ ngắt là 10 ms. Chúng ta cũng chú ý, trong tập tin có hai tham số cần sữa chữa một lúc đó là *USER_HZ* và *HZ*. Để thuận tiện cho việc chuyển đổi qua lại thời gian giữa *kernel* và *user* thì giá trị của chúng phải giống nhau.

CuuDuongThanCong.com https://fb.com/tailieudientucntt

Giá trị *HZ* thay đổi phải phù hợp với ứng dụng mà hệ điều hành đang phục vụ. Làm thế nào để làm được điều này.

Chẳng hạn, giá trị của *HZ* ảnh hưởng rất nhiều đến độ phân giải của những hàm định thời ngắt. Với giá trị *HZ = 100*, thời gian lập trình định thời ngắt tối thiểu là *10ms*. Với giá trị *HZ=1000*, thời gian lập trình định thời ngắt tối thiểu là *1ms*. Điều nàycó nghĩa là giá trị của *HZ* càng lớn càng tốt. Liệu thực sự có phải như thế?

Khi tăng giá trị của *HZ*, điều có những ưu và nhược điểm. Về ưu điểm, tăng giá trị *HZ* làm độ phân giải của thời gian *kernel* tăng lên, như thế một số ứng dụng có liên quan đến thời gian cũng chính xác hơn ,...Về nhược điểm, khi tăng giá trị của *HZ*, thì vi điều khiển thực hiện ngắt nhiều hơn, tiêu tốn thời gian cho việc lưu lưu ngăn xếp, khởi tạo ngắt, ... nhiều hơn, ... do vậy tùy từng ứng dụng cụ thể mà chúng ta thay đổi giá trị *HZ* cho phù hợp để hệ thống hoạt động tối ưu.

III. jiffies:

jiffies là một biến toàn cục được định nghĩa trong thư viện `linux/jiffies.h` để lưu số lượng *ticks* đạt được kể từ khi hệ thống bắt đầu khởi động. Khi xảy ra ngắt timer hệ thống, kernel tiến hành tăng giá trị của jiffies lên 1 đơn vị. Như vậy nếu như có *HZ* ticks trong một giây thì *jiffies* sẽ tăng trong một giây là *HZ* đơn vị. Nếu như chúng ta đọc được giá trị *jiffies* hiện tại là *N*, thì thời gian kể từ khi hệ thống khởi động là *N* ticks hay *N/HZ* giây. Đôi khi chúng ta muốn đổi giá trị *jiffies* sang giây và ngược lại ta chỉ việc chia hay nhân giá trị *jiffies* cho(với) *HZ*.

Trong *kernel*, *jiffies* được lưu trữ dưới dạng số nhị phân 32 bits. Là 32 bits có trong số thấp trong tổng số 64 bits của biến *jiffies_64*. Với chu kỳ *tick* là 10ms thì sau một khoảng thời gian 5.85 tỷ năm đối với biến *jiffies_64* và *1.36* năm đối với biến *jiffes* mới có thể bị tràn. Xác suất để biến *jiffies_64* bị tràn là cực kỳ nhỏ và *jiffies* là rất nhỏ. Thế nhưng vẫn có thể xảy ra đối với những ứng dụng đòi hỏi độ tin cậy rất cao. Nếu cần có thể kiểm tra nếu yêu cầu chính xác cao.

Khi thao tác với *jiffies*, *kernel* hỗ trợ cho chúng ta các hàm so sánh thời gian sau, tất cả các hàm này đều được định nghĩa trong thư viện `linux/jiffies.h`:

CuuDuongThanCong.com https://fb.com/tailieudientucntt

Đầu tiên là hàm *get_jiffies_64(),* với giá trị jiffies thì chúng ta có thể đọc trực tiếp theo tên của nó, thế nhưng *jiffies_64* không thể đọc trực tiếp mà phải thông qua hàm riêng vì giá trị của *jiffies_64* được chứa trong số 64 bits. Hàm không có tham số, giá trị trả về của hàm là số có 64 bits.

Cuối cùng là các hàm so sánh thời gian theo giá trị của *jiffies*. Các hàm này được định nghĩa trong thư viện `linux/jiffies.h` như sau:

```
#define time_after(unknown,known)(long)(known)-(long)(unknown)<0)
#define time_before(unknown,known)((long)(unknown)-(long)(known)<0)
#define time_after_eq(unknown,known)((long)(unknown)-(long)(known)>= 0)
#define time_before_eq(unknown,known)((long)(known)-(long)(unknown)>= 0)
```

các hàm này trả về giá trị kiểu `boolean`, tùy theo tên hàm và tham số của hàm. Hàm `time_after(unknown, known)` trả về giá trị đúng nếu `unknown > known`, tương tự cho các hàm khác. Ứng dụng của các hàm này khi chúng ta muốn so sánh hai khoảng thời gian với nhau để thực thi một tác vụ nào đó, chẳng hạn ứng dụng trong trì hoãn thời gian như trong đoạn chương trình sau:

cuu duong than cong . com

```
/*Đoạn chương trình trì hoãn thời gian 1s dùng jiffies*/
/*Khai báo biến lưu thời điểm cuối cùng  muốn so sánh*/
unsigned long timeout = jiffies + HZ; //Trì hoãn 1s
/*Kiểm tra xem giá trị timeout có bị tràn hay không*/
if (time_after(jiffies, timeout)) {
   printk("This timeout is overflow\n");
   return -1;
}
/*Thực hiện trì hoãn thời gian nếu không bị tràn*/
if (time_before(jiffies, timeout)) {
   /*Do nothing loop to delay*/
}
```

cuu duong than cong . com

IV. Thời gian thực trong kernel:

Trong phần lập trình ứng dụng *user*, chúng ta đã tìm hiểu kỹ về các lệnh xử lý thời gian thực trong thư viện `<time.h>`. Trong *kernel* cũng có những hàm được xây dựng sẵn

CuuDuongThanCong.com https://fb.com/tailieudientucntt

thao tác với thời gian thực nằm trong thư viện <linux/time.h>. Sự khác biệt giữa hai thư viện này là vai trò của chúng trong hệ thống. <time.h> chứa trong lớp *user*, giao tiếp với *kernel*, tương tự như các hàm giao diện trung gian giao tiếp giữa người dùng với thời gian thực trong *kernel*, vì thế thư viện chứa những hàm chủ yếu phục vụ cho người dùng; Đối với thư viện <linux/time.h> hoạt động trong lớp *kernel*, chứa những hàm được lập trình sẵn phục vụ chủ yếu cho hệ thống *kernel*, giúp người lập trình *driver* quản lý thời gian thực tiện lợi hơn, ví dụ như các hàm xử lý thời gian ngắt, định thời, so sánh thời gian, ... điểm đặc biệt là thời gian thực trong *kernel* chủ yếu quản lý dưới dạng số giây tuyệt đối, không theo dạng ngày tháng năm như trong *user*.

