BỘ GIÁO DỤC VÀ ĐÀO TẠO
TRƯỜNG ĐẠI HỌC DÂN LẬP HẢI PHÒNG

ISO 9001:2008

ĐỒ ÁN TỐT NGHIỆP

NGÀNH: ĐIỆN TỬ VIỄN THÔNG

Người hướng dẫn : CN. Nguyễn Huy Dũng
Sinh viên : Lê Quốc Thiên

HẢI PHÒNG – 2013

BỘ GIÁO DỤC VÀ ĐÀO TẠO
TRƯỜNG ĐẠI HỌC DÂN LẬP HẢI PHÒNG

LẬP TRÌNH NHÚNG ARM TRÊN LINUX

ĐỒ ÁN TỐT NGHIỆP ĐẠI HỌC HỆ CHÍNH QUY
NGÀNH: ĐIỆN TỬ VIỄN THÔNG

Người hướng dẫn : CN. Nguyễn Huy Dũng
Sinh viên : Lê Quốc Thiên

HẢI PHÒNG – 2013

BỘ GIÁO DỤC VÀ ĐÀO TẠO
TRƯỜNG ĐẠI HỌC DÂN LẬP HẢI PHÒNG

NHIỆM VỤ ĐỀ TÀI TỐT NGHIỆP

Sinh viên : Lê Quốc Thiên. Mã SV: 1351030018.

Lớp : ĐT 1301 Ngành: Điện tử viễn thông.

Tên đề tài: Lập trình nhúng ARM trên Linux

NHIỆM VỤ ĐỀ TÀI

1. Nội dung và các yêu cầu cần giải quyết trong nhiệm vụ đề tài tốt nghiệp (về lý luận, thực tiễn, các số liệu cần tính toán và các bản vẽ).

.....

.....

.....

.....

.....

.....

.....

.....

.....

2. Các số liệu cần thiết để thiết kế, tính toán.

.....

.....

.....

.....

.....

.....

.....

.....

.....

3. Địa điểm thực tập tốt nghiệp.

.....

.....

.....

.....

.....

CÁN BỘ HƯỚNG DẪN ĐỀ TÀI TỐT NGHIỆP

Người hướng dẫn thứ nhất:

Họ và tên: Nguyễn Huy Dũng.

Học hàm, học vị: Cử nhân.

Cơ quan công tác: Trường Đại học Dân lập Hải Phòng.

Nội dung hướng dẫn:.....

.....

.....

.....

Người hướng dẫn thứ hai:

Họ và tên: .....

Học hàm, học vị: .....

Cơ quan công tác:.....

Nội dung hướng dẫn:.....

.....

.....

.....

Đề tài tốt nghiệp được giao ngày…….tháng…….năm 2013

Yêu cầu phải hoàn thành xong trước ngày…….tháng…….năm 2013

Đã nhận nhiệm vụ ĐTTN	Đã giao nhiệm vụ ĐTTN
Sinh viên	*Người hướng dẫn*

Hải Phòng, ngày ..... tháng.....năm 2013

Hiệu trưởng

GS.TS.NGƯT *Trần Hữu Nghị*

PHẦN NHẬN XÉT CỦA CÁN BỘ HƯỚNG DẪN

1. **Tinh thần thái độ của sinh viên trong quá trình làm đề tài tốt nghiệp:**

.....

.....

.....

.....

.....

.....

.....

.....

2. **Đánh giá chất lượng của khóa luận (so với nội dung yêu cầu đã đề ra trong nhiệm vụ Đ.T. T.N trên các mặt lý luận, thực tiễn, tính toán số liệu…):**

.....

.....

.....

.....

.....

.....

.....

.....

3. **Cho điểm của cán bộ hướng dẫn (ghi bằng cả số và chữ):**

.....

.....

.....

Hải Phòng, ngày……tháng……năm 2013

Cán bộ hướng dẫn

PHẦN NHẬN XÉT TÓM TẮT CỦA NGƯỜI CHẤM PHẢN BIỆN

1. Đánh giá chất lượng đề tài tốt nghiệp về các mặt thu thập và phân tích số liệu ban đầu, cơ sở lý luận chọn phương án tối ưu, cách tính toán chất lượng thuyết minh và bản vẽ, giá trị lý luận và thực tiễn đề tài.

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

2. Cho điểm của cán bộ phản biện (Điểm ghi cả số và chữ).

.....

.....

.....

.....

.....

.....

.....

.....

Hải Phòng, ngày……tháng……năm 2013

Người chấm phản biện

MỤC LỤC

LỜI CẢM ƠN
LỜI MỞ ĐẦU 1
CHƯƠNG 1: TỔNG QUAN VỀ HỆ THỐNG NHÚNG 3
1.1. Khái niệm về hệ thống nhúng 3
1.2. Bộ xử lý hệ thống nhúng 5
1.2.1. Kiến trúc CPU 5
1.2.2. Thiết bị ngoại vi 5
1.2.3. Công cụ phát triển 6
1.2.4. Độ tin cậy 6
1.2.5. Các kiến trúc phần mềm hệ thống nhúng 8
1.2.6. Hệ thống thời gian thực 8
1.2.7. Hệ điều hành thời gian thực (RTOS) và kernel thời gian thực 9
1.2.8. Chương trình, tác vụ và luồng 9
1.2.9. Kiến trúc của hệ thống thời gian thực 10
1.3. Phát triển ứng dụng nhúng 10
CHƯƠNG 2: VI XỬ LÝ ARM 14
2.1. Tổng quan 14
2.2. Cơ chế Pipeline 15
2.3. Các thanh ghi 15
2.4. Thanh ghi trạng thái chương trình hiện hành 16
2.5. Các mode ngoại lệ 17
2.6. Tập lệnh ARM7 19
2.6.1. Các lệnh rẽ nhánh 20
2.6.2. Các lệnh xử lý dữ liệu 21
2.6.3. Các lệnh truyền dữ liệu 22
2.6.4. Lệnh SWAP 23
2.7. Ngắt mềm (SWI – Software Interput instruction) 23
2.8. Đơn vị MAC (Multíply Accumulate Unit (MAC) 23
2.9. Tập lệnh THUMB 24
2.10. Cổng JTAG 26
2.11. Memory Acelerator Module (MAM) 27
2.12. PLL- Phase Locked Loop 29
2.13. Bộ chia bus (VLSI Peripheral Bus Divider) 31

CHƯƠNG 3: HỆ ĐIỀU HÀNH NHÚNG EMBEDĐE LINUX 33
3.1. Giới thiệu hệ điều hành nhúng 33
3.1.1. Hệ điều hành 33
3.1.2. Hệ điều hành nhúng 34
3.2. Các hệ điều hành nhúng điển hình 34
3.2.1. Embedded Linux 34
3.2.2. Windows CE 36
3.2.3. Andriod 37
3.3. Lập trình C/C++ trên Linux 39
3.3.1. Linux và các lệnh cơ bản 39
3.3.2. Chương trình trên Linux 43
3.3.3. Xử lý tiến trình trong linux 48
CHƯƠNG 4:LẬP TRÌNH NHÚNG ARM TRÊN LINUX 59
4.1. Giới thiệu KIT nhúng FriendlyArm Micro2440 59
4.2. Môi trường phát triển ứng dụng 61
4.3. Lập trình điều khiển LED 61
4.4. Lập trình đọc trạng thái nút bấm 63
KẾT LUẬN 67
TÀI LIỆU THAM KHẢO 68

LỜI CẢM ƠN

Trước hết, em xin gửi lời cảm ơn chân thành tới thầy giáo Nguyễn Huy Dũng đã tận tình chỉ bảo, hướng dẫn và giúp cho em có những kiến thức cũng như kinh nghiệm quý báu.

Em xin tỏ lòng biết ơn sâu sắc tới các thầy cô giáo trường Đại Học Dân Lập Hải Phòng và đặc biệt là các thầy cô giáo trong tổ bộ môn điện tử viễn thông đã luôn nhiệt tình giảng dạy và chỉ bảo chúng em trong suốt bốn năm học vừa qua.

Cuối cùng, xin cảm ơn gia đình, người thân và các bạn của tôi, những người đã luôn bên cạnh động viên, khích lệ và giúp đỡ tôi trong thời gian qua.

Mặc dù có nhiều cố gắng, song thời gian thực hiện đồ án có hạn, vốn kiến thức nắm được chưa nhiều nên đồ án còn nhiều hạn chế. Em rất mong nhận được nhiều sự góp ý, chỉ bảo của các thầy, cô để hoàn thiện hơn bài viết của mình.

Em xin chân thành cảm ơn!

Hải Phòng, tháng 6 năm 2013

Sinh viên thực hiện

Lê Quốc Thiên

LỜI MỞ ĐẦU

Thế giới ngày nay với khoa học kĩ thuật phát triển mạnh mẽ cuộc sống con người ngày càng được phát triển tốt hơn. Khoa học kỹ thuật đem lại nhiều tiện ích thiết thực hơn cho cuộc sống con người. Góp phần to lớn trong quá trình phát triển của khoa học kỹ thuật là sự phát triển mạnh mẽ của vi xử lý. Từ bộ vi xử lý đầu tiên Intel 4004 được sản xuất bởi công ty Intel vào năm 1971, đến nay ngành công nghiệp vi xử lý đã phát triển vượt bậc và đa dạng với nhiều loại như: 8951, PIC, AVR, ARM, Pentium,Core i7,….

Cùng với sự phát triển đa dạng về chủng loại thì tài nguyên của vi xử lý cũng được nâng cao. Các vi xử lý ngày nay cung cấp cho người dùng một nguồn tài nguyên rộng lớn và phong phú. Có thể đáp ứng được nhiều yêu cầu khác nhau trong thưc tế. Để giúp cho người dùng sử dụng hiệu quả và triệt để các tài nguyên này thì hệ thống nhúng ra đời.Hệ thống nhúng (Embedded system) là một thuật ngữ để chỉ một hệ thống có khả năng tự trị được nhúng vào trong một môi trường hay một hệ thống mẹ. Đó là các hệ thống tích hợp cả phần cứng và phần phềm phục vụ các bài toán chuyên dụng trong nhiều lĩnh vực công nghiệp, tự động hoá điều khiển, quan trắc và truyền tin. Với sự ra đời của hệ thống nhúng thì vi xử lý ngày càng được ứng dụng rộng rãi trong đời sống cũng như trong công nghiệp vì khả năng xử lý nhanh, đa dạng, tiết kiệm năng lượng và độ ổn định của hệ thống nhúng.

Tuy hệ thống nhúng rất phổ biến trên toàn thế giới và là hướng phát triển của ngành Điện tử sau này nhưng hiện nay ở Việt Nam độ ngũ kỹ sư hiểu biết về hệ thống nhúng còn rất hạn chế không đáp ứng được nhu cầu nhân lực trong lĩnh vực này.

Vì vậy việc biên soạn giáo trình về hệ thống nhúng là một yêu cầu cần thiết trong thời điểm hiện tại cũng như trong tương lai. Nhận thấy được nhu cầu cấp thiết đó nên sinh viên thực hiện đã chọn đề tài: "**LẬP TRÌNH NHÚNG ARM TRÊN LINUX"** để làm đồ án tốt nghiệp cho mình.

Với mục tiêu xác định như trên, đồ án được chia ra làm 3 phần với nội dung cơ bản như sau:

Chương 1: Tổng quan về hệ thống nhúng.

Chương 2: Vi xử lý ARM.

Chương 3: Hê điều hành nhúng Embedded Linux.

Chương 4: Lập trình nhúng ARM trên Linux.

Do thời gian thực hiện ngắn cộng với vốn kiến thức còn rất hạn chế nên đồ án chắc chắn còn nhiều thiếu sót, em rất mong nhận được sự chỉ bảo của các thầy cô để hoàn thiện hơn bài viết của mình.

CHƯƠNG 1:

TỔNG QUAN VỀ HỆ THỐNG NHÚNG

1.1. Khái niệm về hệ thống nhúng

Hệ thống nhúng (Embedded system) là một thuật ngữ để chỉ một hệ thống có khả năng tự trị được nhúng vào trong một môi trường hay một hệ thống mẹ. Đó là các hệ thống tích hợp cả phần cứng và phần mềm phục vụ các bài toán chuyên dụng trong nhiều lĩnh vực công nghiệp, tự động hoá điều khiển, quan trắc và truyền tin. Đặc điểm của các hệ thống nhúng là hoạt động ổn định và có tính năng tự động hoá cao.

Hệ thống nhúng thường được thiết kế để thực hiện một chức năng chuyên biệt nào đó. Khác với các máy tính đa chức năng, chẳng hạn như máy tính cá nhân, một hệ thống nhúng chỉ thực hiện một hoặc một vài chức năng nhất định, thường đi kèm với những yêu cầu cụ thể và bao gồm một số thiết bị máy móc và phần cứng chuyên dụng mà ta không tìm thấy trong một máy tính đa năng nói chung. Vì hệ thống chỉ được xây dựng cho một số nhiệm vụ nhất định nên các nhà thiết kế có thể tối ưu hóa nó nhằm giảm thiểu kích thước và chi phí sản xuất. Các hệ thống nhúng thường được sản xuất hàng loạt với số lượng lớn. Hệ thống nhúng rất đa dạng, phong phú về chủng loại. Đó có thể là những thiết bị cầm tay nhỏ gọn như đồng hồ kĩ thuật số và máy chơi nhạc MP3, hoặc những sản phẩm lớn như đèn giao thông, bộ kiểm soát trong nhà máy hoặc hệ thống kiểm soát các máy năng lượng hạt nhân. Xét về độ phức tạp, hệ thống nhúng có thể rất đơn giản với một vi điều khiển hoặc rất phức tạp với nhiều đơn vị, các thiết bị ngoại vi và mạng lưới được nằm gọn trong một lớp vỏ máy lớn.

Như vậy không phải tất cả các sản phẩm đo lường và điều khiển đều là các hệ nhúng. Hiện nay chúng ta còn gặp nhiều hệ thống điều khiển tự động hoạt động theo nguyên tắc cơ khí, thuỷ lực, khí nén, rơ le, hoặc diện tử tương tự…

Ngược lại phần lớn các sản phẩm cơ điện tử hiện nay đều có nhúng trong nó các chip vi xử lý hoặc một mạng nhúng. Ta biết rằng cơ điện tử là sự cộng năng của các công nghệ cơ khí, điện tử, điều khiển và công nghệ thông tin. Sự phối hợp đa ngành này tạo nên sự vượt trội của các sản phẩm cơ điện tử. Sản phẩm cơ điện tử ngày càng tinh sảo và ngày càng thông minh mà phần hồn của nó do các phần mềm nhúng trong nó tạo nên. Các sản phẩm cơ điện tử là các sản phẩm có ít nhất một quá trình cơ khí (thường là một quá trình chuyển động), là đối tượng để điều khiển do vậy các sản phẩm cơ điện tử ngày nay thường có các hệ nhúng trong nó nhưng ngược lại không phải hệ thống nhúng nào cũng là một hệ cơ điện tử.

Điểm qua sự phát triển của máy tính ta thấy nó đã trải qua 3 giai đoạn. Giai đoạn năm 1960-1980 là giai đoạn phát triển của máy tính lớn và máy mini (main frame và mini computer) với khoảng 1000 chip/máy và mỗi máy có khoảng 100 người dùng. Giai đoạn từ 1980-2000 là giai đoạn phát triển của máy PC với số chip vi xử lý khoảng 10 chip/máy và thông thường cho một người sử dụng. Thời đại hậu PC (Post-PC Era) là giai đoạn mà mọi đồ dùng đều có chip, trung bình 1 chip/một máy và số máy dùng cho một người lên đến >100 máy. Giai đoạn hậu PC được dự báo từ 2001-2010 khi các thiết bị xung quanh ta đều được thông minh hoá và kết nối với nhau thành mạng tạo thành môi trường thông minh phục vụ cho con người.

Điểm qua về chức năng xử lý tin ở PC và ở các thiết bị nhúng có những nét khác biệt. Đối với PC và mạng Internet chức năng xử lý đang được phát triển mạnh ở các lĩnh vực như thương mại điện tử, ngân hàng điện tử, chính phủ điện tử, thư viện điện tử, đào tạo từ xa, báo điện tử….Các ứng dụng này thường sử dụng máy PC để bàn, mạng WAN, LAN hoạt động trong thế giới ảo. Còn đối với các hệ nhúng thì chức năng xử lý tính toán được ứng dụng cụ thể cho các thiết bị vật lý (thế giới thật) như mobile phone, quần áo thông minh, các đồ điện tử cầm tay, thiết bị y tế, xe ô tô, tàu tốc hành, phương tiện vận tải thông minh, máy đo, đầu đo cơ cấu chấp hành thông minh, các hệ thống điều khiển, nhà thông minh, thiết bị gia dụng thông minh …

Hệ thống nhúngcó vai trò đảm nhận một phần công việc cụ thể của hệ thống mẹ. hệ thống nhúngcó thể là một hệ thống phần cứng và cũng có thể là một hệ thống phần mềm. Đặc điểm của hệ thống nhúnglà hoạt động ổn định và có tính năng tự động hoá cao. hệ thống nhúngđược thiết kế để thực hiện một chứa năng chuyên biệt nào đó. Khác với các máy tính đa năng, chẳng hạn như PC, một hệ thống nhúng chỉ thực hiện một hay một vài chức năng nhất định, thường đi kèm với những yêu cầu cụ thể và bao gồm một số thiết bị máy móc và phần cứng chuyên dụng mà ta không tìm thấy trong một máy tính đa năng nói chung. Vì hệ thống chỉ được xây dựng cho một số nhiệm vụ nhất định nên các nhà thiết kế có thể tối ưu hóa nó nhằm giảm thiểu kích thước và chi phí sản xuất. Các hệ thống nhúngthường được sản xuất hàng loạt với số lượng lớn. Hệ thống nhúngrất đa dạng, phong phú về chủng loại. Đó có thể là những thiết bị cầm tay nhỏ gọn như đồng hồ kĩ thuật số và máy chơi nhạc MP3, các thiết bị điện tử dân dụng (máy giặt, tủ lạnh, TV...), các thiết bị điện tử "thông minh" (điện thoại di động), thiết bị truyền thông, thiết bị y tế, xe hơi, thậm chí cả trong một máy tính cá nhân (card mở rộng), hoặc những sản phẩm lớn như đèn giao thông, bộ kiểm soát trong nhà máy hoặc hệ thống kiểm soát các máy năng lượng hạt nhân. Xét về độ phức tạp, hệ thống nhúng có thể rất đơn giản với một vi điều khiển hoặc rất phức tạp với nhiều đơn vị, các thiết bị ngoại vi và mạng lưới được nằm gọn trong một lớp vỏ máy lớn.

Các thiết bị PDA hoặc máy tính cầm tay cũng có một số đặc điểm tương tự với hệ thống nhúng như các hệ điều hành hoặc vi xử lý điều khiển chúng nhưng các thiết bị này không phải là hệ thống nhúng thật sự bởi chúng là các thiết bị đa năng, cho phép sử dụng nhiều ứng dụng và kết nối đến nhiều thiết bị ngoại vi.

Có rất nhiều hãng sản xuất bộ vi xử lý, phần cứng và phần mềm trong thị trường hệ thống nhúng và ứng với mỗi nhà sản xuất lại có nhiều dòng sản phẩm, phong phú về chủng loại và giá thành:

• Những bộ vi xử lý và phần cứng khác nhau: Texas Instrument, Freescale, ARM, Intel, Motorola, Atmel, AVR, Renesas…

• Những hệ điều hành khác nhau: QNX, uITRON, VxWorks, Windows CE/XP Embedded, Embedded Linux, Osek, Symbian…

• Những ngôn ngữ lập trình khác nhau: C/C++, B#, Ada, Assembly, PMC, LabView, PLC…

1.2. Bộ xử lý trong hệ nhúng

1.2.1 Kiến trúc CPU:

Các bộ xử lý trong hệ thống nhúng có thể được chia thành hai loại: vi xử lý và vi điều khiển. Các vi điều khiển thường có các thiết bị ngoại vi được tích hợp trên chip nhằm giảm kích thước của hệ thống. Có rất nhiều loại kiến trúc CPU được sử dụng trong thiết kế hệ nhúng như ARM, MIPS, Coldfire/68k, PowerPC, x86, PIC, 8051, Atmel AVR, Renesas H8, SH, V850, FR-V, M32R, Z80, Z8 … Điều này trái ngược với các loại máy tính để bàn, thường bị hạn chế với một vài kiến trúc máy tính nhất định. Các hệ thống nhúng có kích thước nhỏ và được thiết kế để hoạt động trong môi trường công nghiệp thường lựa chọn PC/104 và PC/104++ làm nền tảng. Những hệ thống này thường sử dụng DOS, Linux, NetBSD hoặc các hệ điều hành nhúng thời gian thực như QNX hay VxWorks. Còn các hệ thống nhúng có kích thước rất lớn thường sử dụng một cấu hình thông dụng là hệ thống on chip (System on a chip – SoC), một bảng mạch tích hợp cho một ứng dụng cụ thể (an application-specific integrated circuit – ASIC). Sau đó nhân CPU được mua và thêm vào như một phần của thiết kế chip. Một chiến lược tương tự là sử dụng FPGA (field-programmable gate array) và lập trình cho nó với những thành phần nguyên lý thiết kế bao gồm cả CPU.

1.2.2 Thiết bị ngoại vi:

HỆ THỐNG NHÚNG giao tiếp với bên ngoài thông qua các thiết bị ngoại vi

• Serial Communication Interfaces (SCI): RS-232, RS-422, RS-485...

• Synchronous Serial Communication Interface: I2C, JTAG, SPI, SSC và ESSI

• Universal Serial Bus (USB)

• Networks: Controller Area Network, LonWorks...

• Bộ định thời: PLL(s), Capture/Compare và Time Processing Units

• Discrete IO: General Purpose Input/Output (GPIO)

1.2.3 Công cụ phát triển:

Tương tự như các sản phẩm phần mềm khác, phần mềm HỆ THỐNG NHÚNG cũng được phát triển nhờ việc sử dụng các trình biên dịch (compilers), chương trình dịch hợp ngữ (assembler) hoặc các công cụ gỡ rối (debuggers). Tuy nhiên, các nhà thiết kế hệ thống nhúngcó thể sử dụng một số công cụ chuyên dụng như:

• Bộ gỡ rối mạch hoặc các chương trình mô phỏng (emulator)

• Tiện ích để thêm các giá trị checksum hoặc CRC vào chương trình, giúp hệ thống nhúng có thể kiểm tra tính hợp lệ của chương trình đó.

• Đối với các hệ thống xử lý tín hiệu số, người phát triển hệ thống có thể sử dụng phần mềm workbench như MathCad hoặc Mathematica để mô phỏng các phép toán.

• Các trình biên dịch và trình liên kết (linker) chuyên dụng được sử dụng để tối ưu hóa một thiết bị phần cứng.

• Một hệ thống nhúng có thể có ngôn ngữ lập trình và công cụ thiết kế riêng của nó hoặc sử dụng và cải tiến từ một ngôn ngữ đã có sẵn.

