Share to be shared

Căn bản và rất căn bản về hệ điều hành NachOS ☺

Tổng quan – System Call – Process (Thread) – Đa chương

.:Gió Reo:.

Mục lục :

Lời tỉ tê :

Sau khi tìm tài liệu về NachOS trên mạng thì rất ít tài liệu tiếng Việt nói về hệ điều hành này , chỉ có bài của mấy anh chị năm ngoái tại đây , tuy nhiên , bài của mấy anh chị chỉ đơn thuần là code và code , muốn hiểu được thì hơi khó , nhiều lúc , mình lại dễ bị ảnh hưởng bởi lối viết code của người khác nữa ^_^ ! Chính vì lý do trên , và sau một học kỳ cực nhọc với NachOS , mình rút ra được một số vấn đề chính cần phải thấu đáo khi các bạn nghiên cứu NachOS.

Nghiên cứu NachOS , ta sẽ được cái lợi gì ? Khi nghiên cứu NachOS , ta sẽ có cơ hội mô phỏng lại các hành động của một hệ điều hành thật sự như là cấp phát bộ nhớ , tạo tiến trình , quản lý các tiến trình , đồng bộ tiến trình bằng cách dùng semaphore , quản lý xuất nhập I/O …

Trong tài liệu này ,mình chủ yếu lấy các tài liệu về NachOS của trường đại học Khoa Học Tự Nhiên TPHCM , nơi mình đang theo học . Có vẻ sẽ hơi khó khăn với các bạn học khác trường , khác khóa với mình vì mỗi nơi , mỗi khóa sẽ có tài liệu khác nhau . Nhưng trên căn bản, NachOS vẫn là thế , ở đâu cũng vậy , cho nên các bạn đừng lo lắng.

Trong tài liệu này, mình chỉ gợi ý các bạn các hướng để nghiên cứu và tìm hiểu NachOS và đi sâu vào một số vấn đề cần thiết chứ không show hết toàn bộ tất tần tật ra . Nếu show hết ra thì các bạn có thể sẽ bị ảnh hưởng code của mình , như thế không tốt , phải tốn xíu thời gian cho NachOS thì kết quả đạt được mới có giá trị , sung sướng zà zui zẻ hơn đúng ko nào :D . À , còn nữa chứ , **HÃY TẬP ĐỌC TÀI LIỆU TIẾNG ANH** nha bạn ^__^ !! Nếu bạn có khả năng đọc được tiếng Anh thì hãy cố gắng đọc , dẫu hơi mất thời gian . Mình tạm gọi là hiểu được NachOS cũng nhờ ngồi giựt tóc mà đọc tài liệu tiếng Anh không đấy !!

Trước khi tiếp tục , các bạn nên coi qua tài liệu này , nó sẽ giúp bạn sử dụng thành thạo với Linux trước khi tiếp tục công việc ^__^ CLICK HERE TO

DOWNLOAD ==> [dành cho những ai chưa một lần chiêm ngưỡng vẻ đẹp đầy bí ẩn của cô nàng Linux] . Đặc biệt chú ý cách sử dụng đường dẫn tuyệt đối và đường dẫn tương đối trong linux nha !

Các tài liệu tham khảo khác được sử dụng trong tài liệu này (có đính kèm) :

+ *Bien_dich_va_Cai_dat_Nachos.pdf*

+ *Giao_tiep_giua_HDH_Nachos_va_chuong_trinh_nguoi_dung.pdf*

+ *narten-roadmap.pdf*

+ *nachos_study_book.pdf*

Các bạn chú ý là vì tài liệu mình viết dựa chủ yếu vào các tài liệu trên , nhất là 2 tài liệu tiếng Việt trên cùng của trường mình , nên lời khuyên là bạn hãy đọc sơ lược qua 2 tài liệu trên cùng (dẫu ko hiểu ráo trọi gì :D)

Nói chung , nếu bạn cố gắng đọc được hết 2 tài liệu tiếng Anh ở dưới thì coi như là bạn khỏi đọc tài liệu của mình nữa , Master rồi á .. hi hi hi !!