Các kiểu cấu trúc thời gian, ý nghĩa các tham số trong những hàm thời gian đa phần tương tự như trong thư viện <time.h> trong *user application* nên chúng sẽ được trình bày sơ lược trong khi giải thích.

1. **Các kiểu, cấu trúc thời gian:**

a. ***Cấu trúc timespec:*** cấu trúc này được định nghĩa như sau:

```
struct timespec {
   __kernel_time_t     tv_sec;         /* seconds */
   long            tv_nsec;        /* nanoseconds */
};
```

Trong đó, tv_sec dùng lưu thông tin của giây; tv_nsec dùng lưu thông tin nano giây của giây hiện tại.

b. Cấu trúc timeval: Cấu trúc này được định nghĩa như sau:

```
struct timeval {
   __kernel_time_t          tv_sec;         /* seconds */
   __kernel_suseconds_t     tv_usec;  /* microseconds */
};
```

Trong đó, tv_sec dùng lưu thông tin về số giây; tv_usec dùng lưu thông tin về micro giây của giây hiện tại.

c. Cấu trúc timezone: Cấu trúc này được định nghĩa như sau:

```
struct timezone {
```


CuuDuongThanCong.com https://fb.com/tailieudientucntt

```
    int   tz_minuteswest;      /* minutes west of Greenwich */
    int   tz_dsttime;     /* type of dst correction */
};
```

Trong đó, `tz_minuteswst` là số phút chênh lệch múi giờ về phía Tây của Greenwich; `tz_dsttime` là tham số điều chỉnh chênh lệch thời gian theo mùa;

2. Các hàm so sánh thời gian:

a. timespec_equal: Hàm được định nghĩa như sau:

```
static inline int timespec_equal(const struct timespec *a,
                                 const struct timespec *b)
{
    return  (a->tv_sec  ==  b->tv_sec)  &&  (a->tv_nsec  ==  b-
>tv_nsec);
}
```

Nhiệm vụ của hàm là so sánh 2 con trỏ cấu trúc thời gian kiểu `timespec`, `const struct timespec *a` và `const struct timespec *b`. So sánh từng thành phần trong cấu trúc, `tv_sec` và `tv_nsec`, nếu hai thành phần này bằng nhau thì hai cấu trúc thời gian này bằng nhau. Khi đó hàm trả về giá trị *true*, ngược lại trả về giá trị *false*;

b. timespec_compare: Hàm được định nghĩa như sau:

```
static inline int timespec_compare(const struct timespec *lhs,
const struct timespec *rhs)
{
    if (lhs->tv_sec < rhs->tv_sec)
        return -1;
    if (lhs->tv_sec > rhs->tv_sec)
        return 1;
    return lhs->tv_nsec - rhs->tv_nsec;
}
```

Nhiệm vụ của hàm là so sánh hai cấu trúc thời gian kiểu `timespec`, `const struc timespec *lhs` và `const struct timespec *rhs`. Kết quả là một trong 3 trường hợp sau:

CuuDuongThanCong.com https://fb.com/tailieudientucntt

- Nếu `lhs < rhs` thì trả về giá trị nhỏ hơn 0;
- Nếu `lhs = rhs` thì trả về giá trị bằng 0;
- Nếu `lhs > rhs` thì trả về giá trị lớn hơn 0;

*c. **timeval_compare:*** Hàm được định nghĩa như sau:

```
static inline int timeval_compare(const struct timeval *lhs,
const struct timeval *rhs)
{
   if (lhs->tv_sec < rhs->tv_sec)
        return -1;
   if (lhs->tv_sec > rhs->tv_sec)
        return 1;
   return lhs->tv_usec - rhs->tv_usec;
}
```

Nhiệm vụ của hàm là so sánh hai cấu trúc thời gian kiểu `timeval`, `const struct timeval *lhs` và `const struc timeval *rhs`. Kết quả trả về là một trong 3 trường hợp:

- Nếu `lhs < rhs` thì trả về giá trị nhỏ hơn 0;
- Nếu `lhs = rhs` thì trả về giá trị bằng 0;
- Nếu `lhs > rhs` thì trả về giá trị lớn hơn 0;

3. Các phép toán thao tác trên thời gian:

*a. **mktime:*** Hàm được định nghĩa như sau:

```
extern unsigned long mktime(
const unsigned int year, const unsigned int mon,
const unsigned int day, const unsigned int hour,
const unsigned int min, const unsigned int sec);
```

Nhiệm vụ của hàm là chuyển các thông tin thời gian dạng ngày tháng năm ngày giờ phút giây thành thông tin thời gian dạng giây tính từ thời điểm epoch.

*b. **set_normalized_timespec:*** Hàm được định nghĩa như sau:

```
extern void set_normalized_timespec(struct timespec *ts,
time_t sec, long nsec);
```


CuuDuongThanCong.com https://fb.com/tailieudientucntt

Sau khi chuyển các thông tin ngày tháng năm ... thành số giây tính từ thời điểm epoch bằng cách sử dụng hàm `mktime`, thông tin trả về chưa phải là một cấu trúc thời gian chuẩn có thể xử lý được trong `kernel`. Đề biến thành cấu trúc thời gian chuẩn ta sử dụng hàm `set_normalize_timespec` để chuyển số giây dạng unsigned long thành cấu trúc thời gian `timespec`.

Hàm có 3 tham số. Tham số thứ nhất, `struct timespec *ts`, là con trỏ trỏ đến cấu trúc `timespec` được định nghĩa trước đó lưu giá trị thông tin thời gian trả về sau khi chuyển đổi xong; Tham số thứ hai, `time_t sec`, là số giây muốn chuyển đổi sang cấu trúc `timespec` (được lưu vào trường `tv_sec`); Tham số thứ ba, `long nsec`, là số `nano` giây của giây của thời điểm muốn chuyển đổi.

c. ***timespec_add_safe:***Hàm được định nghĩa như sau:

```
extern struct timespec timespec_add_safe(
const struct timespec lhs,
const struct timespec rhs);
```

Nhiệm vụ của hàm là cộng hai cấu trúc thời gian kiểu `timespec`, `const struct timespec lhs` và `const struct timespec rhs`. Kết quả trả về dạng `timespec` là tổng của hai giá trị thời gian nếu không bị tràn, nếu bị tràn thì hàm sẽ trả về giá trị lớn nhất có thể có của kiểu `timespec`.

d. ***timespec_sub:***Hàm được định nghĩa như sau:

```
static inline struct timespec timespec_sub(struct timespec
lhs,struct timespec rhs)
{
   struct timespec ts_delta;
   set_normalized_timespec(&ts_delta, lhs.tv_sec - rhs.tv_sec,
                  lhs.tv_nsec - rhs.tv_nsec);
   return ts_delta;
}
```