- Các công cụ phần mềm có thể được tạo ra bởi các công ty phần mềm chuyên dụng về hệ thống nhúng hoặc chuyển đổi từ các công cụ phát triển phần mềm GNU. Đôi khi, các công cụ phát triển dành cho PC cũng được sử dụng nếu bộ xử lý của hệ thống nhúngđó gần giống với bộ xử lý của một máy PC thông dụng.

1.2.4 Độ tin cậy:

Các hệ thống nhúngthường nằm trong các cỗ máy được kỳ vọng là sẽ chạy hàng năm trời liên tục mà không bị lỗi hoặc có thể khôi phục hệ thống khi gặp lỗi. Vì thế, các phần mềm hệ thống nhúngđược phát triển và kiểm thử một cách cẩn thận hơn là phần mềm cho PC. Ngoài ra, các thiết bị rời không đáng tin cậy như ổ đĩa, công tắc hoặc nút bấm thường bị hạn chế sử dụng. Việc khôi phục hệ thống khi gặp lỗi có thể được thực hiện bằng cách sử dụng các kỹ thuật như watchdog timer – nếu phần mềm không đều đặn nhận được các tín hiệu watchdog định kì thì hệ thống sẽ bị khởi động lại.

- Một số vấn đề cụ thể về độ tin cậy như:

• Hệ thống không thể ngừng để sửa chữa một cách an toàn, VD như ở các hệ thống không gian, hệ thống dây cáp dưới đáy biển, các đèn hiệu dẫn đường … Giải pháp đưa ra là chuyển sang sử dụng các hệ thống con dự trữ hoặc các phần mềm cung cấp một phần chức năng.

• Hệ thống phải được chạy liên tục vì tính an toàn, VD như các thiết bị dẫn đường máy bay, thiết bị kiểm soát độ an toàn trong các nhà máy hóa chất,… Giải pháp đưa ra là lựa chọn backup hệ thống.

• Nếu hệ thống ngừng hoạt động sẽ gây tổn thất rất nhiều tiền của, VD như các dịch vụ buôn bán tự động, hệ thống chuyển tiền, hệ thống kiểm soát trong các nhà máy…

1.2.5 Các kiến trúc phần mềm HỆ THỐNG NHÚNG:

Một số loại kiến trúc phần mềm thông dụng trong các hệ thống nhúngnhư sau:

-Vòng lặp kiểm soát đơn giản:

Theo thiết kế này, phần mềm được tổ chức thành một vòng lặp đơn giản. Vòng lặp gọi đến các chương trình con, mỗi chương trình con quản lý một phần của hệ thống phần cứng hoặc phần mềm.

- Hệ thống ngắt điều khiển:

•Các hệ thống nhúngthường được điều khiển bằng các ngắt. Có nghĩa là các tác vụ của hệ thống nhúngđược kích hoạt bởi các loại sự kiện khác nhau. VD: một ngắt có thể được sinh ra bởi một bộ định thời sau một chu kỳ được định nghĩa trước, hoặc bởi sự kiện khi cổng nối tiếp nhận được một byte nào đó.

•Loại kiến trúc này thường được sử dụng trong các hệ thống có bộ quản lý sự kiện đơn giản, ngắn gọn và cần độ trễ thấp. Hệ thống này thường thực hiện một tác vụ đơn giản trong một vòng lặp chính. Đôi khi, các tác vụ phức tạp hơn sẽ được thêm vào một cấu trúc hàng đợi trong bộ quản lý ngắt để được vòng lặp xử lý sau đó. Lúc này, hệ thống gần giống với kiểu nhân đa nhiệm với các tiến trình rời rạc.

- Đa nhiệm tương tác:

•Một hệ thống đa nhiệm không ưu tiên cũng gần giống với kỹ thuật vòng lặp kiểm soát đơn giản ngoại trừ việc vòng lặp này được ẩn giấu thông qua một giao diện lập trình API. Các nhà lập trình định nghĩa một loạt các nhiệm vụ, mỗi nhiệm vụ chạy trong một môi trường riêng của nó. Khi không cần thực hiện nhiệm vụ đó thì nó gọi đến các tiến trình con tạm nghỉ (bằng cách gọi “pause”, “wait”, “yeild” …).Ưu điểm và nhược điểm của loại kiến trúc này cũng giống với kiểm vòng lặp kiểm soát đơn giản. Tuy nhiên, việc thêm một phần mềm mới được thực hiện dễ dàng hơn bằng cách lập trình một tác vụ mới hoặc thêm vào hàng đợi thông dịch (queue-interpreter).

- Đa nhiệm ưu tiên:

•Ở loại kiến trúc này, hệ thống thường có một đoạn mã ở mức thấp thực hiện việc chuyển đổi giữa các tác vụ khác nhau thông qua một bộ định thời. Đoạn mã này thường nằm ở mức mà hệ thống được coi là có một hệ điều hành và vì thế cũng gặp phải tất cả những phức tạp trong việc quản lý đa nhiệm.

•Bất kỳ tác vụ nào có thể phá hủy dữ liệu của một tác vụ khác đều cần phải được tách biệt một cách chính xác. Việc truy cập tới các dữ liệu chia sẻ có thể được quản lý bằng một số kỹ thuật đồng bộ hóa như hàng đợi thông điệp (message queues), semaphores … Vì những phức tạp nói trên nên một giải pháp thường được đưa ra đó là sử dụng một hệ điều hành thời gian thực. Lúc đó, các nhà lập trình có thể tập trung vào

việc phát triển các chức năng của thiết bị chứ không cần quan tâm đến các dịch vụ của hệ điều hành nữa.

- Vi nhân (Microkernel) và nhân ngoại (Exokernel):

•Khái niệm vi nhân (microkernel) là một bước tiếp cận gần hơn tới khái niệm hệ điều hành thời gian thực. Lúc này, nhân hệ điều hành thực hiện việc cấp phát bộ nhớ và chuyển CPU cho các luồng thực thi. Còn các tiến trình người dùng sử dụng các chức năng chính như hệ thống file, giao diện mạng lưới,… Nói chung, kiến trúc này thường được áp dụng trong các hệ thống mà việc chuyển đổi và giao tiếp giữa các tác vụ là nhanh.

•Còn nhân ngoại (exokernel) tiến hành giao tiếp hiệu quả bằng cách sử dụng các lời gọi chương trình con thông thường. Phần cứng và toàn bộ phần mềm trong hệ thống luôn đáp ứng và có thể được mở rộng bởi các ứng dụng.

- Nhân khối (monolithic kernels):

•Trong kiến trúc này, một nhân đầy đủ với các khả năng phức tạp được chuyển đổi để phù hợp với môi trường nhúng. Điều này giúp các nhà lập trình có được một môi trường giống với hệ điều hành trong các máy để bàn như Linux hay Microsoft Windows và vì thế rất thuận lợi cho việc phát triển. Tuy nhiên, nó lại đòi hỏi đáng kể các tài nguyên phần cứng làm tăng chi phí của hệ thống. Một số loại nhân khối thông dụng là Embedded Linux và Windows CE. Mặc dù chi phí phần cứng tăng lên nhưng loại hệ thống nhúng này đang tăng trưởng rất mạnh, đặc biệt là trong các thiết bị nhúng mạnh như Wireless router hoặc hệ thống định vị GPS. Lý do của điều này là:

- Hệ thống này có cổng để kết nối đến các chip nhúng thông dụng
- Hệ thống cho phép sử dụng lại các đoạn mã sẵn có phổ biến như các trình điều khiển thiết bị, Web Servers, Firewalls, …
- Việc phát triển hệ thống có thể được tiến hành với một tập nhiều loại đặc tính, chức năng còn sau đó lúc phân phối sản phẩm, hệ thống có thể được cấu hình để loại bỏ một số chức năng không cần thiết. Điều này giúp tiết kiệm được những vùng nhớ mà các chức năng đó chiếm giữ.
- Hệ thống có chế độ người dùng để dễ dàng chạy các ứng dụng và gỡ rối. Nhờ đó, qui trình phát triển được thực hiện dễ dàng hơn và việc lập trình có tính linh động hơn.
- Có nhiều hệ thống nhúng thiếu các yêu cầu chặt chẽ về tính thời gian thực của hệ thống quản lý. Còn một hệ thống như Embedded Linux có tốc độ đủ nhanh để trả lời cho nhiều ứng dụng. Các chức năng cần đến sự phản ứng nhanh cũng có thể được đặt vào phần cứng.

1.2.6 Hệ thống thời gian thực:

Như đã đề cập ở trên, một hệ thống có khả năng thực hiện thời gian thực nghĩa là hệ thống đó phải thực hiện các chức năng của mình trong một khoảng thời gian xác định và nhỏ nhất có thể chấp nhận được. Khi đáp ứng được yêu cầu này, hệ thống đó có thể gọi là hệ thống thời gian thực.

Các hệ thống này phải có khả năng đáp ứng các tín hiệu ngõ vào hoặc các sự kiện trong giới hạn một khoảng thời gian bắt buộc. Cho nên các hệ thống này không chỉ phải trả về một kết quả đúng mà còn phải nhanh nhất đáp ứng được yêu cầu về tốc độ của hệ thống. Trong các hệ thống thời gian thực, tốc độ cũng quan trọng không kém gì độ chính xác của nó.

Có 2 loại thời gian thực: thời gian thực cứng và thời gian thực mềm. Đối với hệ thống thời gian thực cứng, tất cả các chức năng của nó phải được thực thi chính xác trong một khoảng thời gian xác định, nếu không cả hệ thống sẽ bị lỗi nghiêm trọng. VD: hệ thống điều khiển không lưu, hệ thống dẫn đường tên lửa, thiết bị y tế ... Đối với hệ thống thời gian thực mềm, các chức năng phải được thực hiện trong một khoảng thời gian xác định nhỏ nhất nhưng không bắt buộc.

1.2.7 Hệ điều hành thời gian thực (RTOS) và kernel thời gian thực:

Một số các ứng dụng nhúng có thể thực hiện hiệu quả mà chỉ cần một chương trình đơn giản chạy độc lập điều khiển cả hệ thống. Tuy nhiên, đối với đa số các ứng dụng mang tính thương mại, một hệ thống nhúngcần phải có hệ điều hành thời gian thực hoặc kernel thời gian thực. Một kernel thời gian thực thường nhỏ hơn rất nhiều so với một RTOS hoàn chỉnh. Trong lý thuyết về hệ điều hành, kernel chính là một phần của hệ điều hành, nó sẽ được nạp lên bộ nhớ đầu tiên và vẫn tồn tại trong lúc chương trình hoạt động. Một kernel thời gian thực sẽ cung cấp hầu hết các dịch vụ cần thiết cho các ứng dụng nhúng.Do đó chỉ là một phần của hệ điều hành và được nạp thẳng lên bộ nhớ, nên một kernel thời gian thực thường có kích thước rất nhỏ, rất phù hợp cho các bộ nhớ có dung lượng thấp trong các hệ thống nhúng. Hình dưới mô tả một kernel trong một RTOS hoàn chỉnh.

Hoạt động của hệ thống nhúngđươc thực hiện theo chương trình, gồm các tác vụ (task) hoặc luồng (thread) trong việc đáp ứng các tín hiệu ngõ vào hay trong quá trình xử lý bình thường theo yêu cầu của hệ thống. Các quá trình xử lý phải trả về kết quả đúng trong một khoảng thời gian xác định.

1.2.8 Chương trình, tác vụ và luồng:

Một chương trình trên một hệ thống nhúngchính là một phần mềm có khả năng thực thi độc lập và có vùng nhớ riêng của mình. Nó bao gồm môi trường thực thi một chức năng cụ thể và khả năng tương tác với hệ điều hành. Một chương trình có thể được bắt đầu chạy một cách độc lập hoặc có thể từ các chương trình khác. Một hệ điều hành có khả năng thực thi nhiều chương trình cùng một lúc song song nhau.

Tuy nhiên, khi một chương trình có khả năng tự chia ra một vài phần có khả năng thực thi song song nhau, mỗi phần đó được gọi là một luồng. Một luồng chính là một phần trong chương trình và phụ thuộc về mặt chức năng so với các luồng khác nhưng lại có khả năng hoạt động độc lập nhau. Các luồng sẽ chia sẻ chung một bộ nhớ trong một chương trình. Khái niệm về tác vụ và luồng có thể thay thế cho nhau. Hình dưới mô tả sự khác nhau giữa chương trình và luồng.

1.2.9 Kiến trúc của hệ thống thời gian thực:

Kiến trúc của một hệ thống thời gian thực sẽ quyết định các luồng được thực thi khi nào và bằng cách nào. Có 2 kiến trúc phổ biến là kiến trúc điều khiển vòng lặp với polling và mô hình sắp xếp ưu tiên. Trong kiến trúc điều khiển vòng lặp với polling, kernel sẽ thực thi một vòng lặp vô hạn, vòng lặp này sẽ chọn ra luồng trong một mẫu được định trước. Nếu một luồng cần dịch vụ, nó sẽ được xử lý. Có một vài biến thể của phương pháp này, tuy nhiên vẫn phải đảm bảo mỗi luồng đều có khả năng truy cập đến vi xử lý. Hình dưới mô tả cách xử lý của phương pháp này.

Mặc dù phương pháp điều khiển vòng lặp với polling rất dễ thực hiện, tuy nhiên nó vẫn có những hạn chế nghiêm trọng. Thứ nhất đó chính là nó sẽ mất rất nhiều thời gian, khi mà một luồng cần truy cập đến vi xử lý sẽ phải chờ đến lượt của mình và một chương trình có quá nhiều luồng sẽ bị chậm đi rất nhiều. Thứ hai, phương pháp này không có sự phân biệt giữa các luồng, luồng nào quan trọng và luồng nào ít quan trọng, từ đó xác định mức độ ưu tiên giữa các luồng.

Một phương pháp khác mà các kernel thời gian thực hay sử dụng đó chính là mô hình sắp xếp mức độ ưu tiên. Trong mô hình này, mỗi luồng sẽ đi kèm với mức độ ưu tiên của nó. Lúc này, vi xử lý sẽ thiết lập đường truy cập tới luồng nào có mức độ ưu tiên cao nhất khi nó đòi hỏi được phục vụ. Cũng có một vài biến thể của phương pháp này, tuy nhiên vẫn phải đảm bảo các luồng có mức độ ưu tiên thấp nhất vẫn phải có thể truy cập tới vi xử lý một vài lần. Hình dưới mô tả phương pháp cách xử lý của phương pháp này.

Một ưu điểm cực kỳ quan trọng của phương pháp này đó chính là nó có khả năng tạm hoãn thực thi một luồng khi có một luồng khác với mức độ ưu tiên cao hơn cần phục vụ. Quá trình lưu trữ lại các thông tin hiện thời của luồng bị tạm hoãn thực thi khi có một luồng khác với mức độ ưu tiên cao hơn cần phục vụ gọi là “context switching”. Quá trình này phải được thực hiện nhanh và đơn giản để luồng bị tạm hoãn có thể thực hiện tiếp nhiệm vụ của mình một cách chính xác khi nó lấy lại đươc quyền điều khiển.

Một hệ thống nhúng thời gian thực phải có khả năng đáp ứng lại các tín hiệu ngõ vào hay các sự kiện một cách nhanh nhất và chính xác nhất, đây chính là các ngắt của hệ thống. Ngắt của hệ thống sẽ phải làm cho vi xử lý ngưng nhiệm vụ đang thực thi để xử lý ngắt. Một ngắt sẽ được xử lý bởi ISR (interrupt service routine), nó có khả năng kích hoạt một luồng có mức độ ưu tiên cao hơn luồng đang được thực thi. Lúc này, nó sẽ tạm hoãn lại luồng hiện tại để dành quyền cho luồng mới có mức độ ưu tiên cao hơn. Ngắt có thể được tạo ra bởi phần mềm (ngắt mềm) hay bởi các thiết bị phần cứng (ngắt cứng).

1.3. Phát triển ứng dụng nhúng

Các ứng dụng nhúng ngày nay rất rộng rãi và sẽ được phát triển ngày càng cao ở cả phần cứng lẫn phần mềm. Các ứng dụng nhúng đều cần phải có thời gian thực, đây

là một sự khác biệt rất lớn giữa một hệ thống nhúng và một hệ thống máy tính truyền thống. Ngày nay để tăng tốc độ của một hệ thống nhúng, nó phải có khả năng thực hiện xử lý song song giữa các luồng với nhau. Do vậy, cách viết các chương trình phần mềm truyền thống sẽ không còn phù hợp khi lập trình cho các hệ thống nhúngđa luồng nữa. Hơn nữa, một vi xử lý trong hệ thống nhúng đòi hỏi tốc độ cao sẽ không còn làm nhiệm vụ xử lý, mà chỉ còn làm nhiệm vụ điều khiển và giám sát hoạt động của hệ thống. Chức năng xử lý luồng dữ liệu sẽ được các module phần cứng trong hệ thống đảm nhận và được thực hiện song song nhau. Kiến trúc một hệ thống nhúngthời gian thực đã có sự khác biệt rất nhiều và những cải tiến đáng kể so với kiến trúc hệ thống máy tính truyền thống trước kia. Điều này nhằm đảm bảo về sự chính xác và cải thiện tốc độ của hệ thống. Hầu hết các hệ thống nhúng ngày nay dùng ngôn ngữ C để lập trình, tuy nhiên một số rất ít vẫn dùng hợp ngữ.

- Xu hướng phát triển của các hệ thống nhúng hiện nay là:

    - Phần mềm ngày càng chiếm tỷ trọng cao và đã trở thành một thành phần cấu tạo nên thiết bị bình đẳng như các phần cơ khí, linh kiện điện tử, linh kiện quang học…
    - Các hệ nhúng ngày càng phức tạp hơn đáp ứng các yêu cầu khắt khe về thời gian thực, tiêu ít năng lượng và hoạt động tin cậy ổn định hơn.
    - Các hệ nhúng ngày càng có độ mềm dẻo cao đáp ứng các yêu cầu nhanh chóng đưa sản phẩm ra thương trường, có khả năng bảo trì từ xa, có tính cá nhân cao.
    - Các hệ nhúng ngày càng có khả năng hội thoại cao, có khả năng kết nối mạng và hội thoại được với các đầu đo cơ cấu chấp hành và với người sử dụng,.
    - Các hệ nhúng ngày càng có tính thích nghi, tự tổ chức cao có khả năng tái cấu hình như một thực thể, một tác nhân.
    - Các hệ nhúng ngày càng có khả năng tiếp nhận năng lượng từ nhiều nguồn khác nhau (ánh sáng, rung động, điện từ trường, sinh học….) để tạo nên các hệ thống tự tiếp nhận năng lượng trong quá trình hoạt động.

- Trong các hệ nhúng, hệ thống điều khiển nhúng đóng một vai trò hết sức quan trọng.
- Nhu cầu hệ thống nhúng trên thế giới:

Trong thế giới công nghệ thông tin, các “ông lớn” như IBM, Microsoft, Intel đã chuyển hướng một số bộ phận nghiên cứu phát triển của mình sang làm hệ thống nhúng từ rất sớm. Điển hình là Microsoft với các máy chơi game Xbox, hệ điều hành nhúng Windows CE, Intel với các dòng chip xử lý nhúng như Intel 8008, 8080, 8085, 3000, các thẻ nhớ Nand Flash, các vi điều khiển MCS 51/251, MCS 96/296 …Bên cạnh đó là sự xuất hiện của hàng loạt các nhà sản xuất vi xử lý cho hệ thống nhúngnhư ARM, Atmel, Renesas…Thị trường hệ thống nhúng có tiềm năng phát triển vô cùng

lớn. Theo các nhà thông kê trên thế giới thì số chip xử lý trong các máy PC và các server, các mạng LAN, WAN, Internet chỉ chiếm không đầy 1% tổng số chip vi xử lý có trên thế giới. Hơn 99% số vi xử lý còn lại nằm trong các hệ hệ thống nhúng. Số liệu đánh giá chi tiết của nhóm nghiên cứu BCC (BCC Research Group) về thị trường HỆ THỐNG NHÚNG toàn cầu đến năm 2009 :"Thị trường hệ thống nhúngtoàn cầu đạt doanh thu 45,9 tỷ USD trong năm 2004 và dự báo sẽ tăng 14% trong vòng năm năm tới, đạt 88 tỷ USD.Trong đó thì thị trường phần mềm nhúng sẽ tăng trưởng từ 1,6 tỷ USD năm 2004 lên 3,5 tỷ USD năm 2009, với mức tăng trung bình hằng năm là 16%. Tốc độ tăng trưởng phần cứng nhúng sẽ là 14,2% một năm, đạt 78,7 tỷ USD năm 2009, trong khi lợi nhuận các board mạch nhúng sẽ tăng 10% một năm.

Tại Châu Á, Nhật Bản đang dẫn đầu về thị trường nhúng và là một trong những thị trường phần mềm nhúng hàng đầu thế giới. Theo thống kê của JISA (Hiệp hội Dịch vụ Công nghệ Thông tin Nhật Bản), phần mềm nhúng hiện nay chiếm tới 40% thị phần phần mềm Nhật Bản, với các sản phẩm rất đa dạng : lò vi ba, máy photocopy, máy in laser, máy FAX, các bảng quảng cáo sử dụng hệ thống đèn LED, màn hình tinh thể lỏng…Năm 2004, thị trường phần mềm nhúng của Nhật Bản đạt khoảng 20 tỷ USD với 150.000 nhân viên. Đây được coi là thị trường đầy hứa hẹn với các đối tác chuyên sản xuất phần mềm nhúng như Trung Quốc, Indonesia, Nga, Ireland, Israel và cả Việt Nam.

- Nhu cầu hệ thống nhúng ở Việt Nam:

Với tốc độ tăng trưởng nhanh như vậy, cơ hội cho các doanh nghiệp Việt Nam đối với loại hình phần mềm mới mẻ này đang mở rộng. Chủ tịch Hiệp hội doanh nghiệp phần mềm Việt Nam (VINASA) Trương Gia Bình cho rằng, các doanh nghiệp Việt Nam đang có một số lợi thế. Đó là nguồn nhân lực công nghệ thông tin rẻ và tiếp thu nhanh, có kinh nghiệm làm gia công phần mềm cho nước ngoài, được Chính phủ quan tâm và hỗ trợ phát triển…Tuy nhiên, Việt Nam mới chỉ là "lính mới" trong sân chơi sôi động này.Ở Việt Nam, hệ thống nhúngmới được quan tâm trong thời gian gần đây. Các doanh nghiệp làm phần mềm nhúng cũng chưa nhiều, mới có một số trung tâm thuộc các trường Đại học Quốc gia, Đại học Bách khoa, các đơn vị như Học viện Kỹ thuật quân sự, Viện nghiên cứu Điện tử - Tin học và Tự động hóa, Tổng công ty Điện tử - Tin học, Công ty thiết bị Điện tử y tế, Công ty VTC – Truyền hình số mặt đất và một số công ty phần mềm khác…Các sản phẩm phần mềm nhúng "made in Việt Nam" có lẽ mới chỉ là con số khá khiêm tốn, còn lại là làm gia công cho nước ngoài. Có thể điểm ra một vài sản phẩm tiêu biểu do người Việt làm ra như phần mềm nhúng cho đầu thu kỹ thuật số của Công ty Điện tử HANEL (giải Sao Khuê 2005), Nhúng cá thể hóa thẻ thông minh của Công ty Liên doanh thẻ thông minh MK (giải Sao Khuê 2005)…Con đường để đến với thành công trong sản xuất và xuất khẩu phần mềm nhúng của các doanh nghiệp Việt Nam còn rất nhiều chông gai. Theo ông Phan Văn Hòa, Giám đốc Trung tâm công nghệ của FPT Software, thách thức lớn nhất Việt Nam phải vượt qua hiện nay là chưa có nhiều kinh nghiệm trong lĩnh vực mới mẻ này, mới

chỉ loanh quanh làm gia công phần mềm, làm thuê theo đơn đặt hàng của nước ngoài, chưa có nhiều trung tâm đào tạo chuyên sâu về hệ thống nhúng. Tại hội thảo về CNTT tổ chức tại Hải Phòng tháng 9-2005, Hiệp hội doanh nghiệp phần mềm Việt Nam VINASA cho rằng, xây dựng và phát triển phần mềm nhúng là một trong 3 mũi nhọn có thể coi là đột phá cho hướng đi của công nghệ phần mềm Việt Nam, bên cạnh việc phát triển game và các giải pháp ERP. Trong chiến lược phát triển công nghệ thông tin đến năm 2010, phần mềm nhúng được coi là một trong những sản phẩm trọng điểm.