Mà dù gì đi nữa , bạn cũng phải lao đầu vào đọc 2 tài liệu tiếng Anh đó !! `(-.-`) ..

Và cuối cùng , hãy chắc rằng là bạn đã **nắm vững lý thuyết trên lớp** : **quản lý tiến trình** , **bộ nhớ** , **đồng bộ hóa** … trước khi tiếp tục nghiên cứu nhé !!! Và **đọc tài liệu phải đi đôi với thực hành** , không thể đọc suông được !! Bạn muốn hiểu và thực hành được NachOS không phải chuyện một hai ngày mà ít nhất rai lai từ nửa đến nguyên tháng . Nó được kết cấu từ rất nhiều class , nhiều file link qua link lại , link tới link lui chết luôn … Hãy đảm bảo là bạn có khả năng **bình tĩnh và kiên trì** vì ngoài NachOS còn có các vấn đề khác như lỗi cài đặt , lỗi Linux , lỗi WinSCP , lỗi putty , một đống lỗi tè le ko có lý do cũng ko hiểu vì sao khiến ta điên lên muốn đập nát màn hình … Goodluck !

Lời cảm ơn

Gửi lời cảm ơn chân thành của em đến các thầy cô bộ môn Mạng Máy Tính khoa Công Nghệ Thông Tin trường đại học Khoa Học Tự Nhiên TPHCM là thầy Phạm Tuấn Sơn , thầy Vũ Minh Trí , cô Trần Hồng Ngọc và các bạn cùng khóa học bộ môn Hệ Điều Hành ☺

^(*,..,*)^ hù hù

Phương pháp học đề nghị :

Đọc kỹ tài liệu trước

Lấy giấy bút ghi lại các khai báo mỗi class cần thiết và chú thích mọi thứ

Đọc hiểu đến đâu, thực hành tới đó , lâu lâu tò mò cái mới

Đọc hoài không hiểu thì mở máy ra mà thực hành

Thực hành làm hoài không được thì mở tài liệu ra đọc lại

Đọc và thực hành hoài không được thì … đi hỏi người khác

Chưa nhận được trả lời thì đi chơi , đú đởn hoa lá cành

Đú đởn xong quay về làm lại bước đầu tiên

…

Hãy tập thiền tịnh vì môn này nhiều lúc muốn tạt nước máy tính lắm

…

Bí hết cỡ , bí mọi đường mới coi code làm sẵn …

Copy code thì … ☹ … Oh, U’re loser …

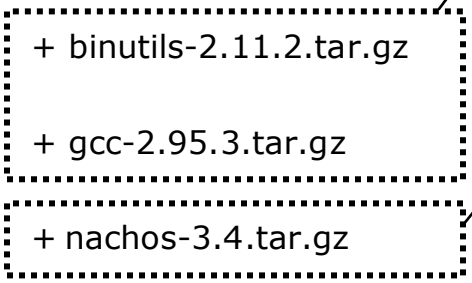
I - Giới thiệu NachOS

NachOS , mấy pé Tự Nhiên gọi nó là naCHÓ , =)) , tên thật là **Not Another Completely Heuristic Operating System** - Một phần mềm giả lập hệ điều hành với kiến trúc **MIPS** [Million Instructions Per Second] . Trong tài liệu này , NachOS được chạy trên môi trường **Linux** với kiến trúc **x86**

II - Cài đặt NachOS

Khi cài đặt NachOS , bạn sẽ cần phải cài đặt **gói phần mềm NachOS** và **trình biên dịch GCC dành cho quá trình biên dịch chương trình người dùng trên hệ điều hành NachOS** *(Mình sẽ gọi tắt là **GCC 2.95.3** nha , còn nó là gì thì mình sẽ nói sau, **GCC 2.95.3** này ko chạy trên NachOS mà chạy trên Linux ,nó chỉ hỗ trợ biên dịch thoy :D)*