Nhiệm vụ của hàm này là trừ hai cấu trúc thời gian kiểu `timespec`, `struct timespec lhs` và `struct timespec rhs`. Hiệu của hai cấu trúc này được trả về

CuuDuongThanCong.com https://fb.com/tailieudientucntt

dưới dạng `timespec`. Thông thường hàm dùng để tính toán khoảng thời gian giữa hai thời điểm.

e. timespec_add_ns: Hàm được định nghĩa như sau:

```
static __always_inline void timespec_add_ns(struct timespec *a, u64 ns)
{
    a->tv_sec+=__iter_div_u64_rem(a->tv_nsec+ns, NSEC_PER_SEC, &ns);
    a->tv_nsec = ns;
}
```

Nhiệm vụ của hàm này là cộng thêm một khoảng thời gian tính bằng nano giây vào cấu trúc `timespec` được đưa vào hàm. Hàm có hai tham số. Tham số thứ nhất, `struct timespec *a`, là cấu trúc `timespec` muốn cộng thêm. Tham số thứ hai, `u64 ns`, là số nano giây muốn cộng thêm vào.

4. Các hàm truy xuất thời gian:

a. do_gettimeofday:Hàm được định nghĩa như sau:

```
extern void do_gettimeofday(struct timeval *tv);
```

Nhiệm vụ của hàm là lấy về thông tin thời gian hiện tại của hệ thống. Thông tin thời gian hiện tại được trả về cấu trúc dạng `struct timeval *tv` trong tham số của hàm.

b. do_settimeofday: Hàm được định nghĩa như sau:

```
extern int do_settimeofday(struct timespec *tv);
```

Nhiệm vụ của hàm là cài đặt thông tin thời gian hiện tại cho hệ thống dựa vào cấu trúc thời gian dạng `timespec` chứa trong tham số `struct timespec *tv` của hàm.

c. do_sys_settimeofday: Hàm được định nghĩa như sau:

```
extern int do_sys_settimeofday(struct timespec *tv, struct timezone *tz);
```

Nhiệm vụ của hàm là cài đặt thông tin thời gian hiện tại của hệ thống có tính đến sự chênh lệch thời gian về múi giờ dựa vào hai tham số trong hàm. Tham số thứ nhất,

CuuDuongThanCong.com https://fb.com/tailieudientucntt

`struct timespec *tv`, là thông tin thời gian muốn cập nhật. Tham số thứ hai, `sruct timezone *tz`, là cấu trúc lưu thông tin chệnh lệch múi giờ muốn cập nhật.

d. getboottime: Hàm được định nghĩa như sau:

```
extern void getboottime (struct timespec *ts);
```

Nhiệm vụ của hàm là lấy về tổng thời gian từ khi hệ thống khởi động đến thời điểm hiện tại. Thông tin thời gian được lưu về cấu trúc kiểu `timespec`, `srtuc timespec`.

5. Các hàm chuyển đổi thời gian:

a. timespec_to_ns: Hàm được định nghĩa như sau:

```
static inline s64 timespec_to_ns(const struct timespec *ts)
{
   return ((s64) ts->tv_sec * NSEC_PER_SEC) + ts->tv_nsec;
}
```

Nhiệm vụ của hàm là chuyển cấu trúc thời gian dạng `timespec`, `const struct timespec *ts`, thành số nano giây lưu vào biến 64 bits làm giá trị trả về cho hàm.

cuu duong than cong . com

b. timeval_to_ns: Hàm được định nghĩa như sau:

```
static inline s64 timeval_to_ns(const struct timeval *tv)
{
   return ((s64) tv->tv_sec * NSEC_PER_SEC) +
        tv->tv_usec * NSEC_PER_USEC;
}
```

Nhiệm vụ của hàm là chuyển đổi cấu trúc thời gian dạng `timeval`, `const struct timeval *tv`, thành số nano giây lưu vào biến `64 bits` làm giá trị trả về cho hàm.

c. ns_to_timespec: Hàm được dịnh nghĩa như sau

```
extern struct timespec ns_to_timespec(const s64 nsec);
```

cuu duong than cong . com

Nhiệm vụ của hàm là chuyển thông tin thời gian dưới dạng nano giây, `const s64 nsec`, thành thông tin thời gian dạng `timespec` làm giá trị trả về cho hàm.

d. ns_to_timeval: Hàm được định nghĩa như sau:

```
extern struct timeval ns_to_timeval(const s64 nsec);
```


CuuDuongThanCong.com https://fb.com/tailieudientucntt

Nhiệm vụ của hàm là chuyển thông tin thời gian dạng nano giây, `const s64 nsec`, thành thông tin thời gian dạng `timeval` làm giá trị trả về cho hàm.

V. Timer và ngắt dùng timer:

1. Khái quát về timer trong kernel:

Timer là một định nghĩa quen thuộc trong hầu hết tất cả các hệ thống khác nhau. Trong *linux kernel*, *timer* được hiểu là trì hoãn thời gian động để gọi thực thi một hàm nào đó được lập trình trước. Nghĩa là thời điểm bắt đầu trì hoãn không cố định, thông thường được tính từ lúc cài đặt khởi động *timer* trong hệ thống, khoảng thời gian trì hoãn do người lập trình quy định, khi hết thời gian trì hoãn, *timer* sinh ra một ngắt. *Kernel* tạm hoãn hoạt động hiện tại của mình để thực thi hàm ngắt được lập trình trước đó.

Để đưa một *timer* vào hoạt động, chúng ta cần phải thực hiện nhiều thao tác. Đầu tiên phải khởi tạo *timer* vào hệ thống *linux*. Tiếp theo, cài đặt khoảng thời gian mong muốn trì hoãn. Cài đặt hàm thực thi ngắt khi thời gian trì hoãn kết thúc. Khi xảy ra hoạt động ngắt, *timer* sẽ bị vô hiệu hóa. Vì thế, nếu muốn *timer* hoạt động theo chu kỳ cố định chúng ta phải khởi tạo lại *timer* ngay trong chương trình thực thi ngắt. Số lượng *timer* khởi tạo trong *kernel* là không giới hạn, vì đây là một timer mềm không phải là *timer* vật lý, có giới hạn là dung lượng vùng nhớ cho phép. Chúng ta sẽ trình bày cụ thể những bước trên trong phần sau.