- Những thách thức và các vấn đề tồn tại của hệ nhúng:

• Độ phức tạp của sự liên kết đa ngành phối hợp cứng - mềm.Độ phức tạp của hệ thống tăng cao do nó kết hợp nhiều lĩnh vực đa ngành, kết hợp phần cứng - mềm, trong khi các phương pháp thiết kế và kiểm tra chưa chin muồi. Khoảng cách giữa lý thuyết và thực hành lớn và còn thiếu các phương pháp và lý thuyết hoàn chỉnh cho khảo sát phân tích toàn cục các hệ nhúng.

• Thiếu phương pháp tích hợp tối ưu giữa các thành phần tạo nên hệ nhúng bao gồm lý thuyết điều khiển tự động, thiết kế máy, công nghệ phần mềm, điện tử, vi xử lý, các công nghệ hỗ trợ khác.

• Thách thức đối với độ tin cậy và tính mở của hệ thống: Do hệ thống nhúng thường phải hội thoại với môi trường xung quanh nên nhiều khi gặp những tình huống không được thiết kế trước dễ dẫn đến hệ thống bị loạn. Trong quá trình hoạt động một số phần mềm thường phải chỉnh lại và thay đổi nên hệ thống phần mềm có thể không kiểm soát được. Đối với hệ thống mở, các hãng thứ 3 đưa các module mới, thành phần mới vào cũng có thể gây nên sự hoạt động thiếu tin cậy.

CHƯƠNG 2:

VI XỬ LÝ ARM

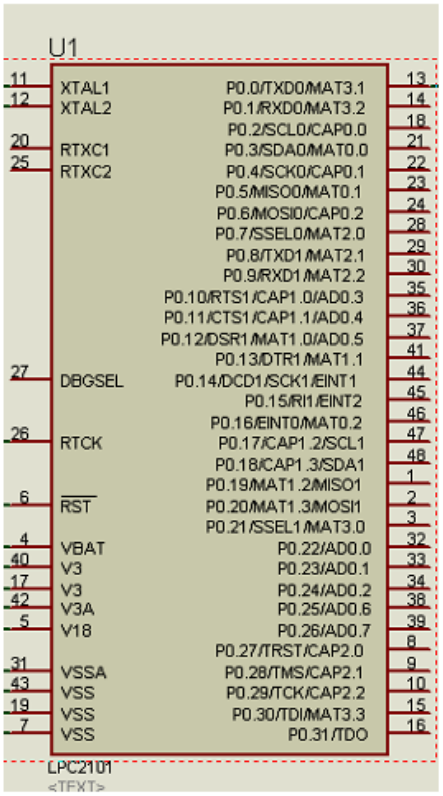
2.1. Tổng quan

Vi điều khiển ARM được phát triển theo kiến trúc RISC *(Reduced Instruction Set Computer):*

- Chỉ có các lệnh nạp hoặc lưu trữ là có thể tham chiếu tới bộ nhớ,
- Tồn tại ít lệnh và kiểu định địa chỉ, khuôn dạng lệnh cố định,
- Có nhiều tập thanh ghi,
- Các lệnh thực hiện trong một chu kỳ máy,
- Lệnh được thực hiện trực tiếp trên phần cứng(CISC có 1 chương trình thông dịch nhỏ),
- Chương trình biên dịch mã nguồn phức tạp(CISC-chương trình thông dịch phức tạp),
- Hỗ trợ cơ chế pipeline,
- Kích thước chương trình lớn.

Cấu trúc các chân:

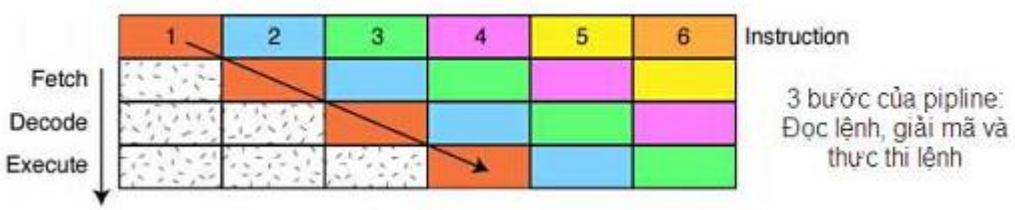
Thí dụ các chân của LPC2101:



Hình 2.1. Cấu trúc các chân của LPC2101

2.2. Cơ chế Pipeline

Cơ chế pipeline của ARM7, thực thi lệnh theo ba bước: đọc lệnh, giải mã lệnh và thực hiện lệnh.



Hình 2.2. Ba bước thực hiện của pipepline

- Pipeline có phần cứng độc lập để thực hiện các bước, trong khi lệnh thứ nhất đang thực thi, lệnh thứ 2 được giải mã và lệnh thứ 3 được đọc lên pipeline.

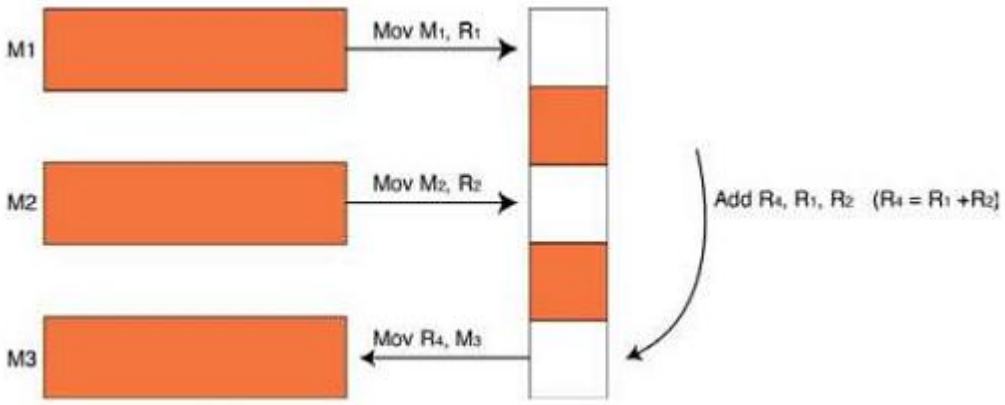
- Hầu hết các lệnh của ARM 7 được thực thi trong 1 chu kỳ máy.

- Pipeline làm việc rất tốt trong trường hợp chương trình không rẽ nhánh. ARM chỉ cho phép thực hiện các bước nhảy ngán trong đoạn chương trình.

- Pipeline là một thành phần của CPU, thanh ghi PC chạy ở 8 bytes đầu của lệnh hiện hành sẽ được thực thi. Thí dụ: 0x4000 LDR PC,[PC,#4]-> PC=0x400C

2.3. Các thanh ghi

ARM7 có kiến trúc kiểu *load and store*, bởi vậy, để thực hiện các lệnh xử lý dữ liệu thì tất cả các dữ liệu phải được tải từ bộ nhớ vào một tập các thanh ghi trung tâm, lệnh xử lý dữ liệu được thực hiện và lưu trữ dữ liệu trở lại bộ nhớ.

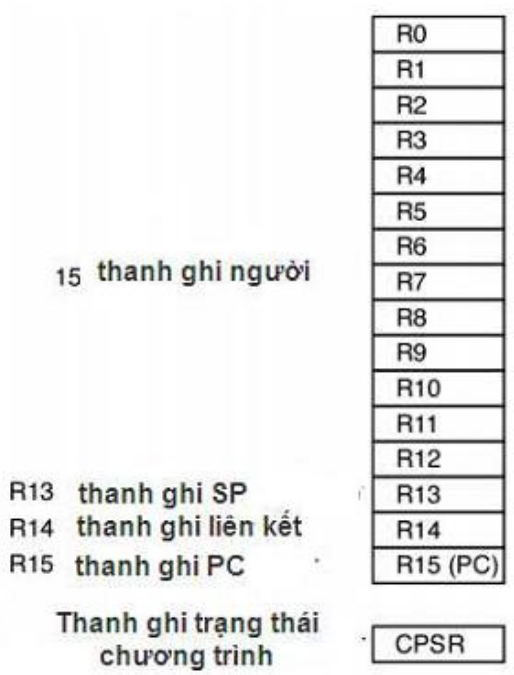


Hình 2.3. Kiến trúc load and store của ARM 7

ARM7 có 17 thanh ghi 32 bít:

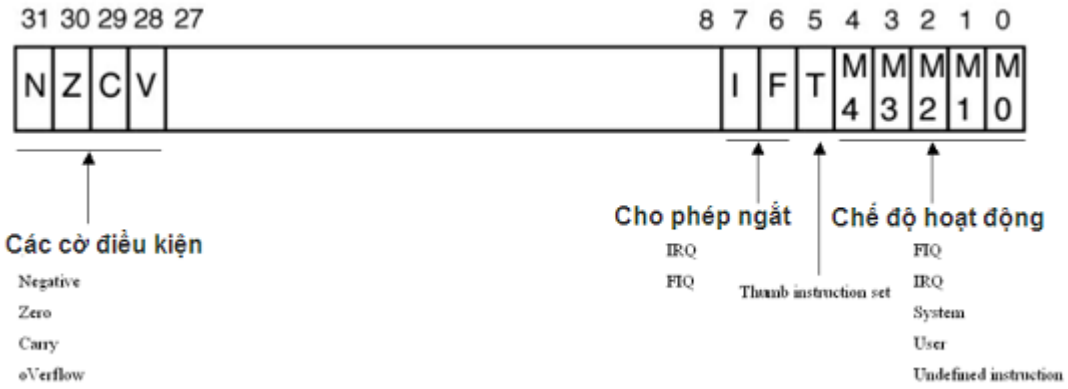
- Các thanh ghi RO đến R12 là các thanh ghi chung,
- Thanh ghi R13 là thanh ghi con trỏ ngăn xếp,
- R14 là thanh ghi liên kết,
- R15 là thanh ghi bộ đếm chương trinh (PC) và
- Thanh ghi trạng thái chương trình CPSR.

Thanh ghi R14 dùng trong chương trường hợp gọi đến chương trình con "gần" thì nó sẽ cất giữ địa chỉ trả về của chương trình chính, nếu trong chương trình con này gọi đến một chương trình con khác thì địa chỉ của chương trình chính phải được cất giữa vào ngăn xếp.



Hình 2.4. Các thanh ghi của ARM7

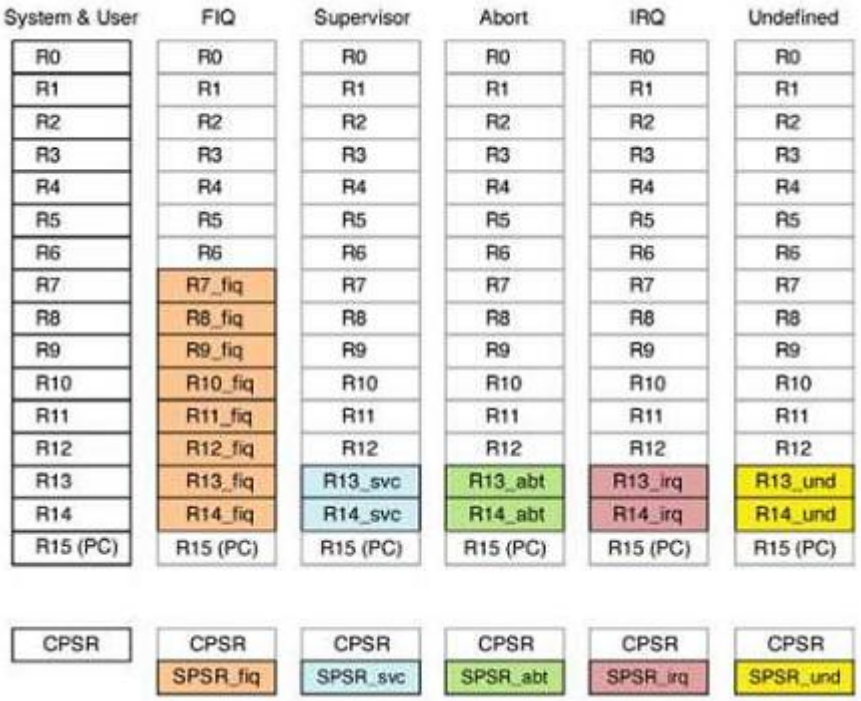
2.4. Thanh ghi trạng thái chương trình hiện hành



Hình 2.5. Thanh ghi trạng thái chương trình CPSR

CPU ARM7 thực thi 2 loại lệnh: Tập lệnh ARM 32 bít và tập lệnh được nén 16 bít. Bít T sẽ quyết định loại lệnh nào sẽ được thực thi, người lập trình không nên set hay xóa giá trị của bít này.

ARM7 có 7 chế độ hoạt động khác nhau, người lập trình thường chạy trong chế độ người dùng để truy cập đến các bank thanh ghi từ R0-R15 và thanh ghi trạng thái chương trình(CPSR). Tuy nhiên khi gặp các ngoại lệ như ngắt, lỗi bộ nhớ, ngắt mềm CPU sẽ chuyển sang chế độ khác. Mỗi chế độ các thanh ghi R13 và R14 có giá trị riêng. Ở chế độ ngắt nhanh FIQ các thanh ghi R7-R12 có giá trị giống nhau (không cần dùng stack để lưu chữ).



Hình 2.6. Các chế độ hoạt động của APU ARM7

2.5. Các mode ngoại lệ

Khi có một ngoại lệ xảy ra, CPU sẽ chuyển chế độ và thanh ghi PC sẽ được nhảy về địa chỉ của vecto ngoại lệ. Bảng vecto bắt đầu từ địa chỉ 0 trùng địa chỉ vecto ngắt.

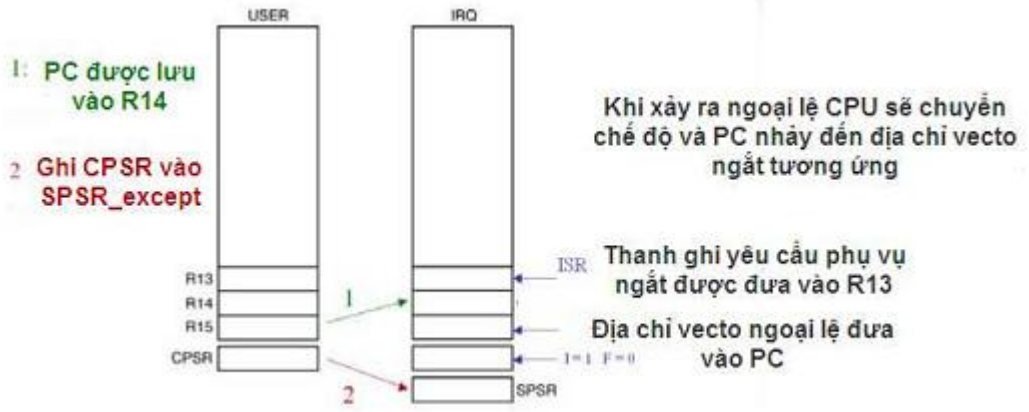
Mỗi vecto ngoại lệ là 4 bytes.

Exception	Mode	Address
Reset	Supervisor	0x00000000
Undefined instruction	Undefined	0x00000004
Software interrupt (SWI)	Supervisor	0x00000008
Prefetch Abort (instruction fetch memory abort)	Abort	0x0000000C
Data Abort (data access memory abort)	Abort	0x00000010
IRQ (interrupt)	IRQ	0x00000018
FIQ (fast interrupt)	FIQ	0x0000001C

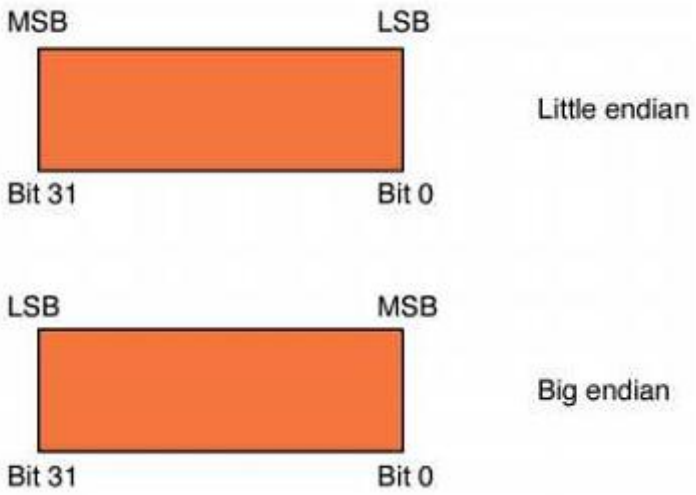
Bảng 2.1. Bảng các vecto ngắt

Priority	Exception
Highest 1	Reset
2	Data Abort
3	FIQ
4	IRQ
5	Prefetch Abort
Lowest 6	Undefined instruction SWI

Bảng 2.2. Thứ tự ưu tiên của các ngắt



Hình 2.7. Thí dụ về trật tự xử lý khi có một ngoại lệ ngắt xảy ra

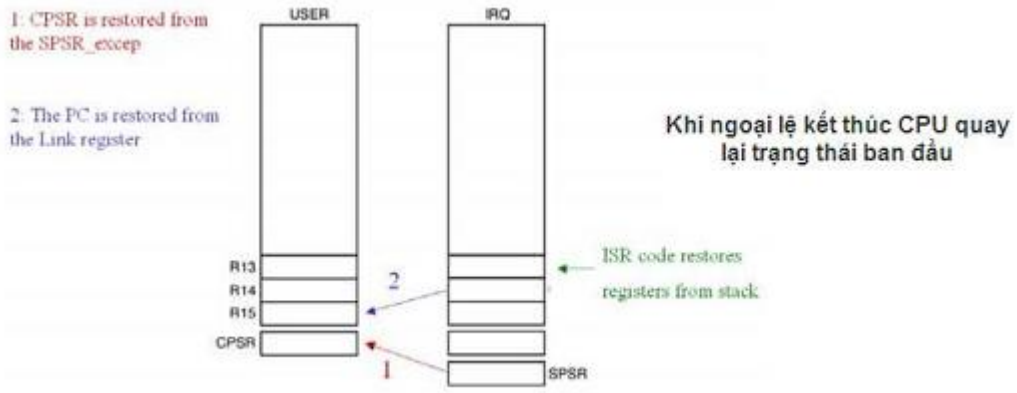


Hình 2.8. CPU trở lại trạng thái ban đầu khi kết thúc phụ vụ ngoại lệ

2.6. Tập lệnh ARM 7

ARM7 có 2 tập lệnh: Tập lệnh mở rộng 32 bít và tập lệnh nén (THUMB) 16 bít.

CPU ARM7 được thiết kế để hỗ trợ xử lý theo kiểu big endian hay little endian:



Hình 2.9. Hai kiểu xử lý của CPU ARM7

Một đặc điểm của ARM7 là tất cả các lệnh đều có thể là các lệnh có điều kiện, bằng cách so *sánh 4 bít từ bit 28 đến bít 31 của kết quả thực hiện lệnh với các bít điều kiện trong thanh ghi CPSR,* nếu điều khiện không thỏa mãn thì lệnh sẽ không được thực thi.

Các lệnh xử lý dữ liệu sẽ bị ảnh hưởng bởi các bít điều kiện trong thanh ghi CPSR. Hai lệnh cơ bản MOV và ADD có thể đặt các tiến tố đằng trước với 16 điều kiện như sau:

Suffix	Flags	Meaning
EQ	Z set	equal
NE	Z clear	not equal
CS	C set	unsigned higher or same
CC	C clear	unsigned lower
MI	N set	negative
PL	N clear	positive or zero
VS	V set	overflow
VC	V clear	no overflow
HI	C set and Z clear	unsigned higher
LS	C clear and Z set	unsigned lower or same
GE	N equals V	greater or equal
LT	N not equal to V	less than
GT	Z clear AND (N equals V)	greater than
LE	Z set OR (N not equal to V)	less than or equal
AL	(ignored)	always

Bảng 2.3. 16 điều kiện khi dung hai lệnh cơ bản MOV và ADD

Thí dụ: EQMOY RI ,#0x00800000;

- Giá trị 0x00800000 chỉ đưa vào RI khi kết quả cuối cùng của lệnh có các bít tương ứng với 4 bít trong thanh ghi CPSR và bít cờ z được set =1.
- Các lệnh của ARM7 có thể chia thành 6 nhóm: Các lệnh rẽ nhánh, các lệnh xử lý dữ liệu, truyền dữ liệu, truyền khối dữ liệu, lệnh số học và ngắt mềm.

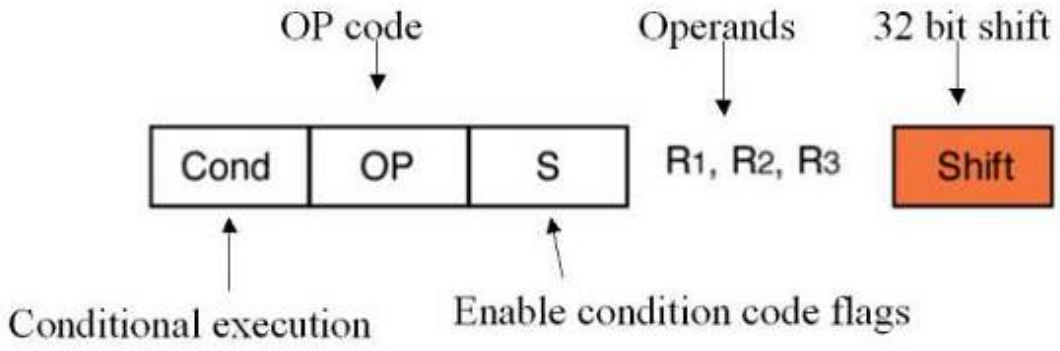
2.6.1. Các lệnh rẽ nhánh

Cho phép nhảy tiến hoặc lùi trong phạm vi 32MB, địa chỉ của lệnh tiếp theo sẽ được lưu vào thanh ghi liên kết R14.