Bạn hãy xem trong file *Bien_dich_va_Cai_dat_Nachos.pdf* để biết những "nguyên liệu" cần thiết trước khi bắt đầu cài NachOS . Bao gồm :



Xét cho cùng , NachOS cũng chỉ là 1 chương trình phần mềm được cài đặt trên hệ điều hành Linux mà thôi . Vậy , để cài NachOS , tức cài một phần mềm lên máy sử dụng hệ điều hành Linux thì ta phải làm thế nào ? (*giờ thì các bạn đừng quan tâm NachOS và GCC cho NachOS là cái gì hết nha, cứ coi như đó là 1 phần mềm cần phải cài đặt*)

*+ **Các cách cài đặt 1 phần mềm trên Linux** : Trên Linux có 3 cách cài đặt chính*

- 1 là vào Add/Remove chọn mấy cái gói mình thích rùi cài nó vào từ đĩa hay từ mạng Internet . Ủa, chứ mấy cái gói này ở đâu ra có sẵn cho mình cài vậy ?? Mấy gói phần mềm này là các gói do các công ty hỗ trợ bản Linux mà bạn đang sử dụng cung cấp cho bạn , tự distro Linux nó kiếm và cập nhật hằng ngày trên server của cộng đồng Linux (hay công ty bảo trợ cho bản Linux bạn đang sử dụng)

- 2 là đi tìm các gói phần mềm cài đặt sẵn (giống như là mấy cái file cài đặt setup.exe hay install.exe trên windows vậy á) . Đối với dòng Linux Redhat thì là .rpm còn với dòng Debian là .deb , cứ nhấp chuột vào là nó tự cài

- 3 là biên dịch lại source code của phần mềm đó (mã nguồn mở mà , người ta gửi mình code , mình diên dịch lại để xài thoy , phải qua các bước compiler , link). Tức là mình có được mã nguồn trong tay , mình dùng 1 trình biên dịch , biên dịch , giải mã cái mã nguồn đó thành mã máy , rồi thành 1 cái file thực thi để chạy . Giống như trên windows , bạn dùng VC++ để biên dịch tập tin hello.cpp sau

```
#include <stdio.h>

void main()

{

      printf("Hello world !!!");

}
```

Thành 1 file hello.exe để thi hành .

Và quá trình cài đặt **GCC 2.95.3** và **NachOS** chính là cách thứ 3 đấy (hix) … Còn cái **Binutils** (**Bin**ary **Util**ity**s**) dạng dạng như 1 tập các công cụ lập trình như thư viện á ..v..v..v dùng để tạo ra mấy cái thư viện hợp ngữ hay gì gì đó (hỏi anh gu gồ đẹp trai đi , mình hem biết ^__^)

****** Các vấn đề về cách cài đặt phần mềm thứ 3 : Biên dịch code *******

Để biên dịch từ một mã nguồn (ở đây là mã nguồn C) thành 1 cái file nhấn vào là chạy thì trên Linux , ta có 1 bộ compiler (trình dịch) là GCC (***G****NU* ***C****ompiler* ***C****ollection*) [*Cái GCC này gần tương ứng với VC++ trên Windows , VC++ là trình biên dịch kết hợp với trình soạn thảo luôn , còn GCC thì ko có , GCC chỉ đảm nhiệm việc biên dịch thôi*]. Vậy , chú ý lại , hiện giờ ta có mã nguồn viết = C của NachOS , để tạo ra 1 chương trình NachOS chạy được trên Linux thì ta phải biên dịch mã nguồn C đó bằng GCC. **Tóm lại , NachOS là một phần mềm trên Linux , và ta dùng GCC của Linux để biên dịch , cài đặt nó [nói chung là vậy , tuy nhiên , khi đi sâu vào việc cài đặt thì có 1 số thay đổi nhỏ mà rất quan trọng]**