2. Các bước sử dụng timer:

***Lưu ý: Khi sử dụng các hàm trong timer chúng ta phải thêm thư viện `<linux/timer.h>` ở đầu chương trình.*

a. Bước 1: Khai báo biến cấu trúc `timer_list` để lưu *timer* khi khởi tạo

Cấu trúc `timer_list` được *linux kernel* định nghĩa trong thư viện `<linux/timer.h>` như sau:

```
struct timer_list {
struct list_head entry;
unsigned long expires;
void (*function) (unsigned long);
unsigned long data;
```


CuuDuongThanCong.com https://fb.com/tailieudientucntt

```
struct tvec_t_base_s *base;
}
```

Ta khai báo timer bằng dòng lệnh sau:

```
/*Khai báo một timer có tên là my_timer*/
struct timer_list my_timer;
```

b. Bước 2: Khai báo và định nghĩa hàm thực thi ngắt

Hàm thực thi ngắt có dạng như sau:

```
void my_timer_function (unsigned long data) {
/*Các thao tác do người lập trình driver quy định*/
...
/*Nếu cần thiết có thể khởi tạo lại timer trong phần cuối chương trình*/
}
```

c. Bước 3: Khởi tạo các tham số cho cấu trúc *timer*

Công việc tiếp theo là yêu cầu *kernel* cung cấp một vùng nhớ cho *timer* hoat động, chúng ta sử dụng câu lệnh:

```
/*Khởi tạo timer vừa khai báo my_timer*/
init_timer (&my_timer);
```

Sau khi *kernel* dành cho *timer* một vùng nhớ, *timer* vẫn còn trống chưa được gán những thông số cần thiết, công việc này của người lập trình *driver*:

```
/*Khởi tạo giá trị khoảng thời gian muốn trì hoãn, đơn vị tính bằng tick*/
my_timer.expires = jiffies  + delay;
/*Gán tham số cho hàm thực thi ngắt, nếu không có tham số ta có thể gán một giá trị bất kỳ*/
my_timer.data = 0;
/*Gán con trỏ hàm xử lý ngắt vào timer*/
my_timer.function = my_function;
```

Trong các tham số trên, chúng ta cần chú ý những tham số sau đây:

CuuDuongThanCong.com https://fb.com/tailieudientucntt

- `my_timer.expires` đây là giá trị thời gian tương lai có đơn vị là *tick*, tại mọi thời điểm, *timer* so sánh với số *jiffies*. Nếu số *jiffies* bằng với số `my_timer.expires` thì ngắt xảy ra.
- `my_timer.data` là tham số chúng ta muốn đưa vào hàm thực thi ngắt, đôi khi chúng ta muốn khởi tạo các thông số ban đầu cho hàm thực thi ngắt, kế thừa những thông tin đã xử lý từ trước hay từ người dùng.
- `my_timer.function` là trường chứa con trỏ của hàm phục vụ ngắt đã được khởi tạo và định nghĩa trước đó.

d. Bước 4: Kích hoạt timer hoạt động trong *kernel*:

Chúng ta thực thi lệnh sau:

```
add_timer (&my_timer);
```

Tham số của hàm add_timer là con trỏ của timer được khởi tạo và gán các tham số cần thiết.

Khi *timer* được kích hoạt, hàm thực thi ngắt hoạt động, nó sẽ bị vô hiệu hóa trong chu kỳ tiếp theo. Muốn *timer* tiếp tục hoạt động, chúng ta phải kích hoạt lại bằng câu lệnh sau:

```
/*Kích hoạt timer hoạt động lại cho chu kỳ ngắt tiếp theo*/
mod_timer (&my_timer, jiffies + new_delay);
```

Nếu muốn xóa timer khỏi hệ thống chúng ta sùng lệnh sau:

```
/*Xóa timer khỏi hệ thống*/
del_timer (&my_timer);
```

Tuy nhiên lệnh này chỉ thực hiện thành công khi *timer* không còn hoạt động, nghĩa là khi *timer* còn đang chờ chu kỳ ngắt tiếp theo dùng lệnh `del_timer()` sẽ không hiệu quả. Muốn khắc phục lỗi này, chúng ta phải dùng `lệnh del_timer_sync()`, lệnh này sẽ chờ cho đến khi *timer* hoàn thành chu kỳ ngắt gần nhất mới xóa *timer* đó.

3. Ví dụ:

Đoạn chương trình sau sẽ định thời xuất thông tin ra màn hình hiển thị vời chu kỳ là 1s. Mỗi lần xuất sẽ thay đổi thông tin, thông báo những lần xuất thông tin là khác nhau;

```
/*Khai báo biến cục bộ lưu giá trị muốn xuất ra màn hình, giá trị ban đầu bằng 0*/
```


CuuDuongThanCong.com https://fb.com/tailieudientucntt

```
int counter = 0;
/*Khai báo biến timer phục vụ ngắt*/
struct time_list my_timer;
/*Khai báo định nghĩa hàm phục vụ ngắt*/
void timer_isr (unsigned long data) {
/*In thông báo cho người dùng*/
printk ("Driver: Hello, the counter's value is %d\n",
counter++);
    /*Cài đặt lại giá trị timer cho lần hoạt động tiếp theo, cài đặt chu kỳ 1 giây*/
mod_timer (&my_timer, jiffies + HZ);
}
/*Thực hiện khởi tạo timer trong khi cài đặt driver vào hệ thống, trong hàm init()*/
static int
__init my_driver_init (void) {
/*Các lệnh khởi tạo khác*/
...
/*Khởi tạo timer hoạt động ngắt*/
/*Khởi tạo timer đã được khai báo*/
init_timer (&my_timer);
/*Cài đặt các thông số cho timer*/
my_timer.expires =  jiffies + HZ; //Khởi tạo trì hoãn ban đầu là 1s;
my_timer.data = 0; //Dữ liệu truyền cho hàm ngắt là 0;
my_timer.function = my_function; //Gán hàm thực thi ngắt cho timer.
/*Kích hoạt timer hoạt động trong hệ thống*/
add_timer (&my_timer);
...
}
```

VI. Trì hoãn thời gian trong kernel:

Trì hoãn thời gian là một trong những vấn đề quan trọng trong lập trình *driver* cũng như trong lập trình *application*. Trong phần trước chúng ta đã tìm hiểu những lệnh trì

CuuDuongThanCong.com https://fb.com/tailieudientucntt

hoãn thời gian từ khoảng thời gian nhỏ tính theo nano giây đến khoảng thời gian lớn hơn tính bằng giây. Tùy thuộc vào yêu cầu chính xác cao hay thấp mà chúng ta áp dụng kỹ thuật trì hoãn thời gian cho phù hợp. *Kernel* cũng hỗ trợ các kỹ thuật trì hoãn khác nhau tùy theo yêu cầu mà áp dụng kỹ thuật nào phù hợp nhất sao cho không ảnh hưởng đến hoạt động của hệ thống.