Lệnh rẽ nhánh có 2 biến thể: Rẽ nhánh trao đổi (branch exchange) và rẽ nhánh liên kết trao đổi (branch link exchange). Hai lệnh này cơ giống nhau nhưng lệnh rẽ nhánh liên kết địa chỉ của lệnh tiếp theo được cộng thêm 4 bytes rồi đưa vào R14.

2.6.2. Các lệnh xử lý dữ liệu

Cú pháp tổng quát:



Hình 2.10. Cú pháp tổng quát của một lệnh xử lý dữ liệu có điều kiện

Mỗi lệnh đều có 2 toán hạng, trong đó toán hạng thứ nhất phải là thanh ghi, toán hạng còn lại có thể thanh ghi hoặc giá trị cụ thể.

Bít 'S' được sử dụng để điều khiển điều kiện của lệnh:

- S=1 thì điều kiện của lệnh phụ thuộc vào kết quả của lệnh,
- S = 0thì không có điều gì xảy ra

Nếu S=1 và PC là thanh ghi chứa kết quả thì SPSR của chế độ hiện hành được copy vào CPSR.

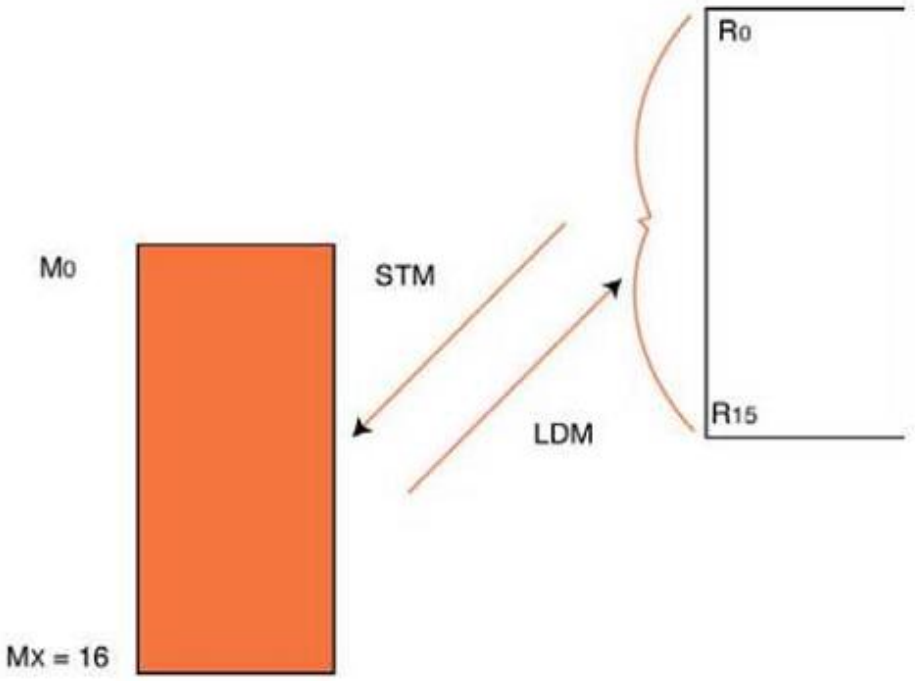
Mnemonic	Meaning
AND	Logical bitwise AND
EOR	Logical bitwise exclusive OR
SUB	Subtract
RSB	Reverse Subtract
ADD	Add
ADC	Add with carry
SBC	Subtract with carry
RSC	Reverse Subtract with carry
TST	Test
TEQ	Test Equivalence
CMP	Compare
CMN	Compare negated
ORR	Logical bitwise OR
MOV	Move
BIC	Bit clear
MVN	Move negated

Bảng 2.4. Bảng các lệnh xử lý dữ liệu

2.6.3. Cáclệnh truyền dữ liệu:

Mnemonic	Meaning
LDR	Load Word
LDRH	Load Half Word
LDRSH	Load Signed Half Word
LDRB	Load Byte
LRDSB	Load Signed Byte
STR	Store Word
STRH	Store Half Word
STRSH	Store Signed Half Word
STRB	Store Byte
STRSB	Store Signed Half Word

Bảng 2.5. Các lệnh truyền dữ liệu

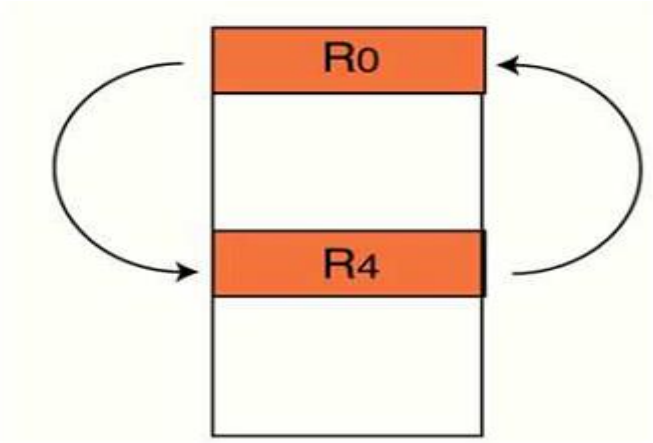


Hình 2.11. Các lệnh truyền một khối dữ liệu

2.6.4. Lệnh SWAP

ARM7 hỗ trợ các tín hiệu thời gian thực YỚi một lệnh swap cho phép trao đổi chỗ nội dung của hai thanh ghi.

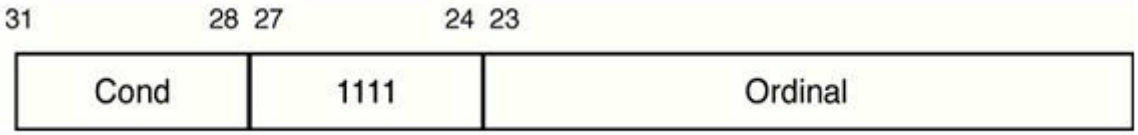
Lệnh này được hỗ trợ trong thư viện ARM chứ không trực tiếp từ ngôn ngữ **C.**



Hình 2.12. Mô tả lệnh swap trong ARM7

2.7. Ngắt mềm (SWI – Software Interput instruction)

Các ngắt mềm sinh ra một ngoại lệ khi thực thi, đưa vi xử lý vào chế độ hoạt động giám sát và PC nhảy tới địa chỉ 0x00000008. Cũng như các lệnh ARM khác, lệnh ngắt mềm chứa một mã điều kiện thực thi trong 4 bít thấp của toán hạng, các bít còn lại là trống rỗng.



Hình 2.13. Điều kiện của ngắt mềm để CPU vào chế độ giám sát

Có thể giả lặp chương trình con phục vụ ngắt của ngắt mềm như sau:

switch(*(R14-4) & 0x00FFFFFF) // Kiểm tra giá trịđược lưu trữ trong thanh ghi liên kết

case (SWI-1):

•••}

2.8. Đơn vị MAC (Multíply Accumulate Unit (MAC)

MAC hỗ trợ phép nhận số nguyên kiểu integer và long integer, khi nhân kiểu integer 2 thanh ghi 32 bít với nhau thì kết quả là 32 bít được đặt vào thanh ghi thứ 3. Khi nhân 32 bít kiểu long integer thì kết quả là 64 bít và được đặt vào 2 thanh ghi.

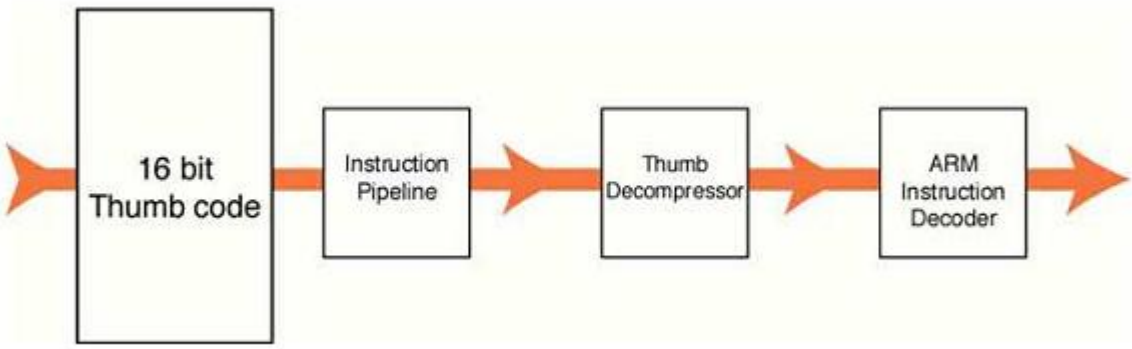
Các lệnh nhân dạng ASM:

Tên lệnh	Viết tắt	Kết quả
MUL	Multiply	32 bit
MULA	Multiply accumulate	32 bit
UMULL	Unsigned multiply	64 bit
UMLAL	Unsigned multiply accumulate	64 bit
SMULL	Signed multiply	64 bit
SMLAL	Signed multiply accumulate	64 bit

Bảng 2.6. Các lệnh nhân trong vi điều khiên ARM

2.9. Tập lệnh THUMB

Tập lệnh ARM là tập lệnh 32 bít, ARM có tập lệnh 16 bít gọi là tập lệnh THUMB. Tập lệnh THUMB thực chất là tập lệnh nén lại từ tập lệnh ARM.



Hình 2.14. Tập lệnh THUMB được nén lại từ tập lệnh ARM

Tập lệnh THUMB tiết kiệm được không gian nhớ 30% và chạy nhanh hơn 40% so với tập lệnh ARM.

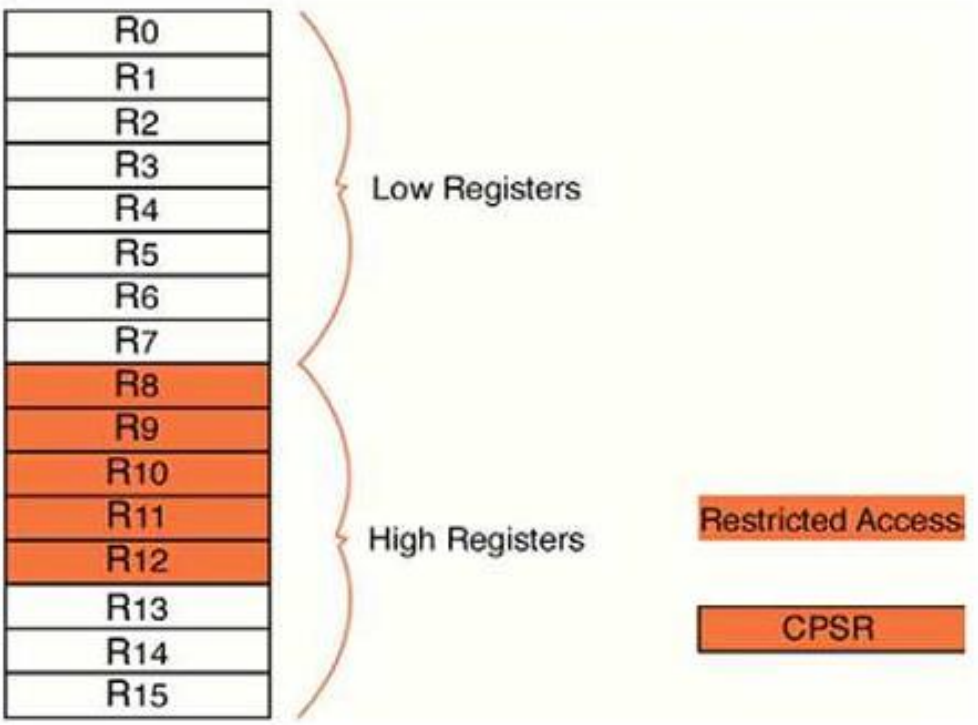
Tập lệnh THUMB không có điều kiện thực thi trừ các lệnh rẽ nhánh. Các lệnh xử lý dữ liệu thì cần có một thanh ghi nguồn và một thanh ghi đích.

Thí dụ: Vói lệnh cộng thanh ghi RO và RI:

Dùng lệnh ARM: ADD RO, RO,RI //RO = R0+R1

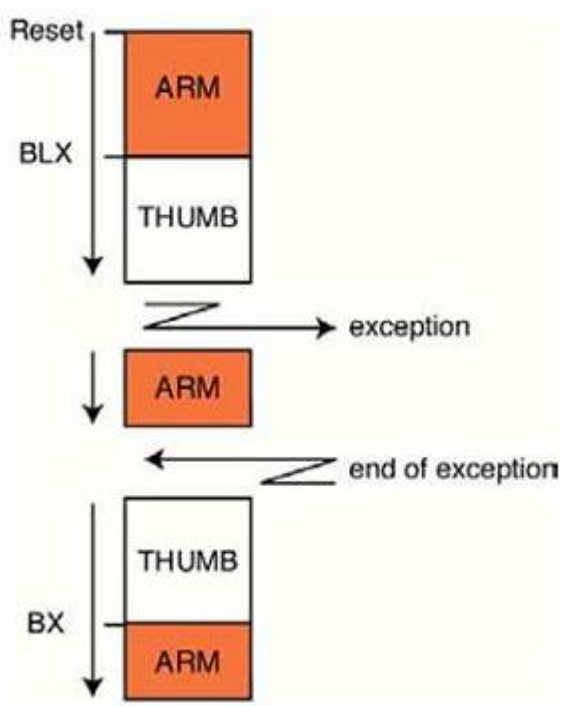
Dùng lệnh THUMB: ADD RO,RI // RO = R0+R1

Tập lệnh THUMB chỉ có thể truy cập đến các thanh ghi thấp từ R0-R7, các thanh ghi cao từ R8-R12 bị giới hạn truy cập:



Hình 2.15. Mô hình lập trình tập lệnh THUMB

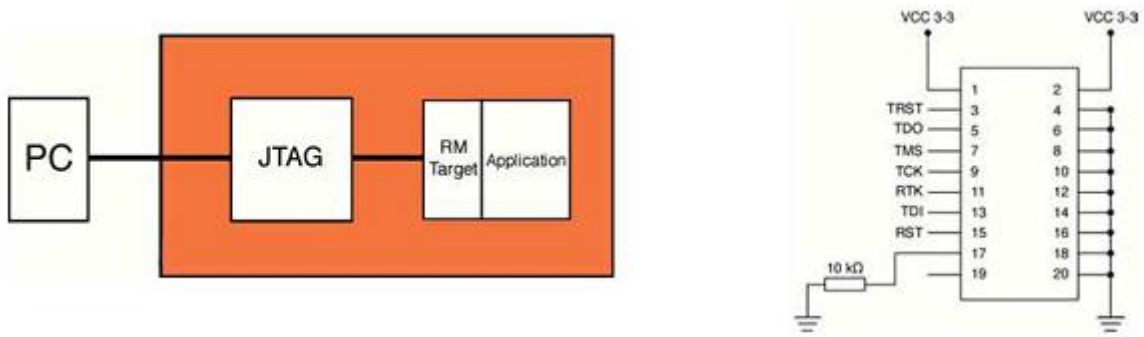
Người lập trình không thể truy cập trực tiếp vào thanh ghi CPSR và SPSR. Người lập trình có thể sử dụng 2 lệnh BLX và BX để chuyển chế độ hoạt động với các lệnh. Khi reset vi điều khiển làm việc với tập lệnh THUMB, khi có 1 ngoại lệ xảy ra thì sẽ chuyển sang làm việc với tập lệnh ARM, khi kết thúc ngoại lệ thì quay trở về làm việc với lệnh ARM.



Hình 2.16. Trao đổi giữa lệnh ARM và lệnh THUMB

2.10. Cổng JTAG

Họ LPC2000 của hãng Philips có nhiều cổng cho phép kết nối vi điều khiển với máy tính, thường được sử dụng nhất là cổng JTAG. Khi kết nối trực tiếp với máy tính cho phép người lập trình debug trực tiếp trên mạch phần cứng:

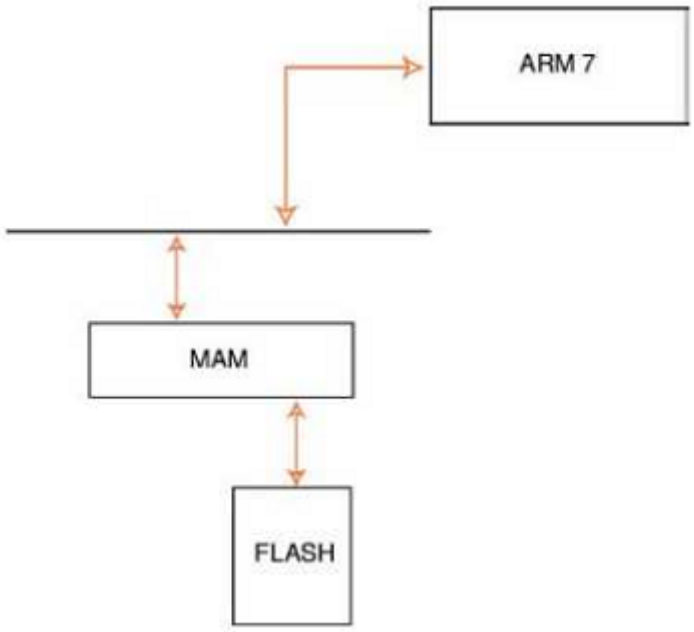


Hình 2.17. Kết nối LPC2000 với máy tính qua cổng JTAG

- Đồng thời kèm theo module ETM cho phép debug chương trình: như theo dõi thời gian thực, theo dõi sự kiện và phân tích quá trình thực thi.

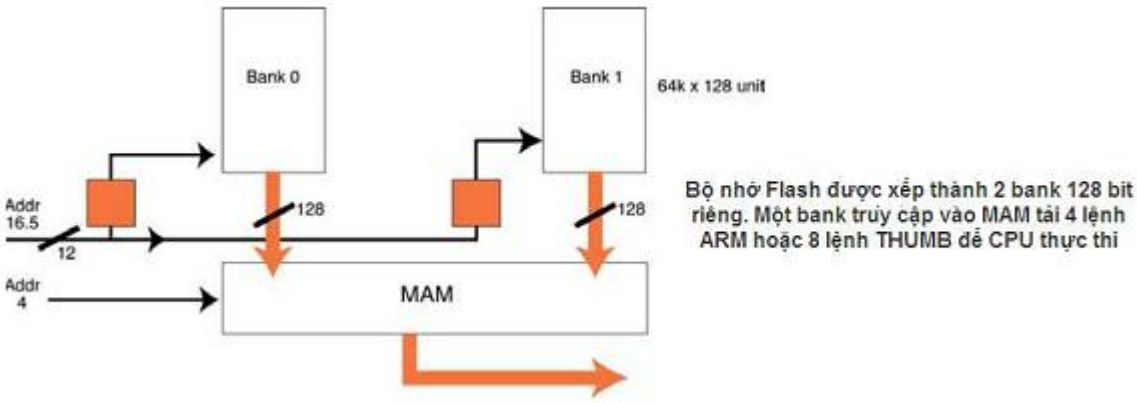
2.11. Memory Acelerator Module (MAM)

Là bộ nhớ nằm giữa bộ nhớ Flash và CPU ARM, có tốc độ thực thi cao.



Hình 2.18. Mô hình bộ nhớ MAM

- CPU ARM có thể chạy ở tốc độ 80MHz, mỗi lần chíp Flash truy cập hết 50ns.
- Flash chạy ở tốc độ 20MHz
- MAM được tạo ra như là một cache đầy đủ cho phép CPU dễ dàng truy cập



trực tiếp vào FLASH.

Hĩnh 2.19. Truy cập bộ nhớ FLASH qua MAM

- Khỉ đọc các lệnh từ bank thứ nhất thì bank thứ hai được chốt.
- MAM là trong suốt với người dùng và được cấu hình bởi 2 thanh ghi: điều khiển (MAMCR) và định thời (MAMTIM). Thanh ghi định thời được sử dụng để điều

khiển mối quan hệ giữa CPU và FLASH bằng cách thiết lập 3 bít đầu tiên của nó để chỉ định chu kỳ xung nhịp của CPU được yêu cầu bởi MAM để truy cập vào FLASH.

- Khi FLASH có tốc độ 20MHz và CPU có có thể có tốc độ cực đại là 60MHz, số chu kỳ yêu cầu truy cập FLASH là 3.

Thí dụ: cấu hình MAM

```
#include "LPC21xx.h"
void ChangeGPIOPinState(unsigned int State);
int main(void){
    unsigned int delay,val;
    unsigned int FLASHer = 0x00010000; // Khai báo cục bộ
    IODIRO = 0x00FF0000;    //Thiết lập các chân ra
VPBDIV = 0x02;
ADCR = 0x00270601;          // Thiết lập A/D: 10-bit AIN0 @ 3MHz
ADCR 1= 0x01000000;     // Khởi tạo bộ chuyển đổi A/D
while(l)
{ do{
val = ADDR; // Đọc thanh ghi dữ liệu bộ chuyển đổi A/D }while ((val &0x80000000) == 0); val = ((val» 6) & 0x03FF); if (val <0x80)
{
MAMCR = 0;
MAMTIM = 0x03;
MAMCR = 0x02;
}else
{ MAMCR = 0x0; }
for(delay = 0;delay<0xl00000;delay++) //tạo vòng lặp {;}
ChangeGPIOPinState(FLASHer); //Đổi trạng thái các chân ra ở cổng
FLASHer = FLASHer «1; //Dịch đến đèn led tiếp if(FLASHer&0x01000000)
{
FLASHer = 0x00010000; //Lặp lại đèn đầu tiên // overflow }}}
void ChangeGPIOPinState(unsigned int State)
```

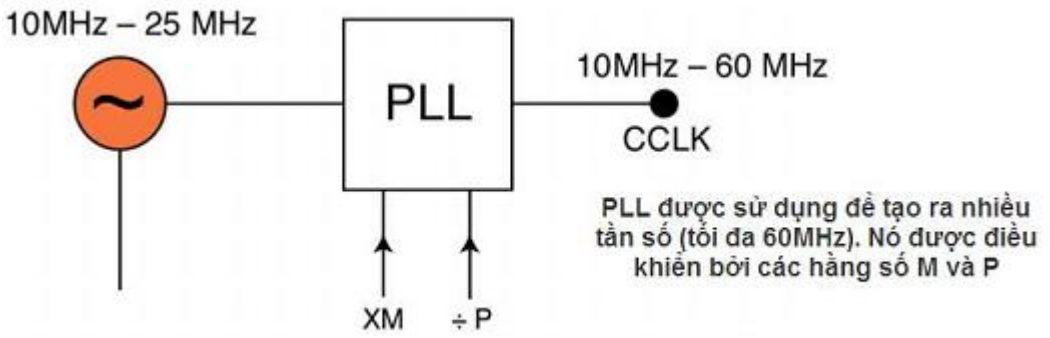
{

IOSETO = State; //set output pins IOCLRO = ~state; //clear output pins

2.12. PLL- Phase Locked Loop

Tạo ra một tần số dao động ngoài từ 10-25MHz từ mạch dao động cơ bản và có thể tăng lên 60 MHz để cung cấp cho CPU ARM và thiết bị ngoại vi.

Tần số đầu ra của PLL có thể thay đổi tự động, cho phép thiết bị điều chỉnh theo



tốc độ thực thi để duy trì nguồn năng lượng khi ở trạng thái rảnh rỗi.

Hình 2.20. Phase Locked Loop

Hai hằng M và p phải được lập trình để quyết định xung clock (Cclk) cho CPU và AHB.

Hằng thứ nhất được nhân một cách tuyến tính với tần số dao động bên ngoài đưa vào. Tần số ra của PLL là:

Cclk=M X Osc

Ngược lại PLL lại được điều khiển bởi một dao động hoạt động hiện hành (CCO) ở dải tần 156MHz-320MHz. Hằng số thứ 2 phải được lập trình đề đảm bảo sao cho cco được giữ một giá trị cụ thể:

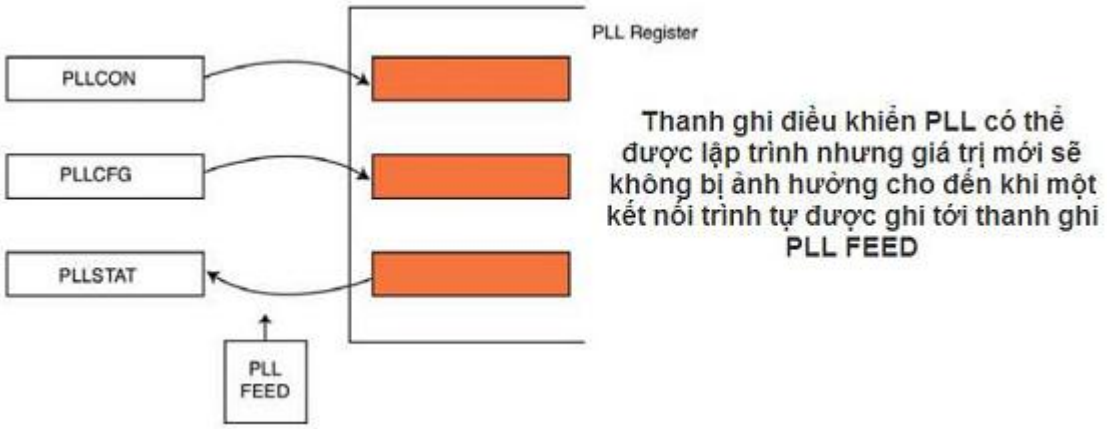
Fcco = Cclk X 2 X p

Trên board phát triển thì dao động thạch anh là 12MHz, bởi vậy để CPU đạt được tốc độ tối đa 60MHz thì: M = Cclk/Osc = 60/12 =5

và 156< Fcco <320 = 60 X 2 X p

Thực nghiệm thì p=2.

Giao diện lập trình PLL:



Hình 2.21. Giao diện lập trình PLL

Giá trị trong các thanh ghi PLLCON, PLLCFG và PLLSTAT sẽ được ghi sau khi giá trị trong PLL FEED được ghi.

Khi cập nhật giá trị thanh ghi PLLCON và PLLCFG thì phải ghi liên tiếp hai giá trị OxOOOOOOAA và 0x00000055 cho thanh ghi PLLFEED, các giá trị này phải ghi trong các chu kỳ liên tiếp.

Nếu lập trình cho phép các ngắt thì ngắt sẽ sinh ran gây sau khỉ từ đầu tiên được ghi và các thiết lập mới cho PLL sẽ không có ảnh hường.

Đê cài đặt PLL phải ghi các giá trị cho p và M tới thanh ghi PLLCFG, sau đó set DO của thanh ghi PLLCON để cho phép PLL khởi động.

Giá trị của M chiếm 5 bít thấp (D4-D0), p chiếm 2 bít D6D5:

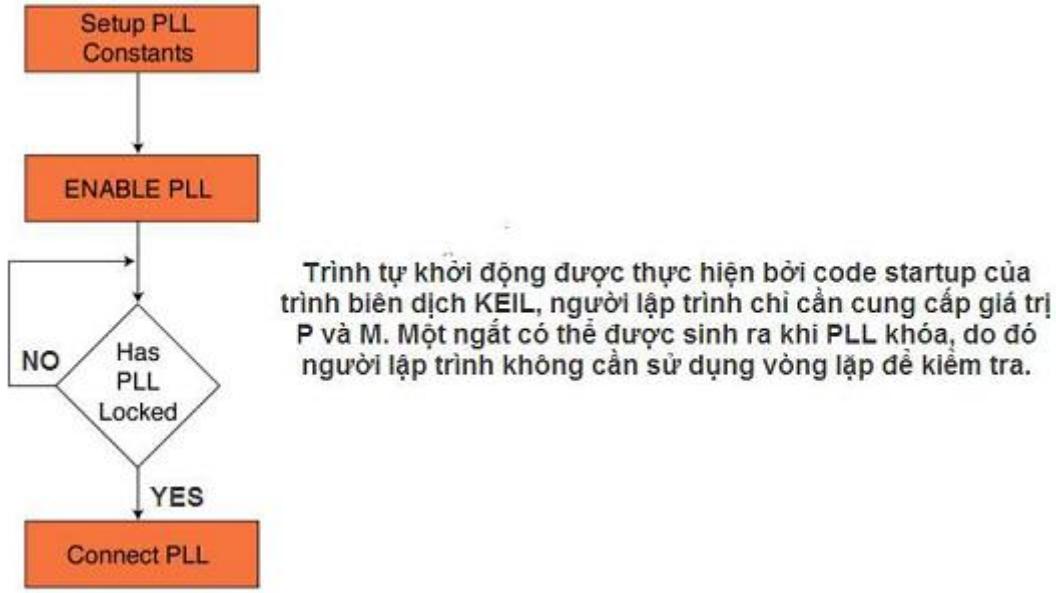
PSEL Bits (PLLCFG bits [6:5])	Value of P
00	1
01	2
10	4
11	8

MSEL Bits (PLLCFG bits [4:0])	Value of M
00000	1
00001	2
00010	3
00011	4
...	...
11110	31
11111	32

Bảng 2.7. Giá trị của p và M trong thanh ghi PLLCFG

PLL mất một khoảng thời gian xác định đủ để sử dụng nguồn xung clock. Sự khởi động PLL có thể được kiểm tra bằng cách đọc bít LOCK (D10) trong thanh ghi trạng thái PLLSTAT.

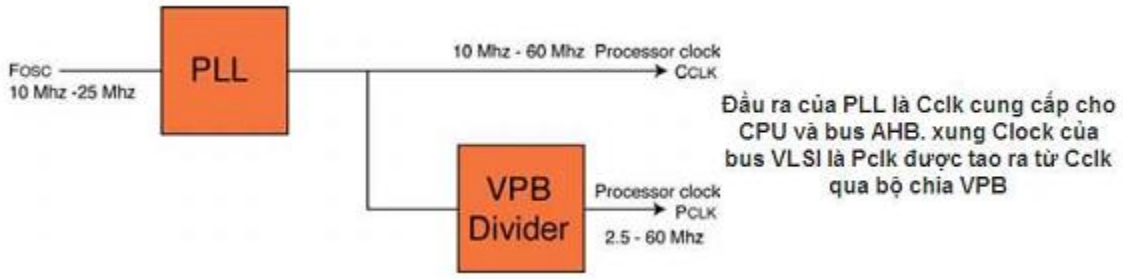
Khi LOCK bít =1, PLL có thể được sử dụng như nguồn xung clock chính. Một ngắt có thể được sinh ra khi PLL khóa, bởi vậy người lập trình có thể thực hiện các nhiệm vụ khác khi PLL khởi động. Khi PLL bị khóa thì có thể thay thế nguồn xung cho Cclk bằng cách điều khiển bít PLLC ương thanh ghi PLLCON.



Hình 2.22. Trình tự khởi động PLL

2.13. Bộ chia bus (VLSI Peripheral Bus Divider)

Mạch dao động ngoài hoặc đầu ra của PLL được sử dụng để tạo ra nguồn xung Cclk cho CPU ARM hoặc hệ thống bus có tần số cao. Các thiết bị ngoại vi có thể sử dụng bus VPB riêng biệt



Hình 2.23. Tạo xung Pclk từ Cclk

Bộ chia có thể chia tốc độ Cclk xuống 2 đến 24 lần. Thanh ghi trong bộ chia VPBDIV có thể lập trình được và chứa số ần giảm tắc độ. Tại thời điểm khỏi động, giá trị cực đại được nạp và bằng VA giá trị Cclk lúc khởi động.

Hiện nay tất cả các thiết bị ngoại vi của họ vi điều khiển LPC có thể chạy ở tần số 60MHz. Vì vậy, bộ chia tần VPB thường được sử dụng để tiết kiệm nguồn bàng cách tạo xung clock chấp nhận được cho các ứng dụng.

Thỉ du: cấu hình PLL và VPB:

Cấu hình PLL để tạo ra tần số Cclk là 60MHz và Pclk là 30MHz với xung đầu vào là 12MHz:

void init_PLL(void)

{

PLLCFG = 0x00000024; // Thiết lập hệ số nhân và bộ chia cho PLL

//give 60.00 MHz

PLLCON = 0x00000001; // Kích hoạt PLL

PLLFEED = OxOOOOOOAA; // Cập nhật thanh ghi PLLFEED PLLFEED = 0x00000055;

while (!(PLLSTAT & 0x00000400)); // kiểm tra bít Lock

PLLCON = 0x00000003; // Kết nối tói PLL

PLLFEED = OxOOOOOOAA; //Cập nhật các thanh ghi PLL

PLLFEED = 0x00000055;

VPBDIV = 0x00000002; //Thiết lập bus VLSI với tần số 30.000MHz

}

2.13. Các cổng vào ra

Các cổng vào ra của ARM là 32 bít, vi điều khiển có thể chỉ có 1 cổng hoặc có nhiều hơn, nhưng các chân dùng cho cổng vào/ra cũng là các chân dùng chung cho các mục đích khác như biến đổi AD, ngắt, giao diện SPI, I2C,...

Các chân được sử dụng vào mục đích nào tùy vào việc cấu hình chúng cho mục đích đó.

CHƯƠNG 3:

HỆ ĐIỀU HÀNH NHÚNG EMBEDDED LINUX

3.1. Giới thiệu hệ điều hành nhúng

3.1.1 Hệ điều hành

Hệ điều hành là một chương trình quản lý phần cứng máy tính. Nó cung cấp nền tảng cho các chương trình ứng dụng và đóng vai trò trung gian giao tiếp giữa người dùng máy tính và phần cứng của máy tính đó. Hệ điều hành thiết lập cho các tác vụ này rất đa dạng. Một vài hệ điều hành thiết kế tiện dụng trong khi một số khác thiết kế hiệu quả hoặc kết hợp cả hai.

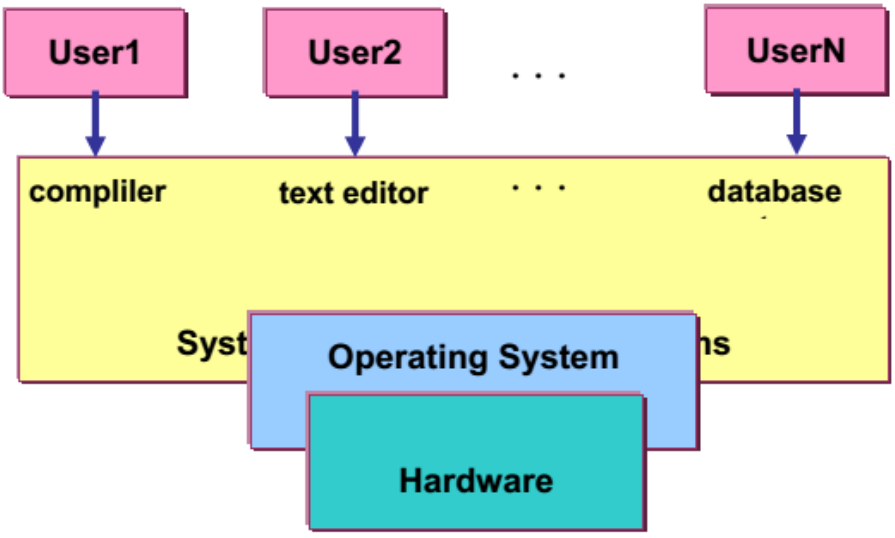
Một hệ điều hành là một thành phần quan trọng của mọi hệ thống máy tính. Một hệ thống máy tính có thể được chia thành bốn thành phần: phần cứng, hệ điều hành, các chương trình ứng dụng và người dùng.

- **Phần cứng (Hardware):** bao gồm bộ xử lý trung tâm (CPU), bộ nhớ, thiết bị xuất/nhập,..cung cấp tài nguyên cơ bản cho hệ thống.

- **Các chương trình ứng dụng (application programs):** trình biên dịch (compiler), trình soạn thảo văn bản (text editor), hệ cơ sở dữ liệu (database system), trình duyệt Web,..định nghĩa cách mà trong đó các tài nguyên được sử dụng để giải quyết yêu cầu của người dùng.

- **Người dùng (user):** có nhiều loại người dùng khác nhau, thực hiện những yêu cầu khác nhau, do đó sẽ có nhiều ứng dụng khác nhau.

- **Hệ điều hành (operating system):** hay còn gọi là chương trình hệ thống, điều khiển và hợp tác việc sử dụng phần cứng giữa những chương trình ứng dụng khác nhau cho những người dùng khác nhau. Hệ điều hành có thể được khám phá từ hai phía: người dùng và hệ thống.



Hình 3.1 Mô tả các thành phần của một hệ thống máy tính

3.1.2. Hệ điều hành nhúng

Hệ điều hành nhúng mang đặc trưng cơ bản của hệ điều hành

- Quản lý tài nguyên phần cứng và phần mềm của hệ thống.
- Trung gian giữa phần cứng và phần mềm, giúp phần cứng làm việc trong suốt với phần mềm ứng dụng.
- Cung cấp giao diện hàm chuẩn cho phần mềm ứng dụng.

3.2. Các hệ điều hành nhúng điển hình

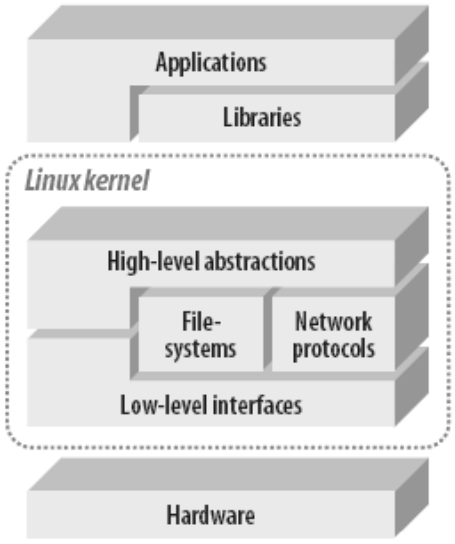
3.2.1. Embedded Linux

Trong những năm gần đây, Linux đã trở nên phổ biến trên các thiết bị nhúng, đặc biệt là trên các sản phẩm điện tử tiêu dùng, thiết bị định tuyến, chuyển mạch, các ứng dụng trên internet và trên ôtô.

Bởi vì bản chất của Linux là module Linux, do đó dễ dàng làm nhỏ lại cho vừa môi trường hoạt động bằng cách bớt các chương trình tiện ích, công cụ, và hệ thống dịch vụ không cần thiết được nhúng vào trong một môi trường hoạt động. Một trong những lợi thế lớn của Linux đó là nó là một hệ điều hành đầy đủ các chức năng, với hỗ trợ cho các mạng đang trở thành một yêu cầu rất quan trọng trong bất kì hệ nhúng nào. Do có thể thêm hoặc bớt từ các mô đun nhân tại chế độ runtime, nên điều này làm cho Linux rất linh hoạt. Vì Linux là mã nguồn mở nên Linux không đòi hỏi người dùng phải trả tiền bản quyền. Yếu tố này đặc biệt quan trong đối với các nhà sản xuất và phát triển các sản phẩm điện tử tiêu dùng vì nó sẽ làm giảm giá thành và nâng cao tính cạnh tranh cho các sản phẩm

Cấu trúc Embedded Linux

Trước hết, để chạy một hệ Linux, phần cứng phải đáp ứng được được các yêu cầu sau: thứ nhất, Linux yêu cầu CPU tối thiểu phải là 32 bit, có chứa một đơn vị quản lý bộ nhớ (MMU). Thứ hai, phải đủ bộ nhớ RAM cung cấp cho hệ thống. Thứ ba, vào/ra (I/O) tối thiểu phải đủ cho việc debug. Cuối cùng, nhân phải có khả năng tải hoặc truy cập vào một hệ thống tập tin gốc (root filesystem) thông qua các thiết bị lưu trữ cố định hoặc kết nối thêm.



Hình 3.2 Cấu trúc Embedded Linux

Trong sơ đồ cấu trúc trên, ngay phía trên phần cứng là nhân. Nhân là thành phần lõi của hệ điều hành. Chức năng của nhân là để quản lý phần cứng một cách hiệu quả đồng thời cung cấp giao diện lập trình, qua đó các phần mềm sử dụng được phần cứng thông qua nhân. Cũng giống như nhân UNIX, nhân Linux điều khiển thiết bị, quản lý truy cập I/O, kiểm soát quá trình lập lịch (scheduling), thi hành việc chia sẻ bộ nhớ, xử lý phân phối các tín hiệu, và phục vụ các nhiệm vụ khác. Nó cũng cho phép ứng dụng sử dụng các API được cung cấp bởi một nhân có thể di chuyển được giữa các cấu trúc khác nhau được hỗ trợ bởi nhân đó với sự thay đổi nhỏ hoặc không thay đổi. Điều này thường xuyên được sử dụng đối với Linux, có thể nhận thấy khối thống nhất của các ứng dụng sẵn có trên tất cả các cấu trúc được hỗ trợ bởi Linux.

Trong nhân, có hai lớp dịch vụ chính cung cấp các chức năng theo yêu cầu của ứng dụng. Tương tác mức thấp (Low-level interfaces) là đặc trưng riêng cho các cấu hình phần cứng mà trên đó nhân chạy và qui định để kiểm soát trực tiếp tài nguyên phần cứng bằng cách sử dụng một phần cứng độc lập với API. Phía trên dịch vụ mức thấp được cung cấp bởi nhân, các thành phần mức cao cung cấp các abstractions phổ biến cho tất cả các hệ thống UNIX, bao gồm cả các tiến trình, tập tin, các socket, và tín hiệu.

Giữa hai mức độ của abstraction, nhân đôi khi cần phải gọi đến các thành phần phiên dịch (interpretation component) để hiểu và tương tác được với cấu trúc dữ liệu đi và đến một số thiết bị. chẳng hạn như hệ thống tập tin và giao thức mạng là những cấu trúc dữ liệu mà nhân cần phải hiểu và tương tác được để cấp quyền truy cập vào dữ liệu đi và đến.

Trong lúc hoạt động bình thường, nhân yêu cầu phải có ít nhất một cấu trúc filesystem đó là root filesystem. Từ filesystem này nhân tải các ứng dụng đầu tiên chạy trên hệ thống. Nhân cũng dựa vào filesystem này cho các hoạt động sau đó, chẳng hạn như tải mô đun và cung cấp mỗi tiến trình với một thư mục làm việc. hệ thống tập tin gốc lưu trữ và hoạt động từ thiết bị lưu trữ thực hoặc tải vào bộ nhớ RAM trong khi khởi động hệ thống và vận hành trên đó.

Thư viện được sử dụng bởi hầu hết các ứng dụng Linux là thư viện GNU C glibc, thư viện được liên kết động với các ứng dụng. Điều này cho phép nhiều ứng dụng có thể sử dụng chung một thư viện. Thư viện C được tìm thấy trên filesystem của hệ thống, chẳng hạn có thể tải thư viện một lần lên bộ nhớ RAM, các ứng dụng sẽ cùng chia sẻ thư viện này. Tuy nhiên trên một số hệ thống nhúng, khi mà chỉ có một phần của thư viện được sử dụng bởi một vài ứng dụng thì việc liên kết tĩnh giữa thư viện và ứng dụng sẽ tiết kiệm được bộ nhớ và đảm bảo được tính gọn nhẹ của hệ thống.

Công cụ phát triển Embedded Linux

Cũng như phát triển các phần mềm khác, để phát triển Embedded Linux cũng cần phải có compiler, linker, IDE, interpreter, và các công cụ khác. Các công cụ để phát triển Embedded Linux có thể tìm thấy dưới dạng mã nguồn mở, được cung cấp miễn phí hoặc sử dụng các công cụ được biên soạn bởi các công ty, cá nhân.

Cross-compiler

Một trong những công cụ quan trọng được sử dụng để xây dựng Embedded Linux đó là cross-compiler. Cross-compiler là trình biên dịch cho phép biên dịch mã nguồn thành các tập tin thực thi chạy được ở các môi trường khác nhau, chứ không chỉ môi trường mà trình biên dịch đang chạy trên đó. Như hình trên mô tả cross-compiler, có thể thấy rõ để biên dịch từ nền tảng x86 sang ARM thì cần phải có cross-compiler.

Trong dự án GNU, có xây dựng bộ công cụ GNU Toolchain. Bộ công cụ này sử dụng để phát triển các ứng dụng, các phần mềm cho hệ thống nhúng, hệ điều hành… chẳng hạn như Embedded Linux. GNU Toolchain gồm có: GNU make, GNU Compiler (GCC), GNU Binutils, GNU Bison, GNU mp4, GNU Debugger (GDB) và GNU build system (autotools).

3.2.2. Windows CE

Windows CE hay Windows Embedded CE là tên một hệ điều hành của Microsoft. Đây là một hệ điều hành nhúng mở, được sử dụng cho các hệ thống nhúng.

Windows CE mở ra khả năng phát triển rất lớn đối với các nhà phát triển ứng dụng bởi nó cung cấp rất nhiều bộ công cụ lập trình để tạo ra các mã quản lí và các ứng dụng mã máy cho các phần cứng cơ sở của Windows CE. Windows CE cung cấp cho các nhà phát triển ứng dụng một môi trường API 32 bit của Microsoft cùng với sự dễ dàng sử dụng và sự linh hoạt của ngôn ngữ kịch bản. Đồng thời nó cũng hỗ trợ cho các ứng dụng đa phương tiện, Internet, mạng nội bộ (LAN), truyền thông và các dịch vụ bảo mật...Vì vậy nên Windows CE được ứng dụng rất rộng rãi trên các thiệt bị điện tử cầm tay như điện thoại, máy chơi trò chơi, máy nghe nhạc và các sản phẩm trong công nghiệp như HMI, PLC.

Cấu trúc Windows CE

Windows CE là một hệ điều hành thời gian thực, hỗ trợ và chạy trên nhiều bộ xử lý khác nhau bao gồm ARM, MIPS, x86 và SH4. Windows CE cho phép 32000 tiến trình chạy đồng thời. Tuy nhiên, trên thực tế số lượng tiến trình còn phụ thuộc vào khả năng xử lý của hệ thống

Sơ đồ dưới đây mô tả cấu trúc của hệ điều hành Windows CE. Trong đó:

User Processes: Bao gồm các tiến trình riêng biệt tạo nên các ứng dụng người dùng, chẳng hạn như ứng dụng được gọi là user-mode server. Những ứng dụng này gồm có Udevice.exe, Servicesd.exe(là tiến trình tải các dịch vụ chẳng hạn HTTP, FTP,UPnP…).