1. Vòng lặp vô tận:

Kỹ thuật trì hoãn thời gian đầu tiên là vòng lặp *busy loop*. Đây là cách trì hoãn thời gian cổ điển và đơn giản nhất, áp dụng cho tất cả các hệ thống. Kỹ thuật trì hoãn này có dạng như sau:

```
/*Khai báo thời điểm tương lai  muốn thực hiện trì hoãn*/
unsigned long timeout = jiffies + HZ/10; //Trì hoãn 10ms;
/*Thực hiện vòng lặp vô tận while () trì hoãn thời gian*/
while (time_before(jiffies, timeout))
   ;
```

cuu duong than cong . com

Với cách trì hoãn thời gian trên, chúng ta dùng biến *jiffies* để so sánh với thời điểm tương lai làm điều kiện cho lệnh `while ()` thực hiện vòng lặp. Như vậy chúng ta chỉ có thể trì hoãn một khoảng thời gian đúng bằng một số nguyên lần của *tick*. Hơn nữa, khác với lớp *user*, vòng lặp trong *kernel* không được chia tiến trình thực hiện. Vì thế vòng lặp vô tận trong kernel sẽ chiếm hết thời gian làm việc của CPU và như thế các hoạt động khác sẽ không được thực thi, hệ thống sẽ bị ngưng lại tạm thời. Điều này rất nguy hiểm cho các ứng dụng đòi hỏi độ tin cậy cao về thời gian thực. Cách trì hoãn thời gian này rất hiếm khi được sử dụng trong những hệ thống lớn.

Để giải quyết vấn đề này, người ta dùng kỹ thuật chia tiến trình trong lúc thực hiện trì hoãn như sau:

cuu duong than cong . com

```
while (time_before(jiffies, timeout))
   schedule(); //Hàm này chứa trong thư viện <linux/sched.h>
```

Trong khi *jiffies* vẫn thõa mãn điều kiện nhỏ hơn thời điểm đặt trước, *kernel* sẽ chia thời gian thực hiện công việc khác trong hệ thống. Cho đến khi vòng lặp được thoát, những lệnh tiếp theo sẽ tiếp tục thực thi.

CuuDuongThanCong.com https://fb.com/tailieudientucntt

Chúng ta cũng có thể áp dụng những lệnh so sánh thời gian thực trong phần trước để thực hiện trì hoãn thời gian với độ chính xác cao hơn.

2. Trì hoãn thời gian bằng những hàm hỗ trợ sẵn:

Linux kernel cũng cung cấp cho chúng ta những hàm thực hiện trì hoãn thời gian với độ chính xác cao, thích hợp cho những khoảng thời gian nhỏ. Đó là những hàm:

- `void udelay(unsigned long usec);` Hàm dùng để trì hoãn thời gian có độ phân giải micrô giây;
- `void ndelay(unsigned long nsec);` Hàm trì hoãn thời gian có độ phân giải nanô giây;
- `void mdelay(unsigned long msec);` Hàm trì hoãn thời gian có độ phân giải mili giây;

***Các hàm trì hoãn thời gian này chỉ thích hợp cho những khoảng thời gian nhỏ hơn 1 tick trong kernel. Nếu lơn hơn sẽ làm ảnh hưởng đến hoạt động của cả hệ thống vì bản chất đây vẫn là những vòng lặp vô tận, chiếm thời gian hoạt động của CPU.*

3. Trì hoãn thời gian bằng hàm schedule_timeout ():

Kỹ thuật này khác với hai kỹ thuật trên, dùng hàm `schedule_timeout()` sẽ làm cho chương trình driver dừng lại tại thời điểm khai báo, rơi vào trạng thái ngủ trong suốt thời gian trì hoãn. Thời gian trì hoãn do người lập trình cài đặt. Để sử dụng hàm này, ta tiến hành các bước sau:

*/*Cài đặt chương trình driver vào trạng thái ngủ*/*

```
set_current_state (TASK_INTERRUPTIBLE);
```

*/*Cài đặt thời gian cho tín hiệu đánh thức chương trình*/*

```
schedule_timeout (unsigned long time_interval);
```

***Trong đó time_interval là khoảng thời gian tính bằng tick muốn cài đặt tín hiệu đánh thức chương trình đang trong trạng thái ngủ.*

VII. Kết luận:

Trong bài này chúng ta đã tìm hiểu rõ về cách quản lý thời gian trong *kernel*, thế nào là *jiffies, HZ, ...* vai trò ý nghĩa của chúng trong duy trì quản thời gian thực cũng như trong trì hoãn thời gian.

CuuDuongThanCong.com https://fb.com/tailieudientucntt

Chúng ta cũng đã tìm hiểu các hàm thao tác với thời gian thực trong *kernel*, với những hàm này chúng ta có thể xây dựng các ứng dụng có liên quan đến thời gian thực trong hệ thống.

Có nhiều các khác nhau để thực hiện trì hoãn trong *kernel* tương tự như trong *user*. Nhưng chúng ta phải biết cách chọn phương pháp phù hợp để không làm ảnh hưởng đến hoạt động của toàn hệ thống.

Đến đây chúng ta có thể bước vào các bài thực hành, viết *driver* và ứng dụng cho một số phần cứng cơ bản trong những bài sau. Hoàn thành những bài này sẽ giúp cho chúng ta có được cái nhìn thực tế hơn về lập trình hệ thống nhúng.

cuu duong than cong . com

cuu duong than cong . com

CuuDuongThanCong.com https://fb.com/tailieudientucntt

GIAO TIẾP ĐIỀU KHIỂN

LED ĐƠN

Bài 1: điều khiển LED đơn trên kit KM9260

a. Kết nối phần cứng:

Thực hiện kết nối phần cứng theo sơ đồ sau trên board SAM9260-EK điều khiển led nối với chân PB20, PB21, PB22, PB23

b. Chương trình driver: *Có tên 1_1_single_led_dev.c*

(Mã nguồn và giải thích chương trình driver được chứa trong CD đính kèm)

c. Chương trình application: *Có tên 1_1_single_led_app.c*

(Mã nguồn và giải thích chương trình application chứa trong CD đính kèm)

d. Biên dịch và thực thi dự án:

- *Biên dịch driver:*

Trong thư mục chứa tập tin mã nguồn drive, tạo tập tin `Makefile` có nội dung sau:

```
export ARCH=arm
export CROSS_COMPILE=arm-none-linux-gnueabi-
obj-m += 1_1_single_led_dev.o
all:
/*Lưu ý phải đúng đường dẫn đến cấu trúc mã nguồn kernel*/
make -C /home/arm/project/kernel/linux-2.6.30 M=$(PWD) modules
clean:
/*Lưu ý phải đúng đường dẫn đến cấu trúc mã nguồn kernel*/
make -C /home/arm/project/kernel/linux-2.6.30 M=$(PWD) clean
```

Biên dịch *driver* bằng lệnh *shell* như sau:

```
make clean all
```

***lúc này tập tin chương trình driver được tạo thành với tên* `1_1_single_led_dev.ko`

- Biên dich *application*: Bằng lệnh *shell* sau:

```
./arm-none-linux-gnueabi-gcc 1_1_single_led_app.c -o
1_1_single_led_app
```


CuuDuongThanCong.com https://fb.com/tailieudientucntt

***Chương trình được biên dịch có tên là 1_1_single_led_app*

- Thực thi chương trình:

➢ Chép *driver* và *application* vào kit;

➢ Cài đặt *driver* bằng lệnh: `insmod 1_1_single_led_dev.ko`

➢ Thay đổi quyền thực thi cho chương trình *application* bằng lệnh:

```
chmod 777 1_1_single_led_app
```

➢ Chạy chương trình và quan sát kết quả:

- Khai báo chân PC0 là ngõ ra:

```
./1_1_single_led_app dirout 96
Using gpio pin 96
```

(Lúc này led kết nối với PC0 tắt)

- Xuất dữ liệu mức cao cho PC0:

```
./1_1_single_led_app set 96 1
Using gpio pin 96
```

(Lúc này ta thấy led nối với chân PC0 sáng lên)

- Xuất dữ liệu mức thấp cho PC0:

```
./1_1_single_led_app set 96 0
Using gpio pin 96
```

(Lúc này ta thấy led nối với chân PC0 tắt xuống)

- Khai báo chân PA23 là ngõ vào:

```
./1_1_single_led_app dirin 55
Using gpio pin 55
```

***Khi công tắc nối với PA23 ở vị trí ON, chân PA23 nối xuống mass;*

- Lấy dữ liệu vào từ chân PA23

```
./1_1_single_led_app get 55
Using gpio pin 55
Pin 55 is LOW
```

*** Khi công tắc nối với PA23 ở vị trí OFF, chân PA23 nối lên VCC;*

- Lấy dữ liệu vào từ chân PA23

```
./1_1_single_led_app get 55
```


CuuDuongThanCong.com https://fb.com/tailieudientucntt

```
Using gpio pin 55
Pin 55 is HIGH
```

Bài 2:

ĐIỀU KHIỂN SÁNG TẮT 1 LED

I. Phác thảo dự án:

Đây là dự án đầu tiên căn bản nhất trong quá trình lập trình điều khiển các thiết bị phần cứng. Người học có thể làm quen với việc điều khiển các chân *gpio* cho các mục đích khác nhau: truy xuất dữ liệu, cài đặt thông số đối với một chân vào ra trong vi điều khiển thông qua *driver* và chương trình ứng dụng. Để hoàn thành được bài này, người học phải có những kiến thức và kỹ năng sau:

- Kiến thức về mối quan hệ giữa *driver* và *application* trong hệ thống nhúng, cũng như việc trao đổi thông tin qua lại dựa vào các giao diện chuẩn;
- Kiến thức về giao diện chuẩn *ioctl* trong giao tiếp giữa *driver* (trong *kernel*) và *application* (trong *user*);
- Kiến thức về *gpio* trong *linux kernel*;
- Lập trình chương trình ứng dụng có sử dụng kỹ thuật hàm *main* có nhiều tham số giao tiếp với người dùng;
- Biên dịch và cài đặt được *driver*, *application* nạp vào hệ thống và thực thi;

***Tất cả những kiến thức yêu cầu nêu trên đều đã được chúng tôi trình bày kỹ trong những phần trước. Nếu cần người học có thể quay lại tìm hiểu để bước vào nội dung này hiệu quả hơn.*

a. Yêu cầu dự án:

Dự án này có yêu cầu là điều khiển thành công 1 led đơn thông qua *driver* và *application*. Người dùng có thể điều khiển led sáng tắt và đọc về trạng thái của một chân gpio theo yêu cầu nhập từ dòng lệnh *shell*.

- Đầu tiên, người dùng xác định chế độ vào ra cho chân *gpio* muốn điều khiển.

CuuDuongThanCong.com https://fb.com/tailieudientucntt

- Tiếp theo, nếu là chế độ ngõ vào thì sẽ xuất thông tin ra màn hình hiển thị cho biết trạng thái của chân *gpio* là mức thấp hay mức cao. Nếu là chế độ ngõ ra thì người dùng sẽ nhập thông tin *high* hoặc *low* để điều khiển led sáng tắt theo yêu cầu.
  **Lưu ý, nếu ngõ vào thì người dùng nên kết nối chân *gpio* với một công tắc điều khiển ON-OFF, nếu ngõ ra thì người dùng nên kết nối chân *gpio* với một LED đơn theo kiểu tích cực mức cao.

b. Phân công nhiệm vụ:

- ***Driver:*** có tên `1_1_single_led_dev.c`

Sử dụng kỹ thuật giao diện *ioctl* để nhận lệnh và tham số từ *user application* thực thi điều khiển chân *gpio* theo yêu cầu. *ioctl* có 5 tham số lệnh tương ứng với 5 khả năng mà *driver* có thể phục vụ cho *application*:

- `GPIO_DIR_IN`: Cài đặt chân *gpio* là ngõ vào;
- `GPIO_DIR_OUT`: Cài đặt chân *gpio* là ngõ ra;
- `GPIO_GET`: Lấy dữ liệu mức logic từ chân gpio ngõ vào trả về một biến của *user application*;
- `GPIO_SET`: Xuất dữ liệu cho chân *gpio* ngõ ra theo thông tin lấy từ một biến trong *user application* tương ứng sẽ là mức thấp hay mức cao;

- ***Application***: có tên `1_1_single_led_app.c`

Sử dụng kỹ thuật lập trình hàm *main* có nhiều tham số lựa chọn cho người dùng khả năng điều khiển trên màn hình *shell* trong quá trình thực thi chương trình ứng dụng. Theo đó, chương trình ứng dụng `1_1_single_led_app` có những thao tác lệnh sau:

- Đầu tiên người dùng nhập tên chương trình cùng với các tham số mong muốn tương ứng với từng lệnh muốn thực thi.
- Nếu là lệnh `dirin`, người dùng phải cung cấp cho *driver* tham số tiếp theo là số chân *gpio* muốn cài đặt chế độ ngõ vào;
- Nếu là lệnh `dirout`, người dùng phải cung cấp cho *driver* tham số tiếp theo là số chân *gpio* muốn cài đặt chế độ ngõ ra;

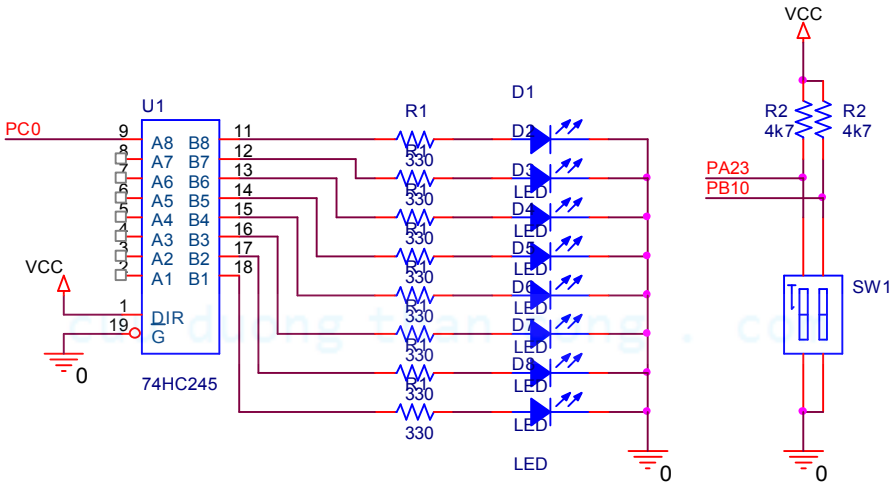
CuuDuongThanCong.com https://fb.com/tailieudientucntt