Hệ thống API sẵn có cho các ứng dụng thông qua thư viện coredll.dll, chúng liên kết với tất cả các mô đun thực thi của hệ điều hành. Bên cạnh đó, hệ điều hành cung cấp các ứng dụng API tương tự như Win32 API trên máy tính để bàn. Người phát triển có thể sử dụng tính năng truy cập thông qua thư viện ứng dụng, chẳng hạn như Wininet.dll, Winsock.dll, Msxml.dll, và Winhttp.dll.

Nhân(Nhân) : được mô tả bởi mô đun NK.exe là lõi của hệ điều hành Windows CE. Nó cung cấp các chức năng cơ bản cho hệ điều hành. Các chức năng này bao gồm việc xử lý cơ sở dữ liệu và quản lý bộ nhớ. nhân cũng cung cấp một số chức năng quản lý tập tin, các dịch vụ cho phép các ứng dụng có thể sử dụng các chức năng của nhân.

Cấu trúc Windows CE

Phần cứng (Hardware): Nhân của Windows CE tương tác với phần cứng thông qua các trình điều khiển (driver). Sự kết hợp của lớp tương thích thiết bị gốc(OAL), driver, và các tập tin cấu hình cho một nền tảng phần cứng cụ thể có tên là gói hỗ trợ mạch(BSP)

Công cụ phát triển Windows CE

Windows CE bao gồm một bộ công cụ hỗ cho việc thiết kế và cấu hình OS images, phát triển các driver, dịch vụ và ứng dụng. Platform builder cho Windows CE 6.0 được plug-in trên Microsoft Visual Studio 2005(VS2005). Để phát triển Windows CE cần có VS2005 và Platform Builder. Việc sử dụng nền tảng VS2005 làm công cụ giúp cho việc phát triển Windows CE được dễ dàng hơn. Platform Builder được plug-in trên VS2005 cho phép xây dựng các BSP, tạo ra các driver, xây dựng runtime image và xuất ra các SDK để hỗ trợ phát triển các ứng dụng.

3.2.3. Android

Android lần đầu tiên ra mắt vào năm 2007, được phát triển bởi nhóm Open Handset Alliance. Android là một hệ điều hành dựa trên nhân Linux (nhân 2.6), các ứng dụng chạy trên máy ảo Java - phiên bản được thiết kế cho các dòng máy di động có tên Dalvik.

Các tính năng mà Android hỗ trợ rất rộng, bao gồm đồ họa 2D, 3D (dựa trên OPENGLES), định vị GPS, Bluetooth, EDGE, 3G, WiFi, hỗ trợ thoại GSM, dữ liệu được lưu trữ trong cơ sở dữ liệu SQLite...

Cấu trúc Android

Trong hình dưới đây có thể thấy rõ bên trong hệ điều hành Android có chứa Nhân Linux. Các thư viện là lớp nằm trên Nhân, tiếp đó là các framework và lớp trên cùng chính là những ứng dụng. Lớp thư viện chính là ngôi nhà để thực hiện các đoạn mã cho các thực thể như bộ xử lý đa phương tiện dùng để xem/ghi lại âm thanh và hình ảnh, Nhân của trình duyệt Web, tiến trình biên dịch kiểu chữ, và bộ máy cơ sở dữ liệu SQLite. Phần runtime của Android cũng trú ngụ tại lớp thư viện.

Nằm trên thư viện chính là các framework, đó là tập hợp các dịch vụ có thể dùng lại được và những thành phần chung phục vụ cho các ứng dụng. Ví dụ, một loại framework là thành phần cung cấp nội dung cho bất kỳ dịch vụ nào có liên quan đến việc líu trữ và truy xuất dữ liệu. Giao diện ứng dụng trong SQLite chính là một thí dụ cụ thể về phần cung cấp nội dung này.

Cấu trúc Android

Các ứng dụng chạy ở lớp trên cùng của hệ điều hành với một bộ các nhân ứng dụng bao gồm thư điện tử, lịch làm việc, trình duyệt web... Khi nhà phát triển viết một ứng dụng dành cho Android, đầu tiên thực hiện các đoạn mã trong môi trường Java. Sau đó, nó sẽ được biên dịch sang các bytecode của Java, tuy nhiên để thực thi được ứng dụng này trên Android thì nhà phát triển phải thực thi một công cụ có tên là dx. Đây là công cụ dùng để chuyển đổi bytecode sang một dạng gọi là dex bytecode. "Dex" là từ viết tắt của "Dalvik executable" đóng vai trò như cơ chế ảo thực thi các ứng dụng Android. Máy ảo Dalvik cũng giống như máy ảo Java (Java Virtual Machine)

Công cụ phát triển Android

Phiên bản Europa của Eclipse là nền tảng phát triển các ứng dụng. Ngoài ra, cần cài đặt ít nhất một bộ JDK 5 hoặc JDK 6 để có thể sử dụng các công cụ của Android. Tuy nhiên, cũng không bắt buộc phải dùng Eclipse để phát triển Android. Bên cạnh đó, Android SDK cung cấp các công cụ cho phép sử dụng các IDE khác. Ví dụ IDE IntelliJ được đề cập chi tiết trong tài liệu mô tả của Android.

Những nhà phát triển nhân cứng sẽ cảm thấy thoải mái khi làm việc với bộ công cụ command-line đi kèm SDK. Chẳng hạn, công cụ activityCreator (được cung cấp như là một tập tin batch file dành cho Windows và đóng vai trò như một script Python cho người dùng Mac và Linux) sẽ xây dựng framework cho các ứng dụng của Android. Việc thực thi activityCreator sẽ dựng lên các tập tin Java nòng cốt, từ đó sẽ tạo ra những thư mục con và các tập tin XML cần thiết. Công cụ này cũng hình thành một tập tin Ant dùng cho việc biên dịch mã nguồn và tạo các ứng dụng.

Những công cụ command-line khác trong SDK bao gồm logCat dùng để xuất các thông điệp ghi nhận tình trạng hệ thống. logCat rất hữu dụng trong việc ghi nhận thời điểm xảy ra lỗi. Nếu cần phân tích các lỗi một cách sâu hơn có thể dẫn nhập một class debug đặc biệt vào ứng dụng. Class này sẽ cung cấp các cách thức để bắt đầu và dừng việc tìm kiếm dấu vết. Khi ở trạng thái hoạt động, debug sẽ ghi nhận các sự kiện thành một tập tin, mà sau đó có thể được kiểm tra bằng ứng dụng TraceView.

Android Emulator

Cuối cùng là bộ mô phỏng Android, khi được khởi động nó sẽ hiển thị toàn bộ giao diện bao gồm cả các nút bấm và bàn phím QWERTY. Nó có thể hoạt động tốt tương tự như thiết bị thật dù cho các một vài giới hạn (ví dụ như không nhận được cuộc gọi đến). Bộ mô phỏng Android chạy một phiên bản đã được sửa đổi của môi trường giả lập mã nguồn mở thuộc Fabrice Bellard, có tên là QEMU. Phiên bản này giả lập bộ xử lý ARM và thực thi hệ điều hành Linux.

3.3. Lập trình C/C++ trên Linux

3.3.1 Linux và các lệnh cơ bản

Các khái niệm cơ bản

- *Users (Người dùng):* Để có thể sử dụng được Linux, bạn phải được cấp tài khoản (account) đăng nhập vào máy Linux. Thông tin về tài khoản bao gồm tên đăng nhập (username), mật khẩu đăng nhập (password), và các quyền truy xuất tập tin và thư mục mà bạn có được dựa vào tài khoản mà bạn đăng nhập và máy.

- *Group (Nhóm):* Các người dùng làm việc trên cùng một bộ phận hoặc đang làm việc chung trên cùng một dự án (project) có thể được đưa vào cùng một nhóm. Đây là một cách đơn giản của việc tổ chức để quản lí người dùng.

- *File (Tập tin):* Tất cả các thông tin trên Linux được lưu giữ trong các tập tin. Các tập tin được tạo ra bởi người dùng và người chủ tập tin có quyền truy xuất, tạo, sửa đổi, thiết lập kích thước của tập tin và phân phối quyền để cho phép người dùng khác có thể truy xuất tập tin.

- *Directory (Thư mục):* Thư mục giống như Folder trong Windows. Nó được dùng để chứa các tập tin và thư mục khác, và tạo ra cấu trúc cho hệ thống tập tin. Dưới Linux, chỉ có một cây thư mục và gốc của nó là /. Giống như tập tin, mỗi thư mục có thông tin kết hợp với nó, kích thước tối đa và những người dùng được quyền truy xuất thư mục này, …

- *Path (Đường dẫn):* Đường dẫn là 1 chuỗi các thư mục và có thể kết thúc bằng tên của một tập tin. Các thư mục và tên tập tin được phân cách bởi kư tự /. Ví dụ : /dir1/dir2/file là một đường dẫn tuyệt đối tới file được chứa trong dir2, với dir2 được chứa trong dir1, và dir1 nằm trong thư mục gốc. Ví dụ khác: ~/homework là một đường dẫn tương đối, tính từ thư mục đăng nhập của người dùng, vào thư mục homework.

- *Permissions (Quyền):* Quyền là một đặc tính quan trọng của Linux. Chúng tạo ra sự bảo mật bằng cách giới hạn các hành động mà người dùng có thể thực hiện đối với tập tin và thư mục. Các quyền đọc (read), ghi (write) và thực thi (execute) điều khiển việc truy xuất tới việc truy xuất tập tin của người tạo ra nó, nhóm và các người dùng khác. Một người dùng sẽ không thể truy xuất tới tập tin của người dùng khác nếu không có đủ quyền truy xuất.

- *Process (Tiến trình):* Khi người dùng thực thi một lệnh, Linux tạo ra một tiến trình chứa các chỉ thị lệnh. Một tiến trình còn chứa các thông tin điều khiển như thông tin người dùng thực thi lệnh, định danh duy nhất của tiến trình (PID – process id). Việc quản lí của tiến trình dựa trên PID này.

- *Shell*: Trong chế độ console, người dùng giao tiếp với máy thông qua shell (hệ vỏ). Một shell là một chương trình thường được dùng để bắt đầu một chương trình khác từ dấu nhắc của shell. Một shell được cấu hình bằng việc thiết lập các biến môi trường cho nó. Khi đăng nhập vào Linux, một shell sẽ được tự động tạo ra, và các biến

môi trường mặc nhiên (default) sẽ được thiết lập. Ở đây, ta sẽ sử dụng shell BASH (Bourne Again SHell), là shell thông dụng của hầu hết các hệ thống Linux.

Thực thi Lệnh

- *Nhập lệnh*: Để nhập lệnh, đơn giản bạn chỉ đánh vào tên của lệnh sau dấu nhắc của shell rồi nhấn Enter. Dấu nhắc của shell thường có dạng [user@host directory]$, nó có thể được thiết lập lại, và có thể khác nhau đối với các máy khác nhau. Hầu hết các lệnh thường chấp nhận nhiều đối số (argument) hoặc lựa chọn (option) (thường được gọi là flag – cờ). Thông thường các đối số được đưa vào bằng cách sử dụng 1 hoặc 2 dấu -. Nếu một lệnh yêu cầu đối số và chúng ta không đưa vào, lệnh sẽ tự động hiển thị một mô tả ngắn về cách sử dụng các đối số kết hợp với nó. Một lệnh và các đối số thường có dạng như sau:

command –a1 –a2

command --long_argument_name

- *Biến môi trường PATH:* Đây là biến môi trường của shell mà cho phép các thư mục mà Linux có thể nhìn thấy được khi thực thi lệnh nếu đường dẫn đầy đủ của lệnh không được chỉ định rõ ràng. Biến môi trường PATH bao gồm 1 chuỗi tên các đường dẫn thư mục, phân cách bởi dấu ‘:’. Hầu hết các lệnh mà chúng ta sẽ thực hành đều nằm trong các thư mục mà đã được đưa vào biến môi trường PATH và có thể thực hiện đơn giản bằng cách nhập tên của nó tại dấu nhắc lệnh. Vìlí do bảo mật, thư mục hiện hành sẽ không được đưa vào biến môi trường PATH, do đó, để chạy một chương trình nằm trong thư mục hiện hành, chúng ta phải thêm ‘./’ vào trước tên chương trình:

./command

Một số lệnh cơ bản

- *Gọi sự trợ giúp*: Hầu hết các console Linux đều chứa một chương trình tiện ích nhỏ để in ra màn hình thông tin về cách sử dụng lệnh khi một cờ ‘-h’ hoặc ‘--help’ được truyền vào cho chúng. Ngoài ra, chúng ta có thể sử dụng lệnh man (manual) để tìm hiểu về một lệnh.

command –h	Hiển thị thông tin trợ giúp ngắn gọn về lệnh.
command -–help	Hiển thị thông tin trợ giúp ngắn gọn về lệnh.
man command	Hiển thị trang trợ giúp đầy đủ của lệnh.

- *Các lệnh liệt kê tập tin (file):* Một trong những tác vụ cơ bản mà chúng ta có thể thực hiện là liệt kê các tập tin nằm trong một thư mục với lệnh ‘ls’ Lệnh này cho phép kiểm tra nội dung của thư mục và tìm kiếm tập tin mà chúng ta muốn làm việc. Nếu các tập tin liệt kê tràn quá một màn hình, chúng ta có thể kết hợp với đường ống (pipe) để xuất kết quả của lệnh ‘ls’ đến một chương trình hiển thị văn bản như ‘less’ chẳng hạn.

ls	Liệt kê nội dung của thư mục hiện hành.
ls –a	Liệt kê tất cả tập tin, kể cả các tập tin có thuộc tính ẩn.

ls –l Hiển thị đầy đủ các thông tin (quyền truy cập, chủ, kích thước, …)

ls | less

- *Thay đổi thư mục:* Khi bạn đăng nhập vào Linux, chúng ta được tự động đặt vào thư mục tiếp nhận (home directory) của chúng ta. Để chuyển tới thư mục khác, dùng lệnh 'cd'. Lệnh 'cd' nhận đối số là một đường dẫn tương đối hoặc tuyệt đốicủa thư mục hiện hành, hoặc một số các đối số đặc biệt như dưới đây:

cd path Chuyển đến thư mục được chỉ định bởi path.

cd ~ Chuyển về thư mục nhà.

cd - Chuyển về thư mục trước của bạn.

cd .. Chuyển về thư mục cha của thư mục hiện hành.

- *Quản lí tập tin và thư mục:*

cp Cho phép tạo ra một bản sao (copy) của một tập tin hoặc thư mục: cp source_path destination_path

mkdir Cho phép tạo ra một thư mục mới (make directory), rỗng, tại vị trí được chỉ định: mkdir directoryname

mv Cho phép di chuyển (move) một tập tin từ thư mục này tới thư mục khác, có thể thực hiện việc đổi tên tập tin:

mv source_path destination_path

rm Cho phép xóa (remove) các tập tin, dùng lệnh 'rm – R' để xóa một thư mục và tất cả những gì nằm trong nó: rm filename

rmdir Dùng để xóa thư mục: rmdir directoryname

touch Tạo tập tin trống: touch filename

- *Xác định vị trí của tập tin:* Khi các tập tin của chúng ta nằm trên nhiều thư mục, hoặc chúng ta cần tìm kiếm một tập tin nào đó, chúng ta có thể sử dụng lệnh 'find' và 'locate'. Lệnh 'find' bắt đầu từ thư mục được chỉ định và sẽ tìm trong tất cả các thư mục con trong đó. Lệnh 'locate' thì tạo ra và duy trì một cơ sở dữ liệu về các tập tin trong hệ thống, và nó đơn giản chỉ tìm trong cơ sở dữ liệu này xem có tập tin cần tìm. Lệnh 'locate' thực hiện nhanh hơn lệnh 'find', nhưng cơ sở dữ liệu của nó chỉ cập nhật một lần trong ngày nên những tập tin mới được tạo ra có thể không được tìm thấy.

find Tìm tập tin filename bắt đầu từ thư mục path: find path –name filename

locate Tìm tập tin trong cơ sở dữ liệu của nó có tên là filename: locate filename

- Làm việc với tập tin văn bản:

cat Để xem nội dung của một tập tin văn bản ngắn, chúng ta dùng lệnh 'cat' để in nó ra màn hình: cat filename

less Cho phép xem một tập tin dài bằng cách cuộn lên xuống bằng các phím mũi tên và các phím pageUp, pageDown. Dùng phím q để thoát chế độ xem: less filename

grep Một công cụ mạnh để tìm một chuỗi trong một tập tin văn bản. Khi lệnh 'grep' tìm thấy chuỗi, nó sẽ in ra cả dựng đó lên màn hình:

grep string filename

sort Sắp xếp các dựng trong tập tin theo thứ tự alphabet và in nội dung ra màn hình: sort filename

- Giải nén:

bunzip2 Giải nén một tập tin bzip2 (*.bz2). Thường dùng cho các tập tin lớn: bunzip2 filename.bz2

gunzip Giải nén một tập tin gzipped (*.gz): gunzip filename.gz

unzip Giải nén một tập tin PkZip hoặc WinZip (*.zip): unzip filename.zip

tar Nén và giải nén các tập tin .tar, .tar.gz: Ví dụ: tar –xvf filename.tar và tar –xvzf filename.tar.gz

- Xem thông tin hệ thống: Các lệnh sau đây hiển thị các thông tin khác trên hệ thống của chúng ta.

date In ngày giờ hệ thống.

df –h In thông tin không gian đĩa được dùng.

free In thông tin bộ nhớ được dùng.

history Hiển thị các lệnh được thực hiện bởi tài khoản hiện tại.

hostname In tên của máy cục bộ (host).

pwd In đường dẫn đến thư mục làm việc hiện hành.

rwho -a Liệt kê tất cả người dùng đã đăng nhập vào network.

uptime In thời gian kể từ lần reboot gần nhất.

who Liệt kê tất cả người dùng đã đăng nhập vào máy.

whoami In tên người dùng hiện hành.

- Các lệnh dùng theo dơi tiến trình:

ps Liệt kê các tiến trình đang kích hoạt bởi người dùng và PID của các tiến trình đó.

ps –aux Liệt kê các tiến trình đang kích hoạt cùng với tên của người dùng là chủ tiến trình.

top Hiển thị danh sách các tiến trình đang kích hoạt, danh sách này được cập nhật liên tục.

command & Chạy command trong nền.

fg Đẩy một tiến trình nền hoặc bị dừng lên bề mặt trở lại.

bg Chuyển một tiến trình vào nền. Có thể thực hiện tương tự với Ctrl-z.

kill pid Thúc đẩy tiến trình kết thúc. Đầu tiên phải xác định pid của tiến trình cần hủy với lệnh ps.

killall -9 name Hủy tiến trình với name chỉ định.

nice program level Chạy program với cấp ưu tiên ngược level. Cấp nice càng cao, chương trình càng có mức ưu tiên thấp

3.3.2 Chương trình trên Linux

Để có thể viết chương trình trên Linux, chúng ta cần phải nắm rõ 1 số vị trí tài nguyên để xây dựng chương trình như trình biên dịch, tập tin thư viện, các tập tin tiêu đề (header), các tập tin chương trình sau khi biên dịch, …

Trình biên dịch gcc thường được đặt trong thư mục /usr/bin hoặc /usr/local/bin (kiểm tra bằng lệnh which gcc). Tuy nhiên, khi biên dịch, gcc cần đến rất nhiều tập tin hỗ trợ nằm trong những thư mục khác nhau như những tập tin tiêu đề (header) của C thường nằm trong thư mục /usr/include hay /usr/local/include. Các tập tin 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 sau khi biên dịch ra tập tin thực thi (dạng nhị phân) có thể đặt bất cứ vị trí nào trong hệ thống.

Các tập tin tiêu đề (header)

Các tập tin tiêu đề trong C thường định nghĩa hàm và khai báo cần thiết cho quá trình biên dịch. Hầu hết các chương trình trên Linux khi biên dịch sử dụng các tập tin tiêu đề trong thư mục /usr/include hoặc các thư mục con bên trong thư mục này, ví dụ: /usr/include/sys. Một số khác được trình biên dịch dò tìm mặc định như /usr/include/X11 đối với các khai báo hàm lập trình đồ họa X-Window, hoặc /usr/include/g++-2 đối với trình biên dịch GNU g++.

Tuy nhiên, nếu chúng ta có các tập tin tiêu đề của riêng mình trong một thư mục khác thư mục mặc định của hệ thống thì chúng ta có thể chỉ rõ tường minh đường dẫn đến thư mục khi biên dịch bằng tùy chọn –I, ví dụ:

$ gcc –I/usr/mypro/include test.c –otest

Khi chúng ta sử dụng một hàm nào đó của thư viện hệ thống trong chương trình C, ngoài việc phải biết khai báo nguyên mẫu của hàm, chúng ta cần phải biết hàm này được định nghĩa trong tập tin tiêu đề nào. Trình man sẽ cung cấp cho chúng ta các thông tin này rất chi tiết. Ví dụ, khi dùng man để tham khảo thông tin về hàm kill(), chúng ta sẽ thấy rằng cần phải khai báo 2 tập tin tiêu đề là types.h và signal.h.

Các tập tin thư viện

Các tập tin tiêu đề của C chỉ cần thiết để trình biên dịch bắt lỗi cú pháp, kiểm tra kiểu dữ liệu của chương trình và tạo ra các tập tin đối tượng. Muốn tạo ra chương trình thực thi, chúng ta cần phải có các tập tin thư viện. Trong Linux, các tập tin thư viện tĩnh của C có phần mở rộng là .a, .so, .sa và bắt đầu bằng tiếp đầu ngữ lib. Ví dụ libutil.a hay libc.so là tên các thư viện liên kết trong Linux.

Linux có hai loại liên kết là liên kết tĩnh (static) và liên kết động (dynamic). Thư viện liên kết động trên Linux thường có phần mở rộng là .so, chúng ta có thể dùng lệnh ls /usr/lib hoặc ls /lib để xem các thư viện hệ thống đang sử dụng. Khi biên dịch, thông thường trình liên kết (ld) sẽ tìm thư viện trong 2 thư viện chuẩn /usr/lib và /lib. Để chỉ định tường minh một thư viện nào đó, chúng ta làm như sau:

$ gcc test.c –otest /usr/lib/libm.a

Bởi vì thư viện bắt buộc phải có tiếp đầu ngữ lib và có phần mở rộng là .a hoặc .so, trình biên dịch cho phép chúng ta sử dụng tùy chọn –l ngắn gọn như sau:

$ gcc test.c –otest -lm

chúng ta sẽ thấy rằng gcc sẽ mở rộng –l thành tiếp đầu ngữ lib và tìm libm.a hoặc libm.so trong thư mục chuẩn để liên kết.

Mặc dù vậy, không phải lúc nào thư viện của chúng ta cũng phải nằm trong thư viện của Linux. Nếu thư viện của chúng ta nằm ở một thư mục khác, chúng ta có thể chỉ định gcc tìm kiếm trực tiếp với tùy chọn –L như sau:

$ gcc test.c –otest -L/usr/myproj/lib -ltool

Lệnh trên cho phép liên kết với thư viện libtool.a hoặc libtool.so trong thư mục /usr/myproj/lib.

Thư viện liên kết trên Linux

Hình thức đơn giản nhất của thư viện là tập hợp các tập tin .o do trình biên dịch tạo ra ở bước biên dịch với tùy chọn –c. Ví dụ

$gcc –c helloworld.c

trình biên dịch chưa tạo ra tập tin thực thi mà tạo ra tập tin đối tượng helloworld.o. Tập tin này chứa các mã máy của chương trình đã được sắp xếp lại. Nếu muốn tạo ra tập tin thực thi, chúng ta gọi trình biên dịch thực hiện bước liên kết:

$gcc helloworld.o –o helloworld

Trình biên dịch sẽ gọi tiếp trình liên kết ld tạo ra định dạng tập tin thực thi cuối cùng. Ở đây, nếu chúng ta không sử dụng tùy chọn –c, trình biên dịch sẽ thực hiện cả hai bước đồng thời.

Thư viện liên kết tĩnh

Thư viện liên kết tĩnh là các thư viện khi liên kết trình biên dịch sẽ lấy toàn bộ mã thực thi của hàm trong thư viện đưa vào chương trình chính. Chương trình sử dụng thư viện liên kết tĩnh chạy độc lập với thư viện sau khi biên dịch xong. Nhưng khi nâng cấp và sửa đổi, muốn tận dụng những chức năng mới của thư viện thì chúng ta phải biên dịch lại chương trình.

Ví dụ sử dụng liên kết tĩnh:

```
/* cong.c */
int cong( int a, int b )
{
return a + b;
}

/* nhan.c */
long nhan( int a, int b )
{
return a * b;
}
```

Thực hiện biên dịch để tạo ra hai tập tin thư viện đối tượng .o

```
$ gcc –c cong.c nhan.c
```

Để một chương trình nào đó gọi được các hàm trong thư viện trên, chúng ta cần tạo một tập tin header .h khai báo các nguyên mẫu hàm để người sử dụng triệu gọi:

```
/* lib.h */
int cong( int a, int b );
long nhan( int a, int b );
```

Cuối cùng, tạo ra chương trình chính program.c triệu gọi hai hàm này.

```
/* program.c */
#include <stdio.h>
#include "lib.h"

int main ()
{
int a, b;
```

printf("Nhap vào a : ");

scanf("%d", &a);

printf("Nhap vào b : ");

scanf("%d", &b);

printf("Tổng %d + %d = %d\n", a, b, cong(a, b));

*printf("Tich %d * %d = %ld\n", a, b, nhan(a, b));*

return (0);

}

- Chúng ta biên dịch và liên kết với chương trình chính như sau:

$ gcc –c program.c

$ gcc program.o cong.o nhan.o -oprogram

Sau đó thực thi chương trình

$ ./program

Ở đây .o là các tập tin thư viện đối tượng. Các tập tin thư viện .a là chứa một tập hợp các tập tin .o. Tập tin thư viện .a thực ra là 1 dạng tập tin nén được tạo ra bởi chương trình ar. Chúng ta hãy yêu cầu ar đóng cong.o và nhan.o vào libfoo.a

$ ar cvr libfoo.a cong.o nhan.o

Sau khi đã có được thư viện libfoo.a, chúng ta liên kết lại với chương trình theo cách sau:

$ gcc program.o –oprogram libfoo.a

Chúng ta có thể sử dụng tùy chọn –l để chỉ định thư viện khi biên dịch thay cho cách trên. Tuy nhiên libfoo.a không nằm trong thư mục thư viện chuẩn, cần phải kết hợp với tùy chọn –L để chỉ định đường dẫn tìm kiếm thư viện trong thư mục hiện hành. Dưới đây là cách biên dịch:

$ gcc program.c –oprogram –L –lfoo

Chúng ta có thể sử dụng lệnh nm để xem các hàm đã biên dịch sử dụng trong tập tin chương trình, tập tin đối tượng .o hoặc tập tin thư viện .a. Ví dụ:

$ nm cong.o

Thư viện liên kết động

Khuyết điểm của thư viện liên kết tĩnh là nhúng mã nhị phân kèm theo chương trình khi biên dịch, do đó tốn không gian đĩa và khó nâng cấp. Thư viện liên kết động được dùng để giải quyết vấn đề này. Các hàm trong thư viện liên kết động không trực tiếp đưa vào chương trình lúc biên dịch và liên kết, trình liên kết chỉ lưu thông tin tham chiếu đến các hàm trong thư viện liên kết động. Vào lúc chương trình nhị phân thực thi, Hệ Điều Hành sẽ nạp các chương trình liên kết cần tham chiếu vào bộ nhớ.

Như vậy, nhiều chương trình có thể sử dụng chung các hàm trong một thư viện duy nhất.

- Tạo thư viện liên kết động:

Khi biên dịch tập tin đối tượng để đưa vào thư viện liên kết động, chúng ta phải thêm tùy chọn –fpic (PIC- Position Independence Code – mã lệnh vị trí độc lập).

Ví dụ: biên dịch lại 2 tập tin cong.c và nhan.c

$ gcc –c –fpic cong.c nhan.c

Để tạo ra thư viện liên kết động, chúng ta không sử dụng trình ar như với thư viện liên kết tĩnh mà dùng lại gcc với tùy chọn –shared.

$ gcc –shared cong.o nhan.o -olibfoo.so

Nếu tập tin libfoo.so đã có sẵn trước thì không cần dùng đến tùy chọn –o

$ gcc –shared cong.o nhan.o libfoo.so

Bây giờ chúng ta đã có thư viện liên kết động libfoo.so. Biên dịch lại chương trình như sau:

$ gcc program.c –oprogram –L. –lfoo

- Sử dụng thư viện liên kết động:

Khi Hệ Điều Hành nạp chương trình program, nó cần tìm thư viện libfoo.so ở đâu đó trong hệ thống. Ngoài các thư mục chuẩn, Linux còn tìm thư viện liên kết động trong đường dẫn của biến môi trường LD_LIBRARY_PATH. Do libfoo.so đặt trong thư mục hiện hành, không nằm trong các thư mục chuẩn nên ta cần đưa thư mục hiện hành vào biến môi trường LD_LIBRARY_PATH:

$ LD_LIBRARY_PATH=.:

$ export LD_LIBRARY_PATH

Kiểm tra xem Hệ Điều Hành có thể tìm ra tất cả các thư viện liên kết động mà chương trình sử dụng hay không:

$ ldd program

rồi chạy chương trình sử dụng thư viện liên kết động này:

$./program

Một khuyết điểm của việc sử dụng thư viện liên kết động đó là thư viện phải tồn tại trong đường dẫn để Hệ Điều Hành tìm ra khi chương trình được triệu gọi. Nếu không tìm thấy thư viện, Hệ Điều Hành sẽ chấm dứt ngay chương trình cho dù các hàm trong thư viện chưa được sử dụng. Ta có thể chủ động nạp và gọi các hàm trong thư viện liên kết động mà không cần nhờ vào Hệ Điều Hành bằng cách gọi hàm liên kết muộn.

3.3.3 Xử lý tiến trình trong linux

Khái quát

Một trong những đặc điểm nổi bật của Linux là khả năng chạy đồng thời nhiều chương trình. Hệ Điều Hành xem mỗi đơn thể mã lệnh mà nó điều khiển là tiến trình (process). Một chương trình có thể bao gồm nhiều tiến trình kết hợp với nhau.

Đối với Hệ Điều Hành, các tiến trình cùng hoạt động chia sẻ tốc độ xử lí của CPU, cùng dùng chung vùng nhớ và tài nguyên hệ thống khác. Các tiến trình được điều phối xoay vòng bởi Hệ Điều Hành. Một chương trình của chúng ta nếu mở rộng dần ra, sẽ có lúc cần phải tách ra thành nhiều tiến trình để xử lí những công việc độc lập với nhau. Các lệnh của Linux thực tế là những lệnh riêng lẻ có khả năng kết hợp và truyền dữ liệu cho nhau thông qua các cơ chế như : đường ống pipe, chuyển hướng xuất nhập (redirect), phát sinh tín hiệu (signal), … Chúng được gọi là cơ chế giao tiếp liên tiến trình (IPC – Inter Process Comunication). Đối với tiến trình, chúng ta sẽ tìm hiểu cách tạo, hủy, tạm dừng tiến trình, đồng bộ hóa tiến trình và giao tiếp giữa các tiến trình với nhau.

Xây dựng ứng dụng trong môi trường đa tiến trình như Linux là công việc khó khăn. Không như môi trường đơn nhiệm, trong môi trường đa nhiệm tiến trình có tài nguyên rất hạn hẹp. Tiến trình của chúng ta khi hoạt động phải luôn ở trạng thái tôn trọng và sẵn sàng nhường quyền xử lí CPU cho các tiến trình khác ở bất kỳ thời điểm nào, khi hệ thống có yêu cầu. Nếu tiến trình của chúng ta được xây dựng không tốt, khi đổ vỡ và gây ra lỗi, nó có thể làm treo các tiến trình khác trong hệ thống hay thậm chí phá vỡ (crash) Hệ Điều Hành.

Định nghĩa của tiến trình: là một thực thể điều khiển đoạn mã lệnh có riêng một không gian địa chỉ, có ngăn xếp stack riêng rẽ, có bảng chứa các thông số mô tả file được mở cùng tiến trình và đặc biệt có một định danh PID (Process Identify) duy nhất trong toàn bộ hệ thống vào thời điểm tiến trình đang chạy.

Như chúng ta đã thấy, tiến trình không phải là một chương trình (tuy đôi lúc một chương trình đơn giản chỉ cần một tiến trình duy nhất để hoàn thành tác vụ, trong trường hợp này thì chúng ta có thể xem tiến trình và chương trình là một). Rất nhiều tiến trình có thể thực thi trên cùng một máy với cùng một Hệ Điều Hành, cùng một người dùng hoặc nhiều người dùng đăng nhập khác nhau. Ví dụ shell bash là một tiến trình có thể thực thi lệnh ls hay cp. Bản thân ls, cp lại là những tiến trình có thể hoạt động tách biệt khác.

Trong Linux, tiến trình được cấp không gian địa chỉ bộ nhớ phẳng là 4GB. Dữ liệu của tiến trình này không thể đọc và truy xuất được bởi các tiến trình khác. Hai tiến trình khác nhau không thể xâm phạm biến của nhau. Tuy nhiên, nếu chúng ta muốn chia sẻ dữ liệu giữa hai tiến trình, Linux có thể cung cấp cho chúng ta một vùng không gian địa chỉ chung để làm điều này.

Cách hoạt động của tiến trình

Khi 1 chương trình đang chạy từ dựng lệnh, chúng ta có thể nhấn phím Ctrl+z để tạm dùng chương trình và đưa nó vào hoạt động phía hậu trường (background). Tiến trình của Linux có các trạng thái:

- *Đang chạy (running)* : đây là lúc tiến trình chiếm quyền xử lí CPU dùng tính toán hay thực các công việc của mình.

-*Chờ (waiting)* : tiến trình bị Hệ Điều Hành tước quyền xử lí CPU, và chờ đến lượt cấp phát khác.

-*Tạm dừng (suspend)* : Hệ Điều Hành tạm dừng tiến trình. Tiến trình được đưa vào trạng thái ngủ (sleep). Khi cần thiết và có nhu cầu, Hệ Điều Hành sẽ đánh thức (wake up) hay nạp lại mã lệnh của tiến trình vào bộ nhớ. Cấp phát tài nguyên CPU để tiến trình tiếp tục hoạt động.

Trên dựng lệnh, thay vì dùng lệnh Ctrl+z, chúng ta có thể sử dụng lệnh bg để đưa một tiến trình vào hoạt động phía hậu trường. Chúng ta cũng có thể yêu cầu 1 tiến trình chạy nền bằng cú pháp &. Ví dụ: *$ls –R &*

Lệnh fg sẽ đem tiến trình trở về hoạt động ưu tiên phía trước. Thực tế khi chúng ta đăng nhập vào hệ thống và tương tác trên dựng lệnh, cũng là lúc chúng ta đang ở trong tiến trình shell của bash. Khi gọi một lệnh có nghĩa là chúng ta đã yêu cầu bash tạo thêm một tiến trình con thực thi khác. Về mặt lập trình, chúng ta có thể dùng lệnh fork() để nhân bản tiến trình mới từ tiến trình cũ. Hoặc dùng lệnh system() để triệu gọi một tiến trình của Hệ Điều Hành. Hàm exec() cũng có khả năng tạo ra tiến trình mới khác.

Cấu trúc tiến trình

Chúng ta hãy xem Hệ Điều Hành quản lí tiến trình như thế nào?

Nếu có hai người dùng: user1 và user2 cùng đăng nhập vào chạy chương trình grep đồng thời, thực tế, Hệ Điều Hành sẽ quản lí và nạp mã của chương trình grep vào hai vùng nhớ khác nhau và gọi mỗi phân vùng như vậy là tiến trình. Hình sau cho thấy cách phân chia chương trình grep thành hai tiến trình cho hai người khác nhau sử dụng

Trong hình này, user1 chạy chương trình grep tìm chuỗi abc trong tập tin file1.

$grep abc file1

user2 chạy chương trình grep và tìm chuỗi cde trong tập tin file2.

$grep cde file2

Chúng ta cần ta cần nhớ là hai người dùng user1 và user2 có thể ở hai máy tính khác nhau đăng nhập vào máy chủ Linux và gọi grep chạy đồng thời. Hình trên là hiện trạng không gian bộ nhớ Hệ Điều Hành Linux khi chương trình grep phục vụ người dùng.

Nếu dùng lệnh ps, hệ thống sẽ liệt kê cho chúng ta thông tin về các tiến trình mà Hệ Điều Hành đang kiểm soát, Ví dụ: $ps –af

Mỗi tiến trình được gán cho một định danh để nhận dạng gọi là PID (process identify). PID thường là số nguyên dương có giá trị từ 2-32768. Khi một tiến trình mới yêu cầu khởi động, Hệ Điều Hành sẽ chọn lấy một số (chưa bị tiến trình nào đang chạy chiếm giữ) trong khoảng số nguyên trên và cấp phát cho tiến trình mới. Khi tiến trình chấm dứt, hệ thống sẽ thu hồi số PID để cấp phát cho tiến trình khác trong lần sau. PID bắt đầu từ giá trị 2 bởi vì giá trị 1 được dành cho tiến trình đầu tiên gọi là init. Tiến trình init được và chạy ngay khi chúng ta khởi động Hệ Điều Hành. init là tiến trình quản lí và tạo ra mọi tiến trình con khác. Ở ví dụ trên, chúng ta thấy lệnh ps –af sẽ hiển thị 2 tiến trình grep chạy bởi user1 và user2 với số PID lần lượt là 101 và 102.

Mã lệnh thực thi của lệnh grep chứa trong tập tin chương trình nằm trên đĩa cứng được Hệ Điều Hành nạp vào vùng nhớ. Như chúng ta đã thấy ở lược đồ trên, mỗi tiến trình được Hệ Điều hành phân chia rõ ràng: vùng chứa mã lệnh (code) và vùng chứa dữ liệu (data). Mã lệnh thường là giống nhau và có thể sử dụng chung. Linux quản lí cho phép tiến trình của cùng một chương trình có thể sử dụng chung mã lệnh của nhau.

Thư viện cũng vậy. Trừ những thư viện đặc thù còn thì các thư viện chuẩn sẽ được Hệ Điều Hành cho phép chia sẻ và dùng chung bởi mọi tiến trình trong hệ thống. Bằng cách chia sẻ thư viện, kích thước chương trình giảm đi đáng kể. Mã lệnh của chương trình khi chạy trong hệ thống ở dạng tiến trình cũng sẽ đỡ tốn bộ nhớ hơn.

Trừ mã lệnh và thư viện có thể chia sẻ, còn dữ liệu thì không thể chia sẻ bởi các tiến trình. Mỗi tiến trình sở hữu phân đoạn dữ liệu riêng. Ví dụ tiến trình grep do user1 nắm giữ lưu giữ biến s có giá trị là 'abc', trong khi grep do user2 nắm giữ lại có biến s với giá trị là 'cde'.

Mỗi tiến trình cũng được hệ thống dành riêng cho một bảng mô tả file (file description table). Bảng này chứa các số mô tả áp đặt cho các file đang được mở. Khi mỗi tiến trình khởi động, thường Hệ Điều Hành sẽ mở sẵn cho chúng ta 3 file : stdin (số mô tả 0), stdout (số mô tả 1), và stderr (số mô tả 2). Các file này tượng trưng cho các thiết bị nhập, xuất, và thông báo lỗi. Chúng ta có thể mở thêm các file khác. Ví dụ user1 mở file file1, và user2 mở file file2. Hệ Điều Hành cấp phát số mô tả file cho mỗi tiến trình và lưu riêng chúng trong bảng mô tả file của tiến trình đó.

Ngoài ra, mỗi tiến trình có riêng ngăn xếp stack để lưu biến cục bộ và các giá trị trả về sau lời gọi hàm. Tiến trình cũng được dành cho khoảng không gian riêng để lưu các biến môi trường. Chúng ta sẽ dùng lệnh putenv và getenv để đặt riêng biến môi trường cho tiến trình.

Bảng thông tin tiến trình

Hệ Điều Hành lưu giữ một cấu trúc danh sách bên trong hệ thống gọi là bảng tiến trình (process table). Bảng tiến trình quản lí tất cả PID của hệ thống cùng với thông tin chi tiết về các tiến trình đang chạy. Ví dụng khi chúng ta gọi lệnh ps, Linux thường

đọc thông tin trong bảng tiến trình này và hiển thị những lệnh hay tên tiến trình được gọi: thời gian chiếm giữ CPU của tiến trình, tên người sử dụng tiến trình, …

Xem thông tin của tiến trình

Lệnh ps của Hệ Điều Hành dùng để hiển thị thông tin chi tiết về tiến trình. Tùy theo tham số, ps sẽ cho biết thông tin về tiến trình người dùng, tiến trình của hệ thống hoặc tất cả các tiến trình đang chạy. Ví dụ ps sẽ đưa ra chi tiết bằng tham số -af

Trong các thông tin do ps trả về, UID là tên của người dùng đã gọi tiến trình, PID là số định danh mà hệ thống cấp cho tiến trình, PPID là số định danh của tiến trình cha (parent PID). Ở đây chúng ta sẽ gặp một số tiến trình có định danh PPID là 1, là định danh của tiến trình init, được gọi chạy khi hệ thống khởi động. Nếu chúng ta hủy tiến trình init thì Hệ Điều Hành sẽ chấm dứt phiên làm việc. STIME là thời điểm tiến trình được đưa vào sử dụng. TIME là thời gian chiếm dụng CPU của tiến trình. CMD là toàn bộ dựng lệnh khi tiến trình được triệu gọi. TTY là màn hình terminal ảo nơi gọi thực thi tiến trình. Như chúng ta đã biết, người dùng có thể đăng nhập vào hệ thống Linux từ rất nhiều terminal khác nhau để gọi tiến trình. Để liệt kê các tiến trình hệ thống, chúng ta sử dụng lệnh: $ps –ax

Tạo lập tiến trình

Gọi tiến trình mới bằng hàm system()

Chúng ta có thể gọi một tiến trình khác bên trong một chương trình đang thực thi bằng hàm system(). Có nghĩa là chúng ta có thể tạo ra một tiến trình mới từ một tiến trình đang chạy. Hàm system() được khai báo như sau:

```
#include <stdlib.h>

int system( const char (cmdstr) )
```

Hàm này gọi chuỗi lệnh cmdstr thực thi và chờ lệnh chấm dứt mới quay về nơi gọi hàm. Nó tương đương với việc bạn gọi shell thực thi lệnh của hệ thống:

```
$sh –c cmdstr
```

system() sẽ trả về mã lỗi 127 nếu như không khởi động được shell để gọi lệnh cmdstr. Mã lỗi -1 nếu gặp các lỗi khác. Còn lại, mã trả về của system() là mã lỗi do cmdstr sau khi lệnh được gọi trả về.

Ví dụ sử dụng hàm system(), system.c

```
#include <stdlib.h>

#include <stdio.h>

int main()

{

printf( "Thuc thi lenh ps voi system\n" );

system( "ps –ax" );
```

printf("Thuc hien xong. \n");

exit(0);

}

Hàm system() của chúng ta được sử dụng để gọi lệnh "ps –ax" của Hệ Điều Hành.

Thay thế tiến trình hiện hành với các hàm exec

Mỗi tiến trình được Hệ Điều Hành cấp cho 1 không gian nhớ tách biệt để tiến trình tự do hoạt động. Nếu tiến trình A của chúng ta triệu gọi một chương trình ngoài B (bằng hàm system()chẳng hạn), Hệ Điều Hành thường thực hiện các thao tác như: cấp phát không gian bộ nhớ cho tiến trình mới, điều chỉnh lại danh sách các tiến trình, nạp mã lệnh của chương trình B trên đĩa cứng và không gian nhớ vừa cấp phát cho tiến trình. Đưa tiến trình mới vào danh sách cần điều phối của Hệ Điều Hành. Những công việc này thường mất thời gian đáng kể và chiếm giữ thêm tài nguyên của hệ thống.

Nếu tiến trình A đang chạy và nếu chúng ta muốn tiến trình B khởi động chạy trong không gian bộ nhớ đã có sẵn của tiến trình A thì có thể sử dụng các hàm exec được cung cấp bởi Linux. Các hàm exec sẽ thay thế toàn bộ ảnh của tiến trình A (bao gồm mã lệnh, dữ liệu, bảng mô tả file) thành ảnh của một tiến trình B hoàn toàn khác. Chỉ có số định danh PID của tiến trình A là còn giữ lại. Tập hàm exec bao gồm các hàm sau:

#include <unistd.h>

*extern char **environ;*

*int execl(const char *path, const char *arg, ...);*

*int execlp(const char *file, const char *arg, ...);*

*int execle(const char *path, const char *arg, ..., char *const envp[]);*

*int exect(const char *path, char *const argv[]);*

*int execv(const char *path, char *const argv[]);*

*int execvp(const char *file, char *const argv[]);*

Đa số các hàm này đều yêu cầu chúng ta chỉ đối số path hoặc file là đường dẫn đến tên chương trình cần thực thi trên đĩa. arg là các đối số cần truyền cho chương trình thực thi, những đối số này tương tự cách chúng ta gọi chương trình từ dựng lệnh.

Ví dụ sử dụng hàm exec, pexec.c

```
#include <unistd.h>
#include <stdio.h>
int main()
{
printf( "Thuc thi lenh ps voi execlp\n" );
```

```
execlp( "ps", "ps", "–ax", 0 );
printf( "Thuc hien xong. Nhung chung ta se khong thay duoc dong nay.\n" );
exit( 0 );
}
```

Nhân bản tiến trình với hàm fork()

Thay thế tiến trình đôi khi bất lợi với chúng ta. Đó là tiến trình mới chiếm giữ toàn bộ không gian của tiến trình cũ và chúng ta sẽ không có khả năng kiểm soát cũng như điều khiển tiếp tiến trình hiện hành của mình sau khi gọi hàm exec nữa. Cách đơn giản mà các chương trình Linux thường dùng đó là sử dụng hàm fork() để nhân bản hay tạo bản sao mới của tiến trình. fork() là một hàm khá đặc biệt, khi thực thi, nó sẽ trả về 2 giá trị khác nhau trong lần thực thi, so với hàm bình thường chỉ trả về 1 giá trị trong lần thực thi. Khai báo của hàm fork() như sau:

```
#include <sys/types.h>
#include <unistd.h>
pid_t fork()
```

Nếu thành công, fork() sẽ tách tiến trình hiện hành 2 tiến trình (dĩ nhiên Hệ Điều Hành phải cấp phát thêm không gian bộ nhớ để tiến trình mới hoạt động). Tiến trình ban đầu gọi là tiến trình cha (parent process) trong khi tiến trình mới gọi là tiến trình con (child process). Tiến trình con sẽ có một số định danh PID riêng biệt. ngoài ra, tiến trình con còn mang thêm một định danh PPID là số định danh PID của tiến trình cha.