- Nếu là lệnh `set`, thông tin tiếp theo phải cung cấp là 1 hoặc 0 và chân *gpio* muốn xuất dữ liệu;
- Nếu là lệnh `get`, thông tin tiếp theo người dùng phải cung cấp là số chân *gpio* muốn lấy dữ liệu. Sau khi lấy dữ liệu, xuất ra màn hình hiển thị thông báo cho người dùng biết.

II. Thực hiện:

e. Kết nối phần cứng:

Thực hiện kết nối phần cứng theo sơ đồ sau:



Hình 4-1- Sơ đồ kết nối LED đơn và công tắc điều khiển

f. Chương trình driver: *Có tên 1_1_single_led_dev.c*

(Mã nguồn và giải thích chương trình driver được chứa trong CD đính kèm)

g. Chương trình application: *Có tên 1_1_single_led_app.c*

(Mã nguồn và giải thích chương trình application chứa trong CD đính kèm)

h. Biên dịch và thực thi dự án:

- *Biên dịch driver:*

Trong thư mục chứa tập tin mã nguồn drive, tạo tập tin `Makefile` có nội dung sau:

```
export ARCH=arm
export CROSS_COMPILE=arm-none-linux-gnueabi-
obj-m += 1_1_single_led_dev.o
all:
```


CuuDuongThanCong.com https://fb.com/tailieudientucntt

*/*Lưu ý phải đúng đường dẫn đến cấu trúc mã nguồn kernel*/*

```
make -C /home/arm/project/kernel/linux-2.6.30 M=$(PWD) modules
clean:
```

*/*Lưu ý phải đúng đường dẫn đến cấu trúc mã nguồn kernel*/*

```
make -C /home/arm/project/kernel/linux-2.6.30 M=$(PWD) clean
```

Biên dịch *driver* bằng lệnh *shell* như sau:

```
make clean all
```

***lúc này tập tin chương trình driver được tạo thành với tên 1_1_single_led_dev.ko*

- Biên dich *application*: Bằng lệnh *shell* sau:

```
./arm-none-linux-gnueabi-gcc 1_1_single_led_app.c -o
1_1_single_led_app
```

***Chương trình được biên dịch có tên là 1_1_single_led_app*

- Thực thi chương trình:

➢ Chép *driver* và *application* vào kit;

➢ Cài đặt *driver* bằng lệnh: `insmod 1_1_single_led_dev.ko`

➢ Thay đổi quyền thực thi cho chương trình *application* bằng lệnh:

```
chmod 777 1_1_single_led_app
```

➢ Chạy chương trình và quan sát kết quả:

- Khai báo chân PC0 là ngõ ra:

```
./1_1_single_led_app dirout 96
Using gpio pin 96
```

(Lúc này led kết nối với PC0 tắt)

- Xuất dữ liệu mức cao cho PC0:

```
./1_1_single_led_app set 96 1
Using gpio pin 96
```

(Lúc này ta thấy led nối với chân PC0 sáng lên)

- Xuất dữ liệu mức thấp cho PC0:

```
./1_1_single_led_app set 96 0
Using gpio pin 96
```


CuuDuongThanCong.com https://fb.com/tailieudientucntt

(Lúc này ta thấy led nối với chân PC0 tắt xuống)

- Khai báo chân PA23 là ngõ vào:

```
./1_1_single_led_app dirin 55
Using gpio pin 55
```

***Khi công tắc nối với PA23 ở vị trí ON, chân PA23 nối xuống mass;*

- Lấy dữ liệu vào từ chân PA23

```
./1_1_single_led_app get 55
Using gpio pin 55
Pin 55 is LOW
```

*** Khi công tắc nối với PA23 ở vị trí OFF, chân PA23 nối lên VCC;*

- Lấy dữ liệu vào từ chân PA23

```
./1_1_single_led_app get 55
Using gpio pin 55
Pin 55 is HIGH
```

***Tương tự cho các chân gpio khác;*

cuu duong than cong . com

cuu duong than cong . com

CuuDuongThanCong.com https://fb.com/tailieudientucntt

Bài 2:

ĐIỀU KHIỂN SÁNG TẮT 8 LED

I. Phác thảo dự án:

Dự án này chủ yếu truy xuất chân *gpio* theo chế độ ngõ ra, nhưng điểm khác biệt so với dự án trước là không điều khiển riêng lẻ từng bit mà công việc điều khiển này sẽ do *driver* thực hiện. Phần này sẽ cho chúng ta làm quen với cách điều khiển thông tin theo từng *port 8 bits*. Để việc tiếp thu đạt hiệu quả cao nhất, trước khi nghiên cứu người học phải có những kiến thức và kỹ năng sau:

- Kiến thức tổng quát về mối quan hệ giữa *driver* và *application* trong hệ thống nhúng, cũng như việc trao đổi thông tin qua lại dựa vào các giao diện chuẩn;
- Kiến thức về giao diện chuẩn *write* trong giao tiếp giữa *driver* (trong *kernel*) và *application* (trong *user*);
- Kiến thức về *gpio* trong *linux kernel*;
- Lập trình chương trình ứng dụng có sử dụng kỹ thuật hàm *main* có nhiều tham số giao tiếp với người dùng;
- Biên dịch và cài đặt được *driver*, *application* nạp vào hệ thống và thực thi;

***Tất cả những kiến thức yêu cầu nêu trên đều đã được chúng tôi trình bày kỹ trong những phần trước. Nếu cần người học có thể quay lại tìm hiểu để bước vào nội dung này hiệu quả hơn.*

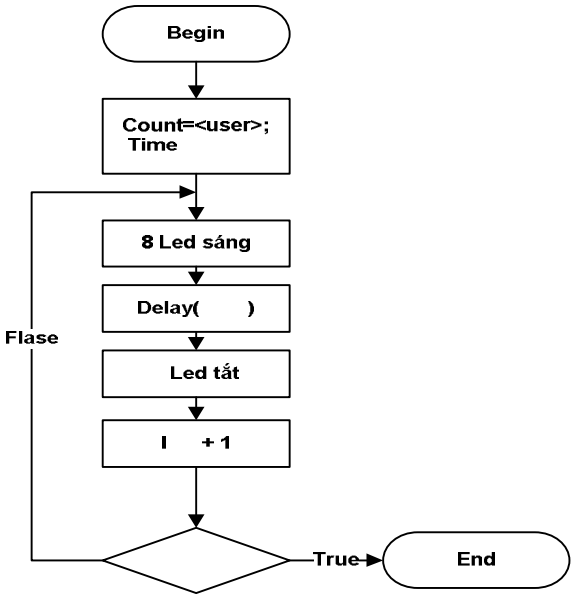
a. Yêu cầu dự án:

Yêu cầu của dự án là điều khiển thành công *1 port 8 leds* hoạt động chớp tắt cùng lúc theo chù kỳ và số lần được nhập từ người dùng trong lúc gọi chương trình thực thi. Khi hết nhiệm vụ chương trình sẽ được thoát và chờ lần gọi thực thi tiếp theo.