Sau khi tách tiến trình, mã lệnh thực thi ở cả hai tiến trình được sao chép là hoàn toàn giống nhau. Chỉ có một dấu hiệu để chúng ta có thể nhận dạng tiến trình cha và tiến trình con, đó là trị trả về của hàm fork(). Bên trong tiến trình con, hàm fork() sẽ trả về trị 0. Trong khi bên trong tiến trình cha, hàm fork() sẽ trả về trị số nguyên chỉ là PID của tiến trình con vừa tạo. Trường hợp không tách được tiến trình, fork() sẽ trả về trị -1. Kiểu pid_t được khai báo và định nghĩa trong uinstd.h là kiểu số nguyên (int).

Đoạn mã điều khiển và sử dụng hàm fork() thường có dạng chuẩn sau:

```
pid_t new_pid;
new_pid = fork();     // tách tiến trình
switch (new_pid)
{
case -1: printf( "Khong the tao tien trinh moi" ); break;
case 0: printf( "Day la tien trinh con" );
// mã lệnh dành cho tiến trình con đặt ở đây
break;
```

```
default: printf( "Day la tien trinh cha" );
// mã lệnh dành cho tiến trình cha đặt ở đây
break;
}
```

- Ví dụ sử dụng hàm fork(), fork_demo.c

```
#include <sys/types.h>
#include <unistd.h>
#include <stdio.h>
int main()
{
pid_t pid;
char * message;
int n;
printf( "Bat dau.\n" );
pid = fork();
switch ( pid ) {
case -1: printf( "Khong the tao tien trinh moi" ); exit(1);
case 0: message = "Day la tien trinh con";
n = 0;
for ( ; n < 5; n++ ) {
printf( "%s", message );
sleep( 1 );
}
break;
default: message = "Day la tien trinh cha";
n = 0;
for ( ; n < 3; n++ ) {
printf( "%s", message );
sleep( 1 );
}
break;
}
```

exit(0);

}

Biên dịch và thực thi chương trình này, chúng ta sẽ thấy rằng cả 2 tiến trình hoạt động đồng thời và in ra kết quả đan xen nhau. Nếu muốn xem sự liên quan về PID và PPID của cả 2 tiến trình cha và con khi lệnh fork() phát sinh, chúng ta có thể thực hiện chương trình như sau:

$fork_demo & ps – af

Kiểm soát và đợi tiến trình con

Khi fork() tách tiến trình chính thành hai tiến trình cha và con, trên thực tế cả hai tiến trình cha lẫn tiến trình con đều hoạt động độc lập. Đôi lúc tiến trình cha cần phải đợi tiến trình con thực hiện xong tác vụ thì mới tiếp tục thực thi. Ở ví dụ trên, khi thực thi, chúng ta sẽ thấy rằng tiến trình cha đã kết thúc mà tiến trình con vẫn in thông báo và cả tiến trình cha và tiến trình con đều tranh nhau gởi kết quả ra màn hình. Chúng ta không muốn điều này, chúng ta muốn rằng khi tiến trình cha kết thúc thì tiến trình con cũng hoàn tất thao tác của nó. Hơn nữa, chương trình con cần thực hiện xong tác vụ của nó thì mới đến chương trình cha. Để làm được việc này, chúng ta hãy sử dụng hàm wait()

#include <sys/types.h>

#include <sys/wait.h>

pid_t wait(int &stat_loc);

Hàm wait khi được gọi sẽ yêu cầu tiến trình cha dừng lại chờ tiến trình con kết thúc trước khi thực hiện tiếp các lệnh điều khiển trong tiến trình cha. wait() làm cho sự liên hệ giữa tiến trình cha và tiến trình con trở nên tuần tự. Khi tiến trình con kết thúc, hàm sẽ trả về số PID tương ứng của tiến trình con. Nếu chúng ta truyền thêm đối số stat_loc khác NULL cho hàm thì wait() cũng sẽ trả về trạng thái mà tiến trình con kết thúc trong biến stat_loc. Chúng ta có thể sử dụng các macro khai báo sẵn trong sys/wait.h như sau:

WIFEXITED (stat_loc) Trả về trị khác 0 nếu tiến trình con kết thúc bình thường.

WEXITSTATUS (stat_loc) *Nếu WIFEXITED* trả về trị khác 0, macro này sẽ trả về mã lỗi của tiến trình con.

WIFSIGNALED (stat_loc) Trả về trị khác 0 nếu tiến trình con kết thúc bởi một tín hiệu gửi đến.

WTERMSIG(stat_loc) Nếu WIFSIGNALED khác 0, macro này sẽ cho biết số tín hiệu đã hủy tiến trình con.

WIFSTOPPED(stat_loc) Trả về trị khác 0 nếu tiến trình con đã dừng.

WSTOPSIG(stat_loc) Nếu WIFSTOPPED trả về trị khác 0, macro này trả về số hiệu của signal.

Ví dụ cách sử dụng hàm wait() để chờ tiến trình con kết thúc sau khi gọi fork(), wait_child.c

```
#include <sys/tyes.h>
#include <unistd.h>
#include <sys/wait.h>
int main()
{
pid_t pid;
int child_status;
int n;
// nhân bản tiến trình, tạo bản sao mới
pid = fork();
switch ( pid ) {
case -1: // fork không tạo được tiến trình mới
printf( "Khong the tao tien trinh moi" );
exit( 1 );
case 0: // fork thành công, chúng ta đang ở trong tiến trình con
printf( "Hello world from child\n" );
n = 0;
for ( ; n < 3; n++ ) {
printf( "Tien trinh con" );
sleep( 1 );
}
exit( 0 );          // Mã lỗi trả về của tiến trình con
default: // fork thành công, chúng ta đang ở trong tiến trình cha
printf( "Tien trinh cha, cho tien trinh con hoan thanh.\n" );
// Chờ tiến trình con kết thúc
wait( &child_status );
printf( "Tien trinh cha – tien trinh con hoan thanh.\n" );
}
return ( 0 );
}
```

Trong ví dụ trên, chúng ta tuy rằng đã có sử dụng biến child_status, nhưng chưa dùng nó để xem xét mã lỗi trả về của tiến trình con. Ví dụ dưới đây sẽ thực hiện việc này, wait_child2.c

```
#include <sys/tyes.h>
#include <unistd.h>
#include <sys/wait.h>
#include <studio.h>
int main()
{
char *message;
int n;
pid_t pid;
int child_status;
// nhân bản tiến trình, tạo bản sao mới
pid = fork();
switch ( pid ) {
case -1: // fork không tạo được tiến trình mới
printf( "Khong the tao tien trinh moi" );
exit( 1 );
case 0: // fork thành công, chúng ta đang ở trong tiến trình con
printf( "Hello world from child\n" );
n = 5;
for ( ; n > 0; n-- ) {
printf( "Tien trinh con" );
sleep( 1 );
}
exit( 37 );       // Mã lỗi trả về của tiến trình con
default: // fork thành công, chúng ta đang ở trong tiến tình cha
n = 3;
for ( ; n > 0; n-- ) {
printf( "Tien trinh cha" );
sleep( 1 );
```

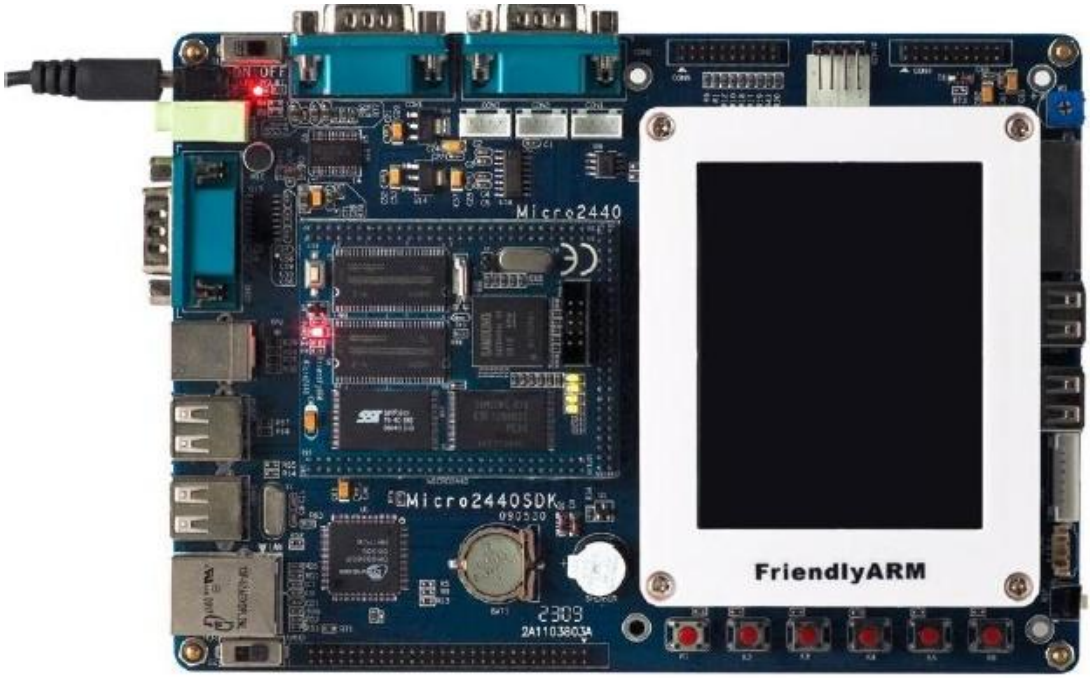
```
}
// Chờ tiến tình con kết thúc
wait( &child_status );
// Kiểm tra và in mã lỗi trả về của tiến trình con
printf( "Tien trinh con hoan thanh: PID = %d\n", pid );
if ( WIFEXITED( child_status ) )
printf( "Tien trinh con thoat ra voi ma %d\n", WEXITSTATUS( child_status ) );
else
printf( "Tien trinh con ket thuc binh thuong\n" );
break;
}
exit( 0 );
}
```

CHƯƠNG 4:

LẬP TRÌNH NHÚNG ARM TRÊN LINUX

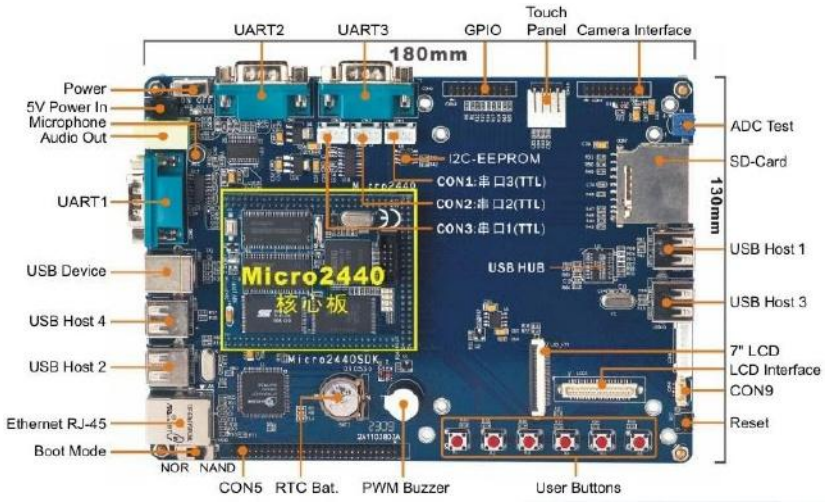
4.1.Giới thiệu KIT nhúng FriendlyArm Micro2440

KIT nhúng FriendlyArm Micro2440 do hãng Samsung, Hàn Quốc sản xuất. Sau đây là hình ảnh tổng thể của KIT:



Hình 4.1. KIT nhúng FriendlyArm Micro2440

Bố trí các cổng vào-ra, các khối linh kiện chức năng trên KIT:



Hình 4.2

Thông số kỹ thuật:

Khối chức năng	Thông số kỹ thuật
CPU	SAMSUNG S3C2440A, 400Mhz, max 533Mhz
SDRAM	- 64 MSDRAM - 32 bit dataBus - SDRAM Clock 100Mhz
Flash	- 64M or 128 M nand Flash - 2 M Nor Flash (đã được cài đặt sẵn BIOS)
Màn hình LCD	- Màn hình cảm ứng - Tối đa 4096 màu STN, kích thước 3,5 inches
Các thiết bị ngoại vi	-1 khối 10/100 M Ethernet RJ-45 (DM9000) - 3 Serial Port - 1 USB host - 1 USB Slave Type B - 1 giao tiếp SD Card - 1 stereo audio out, 1 micro in - 20 pin JTAG (kết nối với mạch nạp, debug) - 4 led đơn - 6 nút bấm - 1 còi điều khiển sử dụng PWM - 1 biến trở sử dụng kiểm tra bộ chuyển đổi số /tương tự (ADC) - 1 EEPROM giao tiếp theo chuẩn I2C - 1 giao tiếp với cảm biến ảnh (20- chân) - 1 pin cho đồng hồ thời gian thực - Nhuồn 5 V
Các hệ điều hành được hỗ trợ	- Linux 2.6 - Android - Win CE 5 và 6

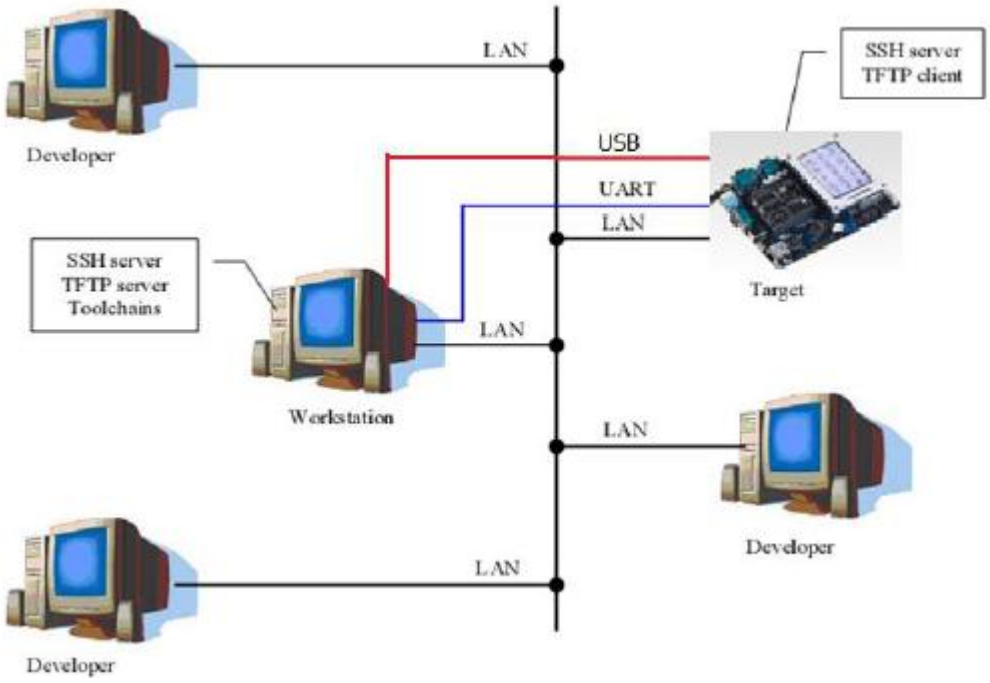
4.2. Môi trường phát triển ứng dụng

Phần mềm:

- Máy tính Linux (Ubuntu 9.04 hoặc mới hơn)
- Trình biên dịch chéo (C/C++ cross compiler): Cross toolchains (arm linux gcc 4.4.3)
- gFTP (Công cụ truyền nhận file theo giao thức FTP)
- minicom (phần mềm giao tiếp cổng Com trên Linux)
- USB push (Công cụ truyền file qua USB trên Linux)
- QT SDK, QT Embedded (Môi trường IDE để ph|t triển ứng dụng giao diện đồ họa trên nền tảng Qt Framework, dựa trên C/C++)

Phần cứng:

Hình 4.3 là môi trường phát triển ứng dụng theo nhóm



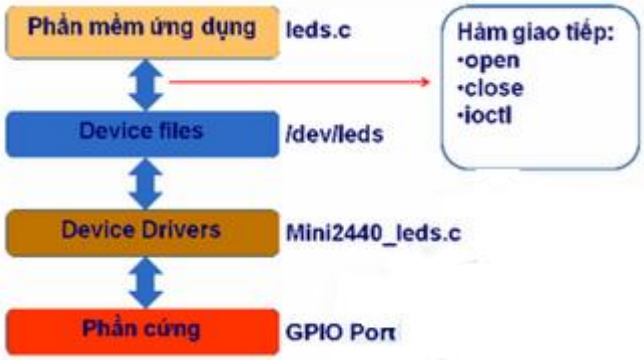
Hình 4.3

4.3. Lập trình điều khiển LED

Yêu cầu:

- Dãy 4 led đơn trên KIT ghép nối qua cổng GPIO đã có sẵn driver trên Embedded Linux.

- Mô hình giao tiếp:



Hình 4.4. Mô hình giao tiếp

Chương trình:

```
#include <stdio.h>

#include <stdlib.h>

#include <unistd.h>

#include <sys/ioctl.h>

 int main(int argc, char **argv)

{

 int on;

        int led_no;

        int fd;

        //Kiểm tra các tham số truyền vào đã đúng quy định chưa

         if (argc != 3 || sscanf(argv[1], "%d", &led_no) != 1 || sscanf(argv[2],"%d",
&on) != 1 ||

                on < 0 || on > 1 || led_no <  0 || led_no > 3)

{

                fprintf(stderr, "Usage: leds led_no 0|1\n");

                exit(1);

        }

        //Mở file
```

```
        fd = open("/dev/leds", 0);
        //Kiểm tra xem quá trình mở file có thành công không
        if (fd < 0) {
                perror("open device leds");
                exit(1);
        }
       //Điều khiển led
        ioctl(fd, on, led_no);
        close(fd);
        return 0;
}
```

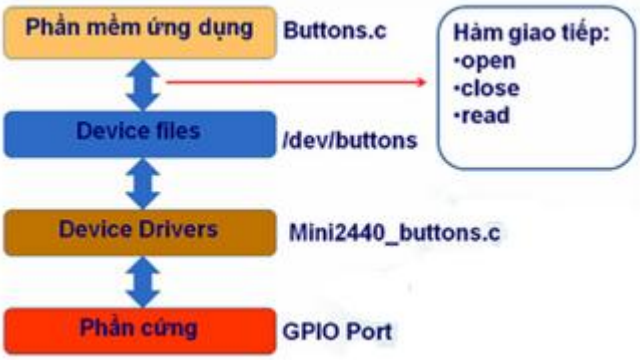
4.4. Lập trình đọc trạng thái nút bấm

Yêu cầu:

- Dãy nút bấm K1, K2, K3, K4, K5, K6 trên KIT được ghép nối qua GPIO, đã có sẵn driver trên hệ điều hành Linux nhúng

Có thể đọc trạng thái các nút bấm này (pressed/release or not ?) và có xử lý thích hợp.

- Mô hình lập trình với nút bấm:



Hình 4.5. Mô hình giao tiếp nút bấm

Chương trình:

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/ioctl.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <sys/select.h>
#include <sys/time.h>
#include <errno.h>
int main(int argc, char** argv)
{
        int buttons_fd;
        //Mảng lưu trạng thái của 6 nút bấm
        char buttons[6] = {'0', '0', '0', '0', '0', '0'};
        //Mở file
        buttons_fd = open("/dev/buttons", 0);
        //Kiểm tra quá trình mở file
        if (buttons_fd < 0) {
            perror("open device buttons");
            exit(1);
        }
        //Hỏi vòng kiểm tra trạng thái các nút bấm
        for (;;) {
        char current_buttons[6];
```

```
int count_of_changed_key;
int i;
//Đọc trạng thái các nút bấm
if (read(buttons_fd, current_buttons, sizeof current_buttons) != sizeof
current_buttons)
{
        perror("read buttons:");
        exit(1);
}
//Kiểm tra trạng thái các nút bấm và in ra trạng thái phù hợp (Key up
hay Key down)
for (i = 0, count_of_changed_key = 0; i < sizeof buttons / sizeof
buttons[0]; i++)
{
        if (buttons[i] != current_buttons[i])
{
            buttons[i] = current_buttons[i];
            printf("%skey %d is
            %s",count_of_changed_key ? ", ": "", i+1,
buttons[i] == '0' ?
            "up" : "down");
            count_of_changed_key++;
        }
 }
 if (count_of_changed_key) {
        printf("\n");
```

```
                }
        }
        //Đóng file thiết bị
        close(buttons_fd);
        return 0;
}
```

KẾT LUẬN

Qua việc thực hiện đề tài đã cho thấy kết quả khả quan, tạo tiền đề cho phát triển các ứng dụng với họ vi điều khiển ARM. Và qua nghiên cứu này em đã biết được tầm quan trọng và việc ứng dụng rộng rãi của hệ thống nhúng trong nghiên cứu cũng như ứng dụng vào đời sống. Giúp em có thêm được nhiều kiến thức về thực tế và bổ sung được thêm kiến thức đã học ở nhà trường. Với đề tài này em đã cơ bản nắm được lập trình nhúng ARM trên nền tảng hệ điều hành Linux. Nhưng do thị trường ARM ở Việt Nam chưa rộng, gây khó trong việc tìm kiếm tài liệu cũng như việc mua kit thực hành, do vậy việc nghiên cứu vẫn còn gặp nhiều khó khăn.

Hướng nghiên cứu và phát triển của đề tài được sự giúp đỡ tận tình và chu đáo của Thầy hướng dẫn Nguyễn Huy Dũng đề tài đã hoàn thành tốt việc nghiên cứu và lập trình ứng dụng ARM trên Linux. Một lần nữa em xin chân thành cảm ơn các thầy giáo, cô giáo đã truyền đạt kiến thức để em có thể hoàn thành đồ án có kết quả như mong đợi.

Em xin chân thành cảm ơn!

TÀI LIỆU THAM KHẢO

[1] . Philips (2005), The insider's guide to the Philips ARM7 based microcontroller.

[2] . *Jean J. Labrosse (2000),* Embbeded System Building Block Second Edition

[3] . Michael Barr, Anthony Massa (2006), *Programming Embedded Systems*, 0'Reilly

[4] Bởi Peter Marwedel Embedded System Design: Embedded Systems Foundations of Cyber-Physical Systems

[5] Williams, John A. Embedded Linux as a Platform for Dynamically Self-Reconfiguring Systems-On-Chip