- Đầu tiên người dùng gọi chương trình *driver*, cung cấp thông tin về thời gian của chu kỳ và số lần nhấp nháy mong muốn;
- Chương trình *application* nhận dữ liệu từ người dùng, tiến hành điều khiển *driver* tác động vào ngõ ra gpio làm led sáng tắt theo yều cầu;

CuuDuongThanCong.com https://fb.com/tailieudientucntt

- Lưu đồ điều khiển như sau:



Hình 4-2- Lưu đồ điều khiển LED sáng tắt theo số chu kỳ được quy định

b. Phân công nhiệm vụ:

- ***Driver:*** Có tên là `1_2_port_led_dev.c`

Driver sử dụng giao diện `write()` nhận dữ liệu từ *user application* xuất ra led tương ứng với dữ liệu nhận được. Dữ liệu nhận từ *user application* là một số char có *8 bits*. Mỗi bit tương ứng với 1 led cần điều khiển. Nhiệm vụ của *driver* là so sánh tương ứng từng bit trong số `char` này để quyết định xuất mức cao hay mức thấp cho led ngoại vi. Công việc của *driver* được thực hiện tuần tự như sau:

- Yêu cầu cài đặt các chân ngoại vi là ngõ ra, kéo lên. Công việc này được thực hiện khi thực hiện lệnh cài đặt *driver* vào hệ thống linux;
- Trong giao diện hàm `write()` (nhận dữ liệu từ *user*) thực hiện xuất ra mức cao hoặc mức thấp cho gpio điều khiển led.
- Giải phóng các chân gpio đã được khai báo khi không cần sử dụng, công việc này được thực hiện ngay trước khi tháo bỏ *driver* ra khỏi hệ thống.

- ***Application:*** Có tên là `1_2_port_led_app.c`

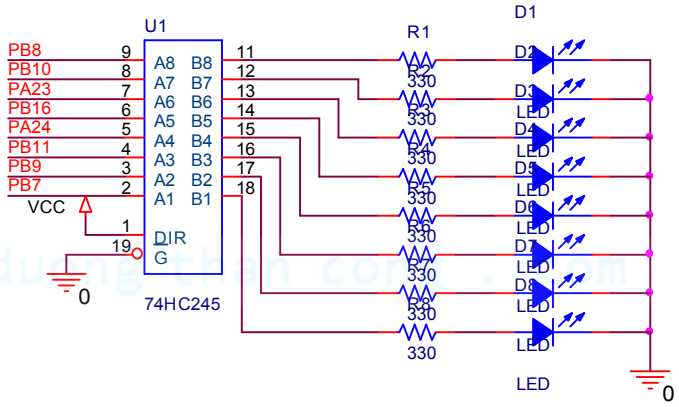
CuuDuongThanCong.com https://fb.com/tailieudientucntt

Thực hiện khai báo hàm `main` theo cấu trúc tham số để đáp ứng các yêu cầu khác nhau từ người dùng. Chương trình *application* có hai tham số: Tham số thứ nhất là thời gian tính bằng giây của chu kỳ chớp tắt, tham số thứ hai là số chu kỳ muốn chớp tắt.

Bên cạnh đó, phần này còn lập trình thêm một số chương trình tạo hiệu ứng điều khiển led khác như: 8 led sáng dần tắt dần (Trái qua phải, phải qua trái, ...). Các chức năng này được tổng hợp trong một chương trình *application* duy nhất, người sử dụng sẽ lựa chọn hiệu ứng thông qua các tham số người dùng của hàm main.

II. Thực hiện:

a. Kết nối phần cứng: Các bạn thực hiện kết nối phần cứng theo sơ đồ sau:



Hình 4-3- Sơ đồ kết nối 8 LEDs đơn.

***Lưu ý phải đúng số chân đã quy ước.*

b. Chương trình driver: *Chương trình có tên là 1_2_port_led_dev.c*

(Mã nguồn và giải thích chương trình được chứa trong CD đính kèm)

c. Chương trình application: *Có tên là `1_2_port_led_app.c`*

(Mã nguồn và giải thích chương trình được chứa trong CD đính kèm)

d. Biên dịch và thực thi dự án:

- ***Biên dịch driver:***

Tạo tập tin `Makefile` trong cùng thư mục với *driver*. Có nội dung sau:

```
export ARCH=arm
export CROSS_COMPILE=arm-none-linux-gnueabi-
obj-m += 1_2_port_led_dev.o
```


CuuDuongThanCong.com https://fb.com/tailieudientucntt

```
all:
/*Lưu ý phải đúng đường dẫn đến cấu trúc mã nguồn kernel*/
make -C /home/arm/project/kernel/linux-2.6.30 M=$(PWD) modules
clean:
/*Lưu ý phải đúng đường dẫn đến cấu trúc mã nguồn kernel*/
make -C /home/arm/project/kernel/linux-2.6.30 M=$(PWD) clean
```

- ***Biên dịch application:***

Trỏ vào thư mục chứa tập tin chương trình, biên dịch chương trình ứng dụng với lệnh sau:

```
arm-none-linux-gnueabi-gcc 1_2_port_led_app.c -o 1_2_port_led_app
```

***Chương trình biên dịch thành công có tên là: 1_2_port_led_app*

- ***Thực thi chương trình:***

Chép *driver* và chương trình vào kit, thực thi và kiểm tra kết quả;

- Cài đặt *driver* vào kit theo lệnh sau:

  ```
  insmod 1_2_port_led_dev.ko
  ```

- Thay đổi quyền thực thi cho chương trình ứng dụng:

  ```
  chmod 777 1_2_port_led_app
  ```

- Thực thi và kiểm tra kết quả:

  ```
  ./1_2_port_led_app 1 10
  ```

  ***Chúng ta thấy 8 led nhấp nháy 10 lần với chu kỳ 1s. Các bạn thay đổi chu kỳ và số lần nhấp nháy quan sát kết quả.*

cuu duong than cong . com

CuuDuongThanCong.com https://fb.com/tailieudientucntt