Để tiếp tục nói tiếp phần cài đặt NachOS chi tiết hơn , mình phải rẽ qua một vài khái niệm 1 chút để các bạn có thể hình dung được trong quá trình cài NachOS , mình gõ lệnh gì , ý nghĩa ra sao , nó làm cái gì ! Sau phần nói sơ về GCC , mình sẽ tiếp tục nói về việc cài đặt NachOS . Như mình nói ở trên , trong hệ điều hành Linux , thường có tích hợp trình biên dịch GCC (đóng vai trò ~ VC++ trên windows).Để kiểm tra xem máy mình có cài gcc chưa , vào Terminal gõ **gcc --version** , nếu hiện ra phiên bản gcc thì coi như là đã có trong máy (lưu ý , với mã nguồn của NachOS 3.4 thì chỉ biên dịch tốt , ko lỗi với GCC phiên bản từ 3.0 đến 4.0)

Nếu không có , bạn có thể cài từ đĩa CD cài đặt bằng cách vào Add/Remove Application rồi tick chọn bộ Development Tools (với RedHat 9). Sau đó , Linux nó sẽ tự cài cho bạn.

Rồi , OK, tóm lại , trên Linux , phải có một trình biên dịch là GCC để biên dịch mã nguồn thành chương trình thực thi . Mình sẽ gọi cái GCC chính trên Linux đó là **GCC 3x** nhé !

GCC 3x

Là trình dịch mã nguồn cho Linux , nó chạy trên hệ điều hành Linux/nền kiến trúc x86 , biên dịch mã nguồn cấp cao C thành mã máy cho **Linux**/nền kiến trúc **x86** hiểu .

GCC 2.95.3

Đấy chính là cái GCC mà mình phải cài khi cài đặt NachOS đấy , **GCC 2.95.3** được cài trong gói gcc-2.95.3.tar.gz

Là trình dịch mã nguồn cho Linux , nó chạy trên hệ điều hành Linux/nền kiến trúc x86 , biên dịch mã nguồn cấp cao C thành mã máy cho **Linux**/nền kiến trúc **MIPS** hiểu .

Sau khi biên dịch, từ <tênfile>**.c** một hồi qua nhiều giai đoạn (quá trình biên dịch sẽ nói sau) , kết quả cuối cùng nó sẽ tạo ra file <tênfile>.**coff** (đây là dạng file thực thi cho **Linux**/nền kiến trúc **MIPS** đó)

Sau đó , nó dùng một phần mềm là **coff2noff** để biến file có đuôi là .**coff** đó thành file có đuôi là .**noff** . File .**noff** này chính là file thực thi của hệ điều hành **NachOS**/kiến trúc **MIPS**

Nói chung, tóm lại , GCC 2.95.3 là 1 dạng cross-compiler . [*Cross-compiler là gì , nó là 1 compiler có khả năng tạo những đoạn mã thực thi được cho một platform (nền kiến trúc phần cứng hay là tập thư viện cho phần mềm …) khác với platform mà nó đang chạy trên đó*] . GCC 2.95.3 dùng Linux /kiến trúc x86 để biên dịch ngôn ngữ C thành mã máy cho Linux / MIPS .

Một số chú ý khi biên dịch và cài đặt NachOS

Chúng ta lướt sơ qua phần biên dịch & cài đặt NachOS :

Những phần cài đặt chi tiết đều có nói ở trong tài liệu :

Bien_dich_va_Cai_dat_Nachos.pdf

Các bạn chú ý **phiên bản của NachOS** nhé . Hãy tìm các phiên bản GCC phù hợp với phiên bản NachOS của bạn ! Cái này hỏi thầy cô hen ^__^ !!

Mình sẽ ko nói về các lỗi cài đặt chi tiết, các bạn phải tự cài đặt để tìm hiểu tại sao mình sai , nhớ chú ý khoảng trắng <space> giữa các thành phần trong câu lệnh, mình chỉ nói về ý nghĩa một số câu lệnh để hiểu rõ hơn về NachOS mà thôi

Câu lệnh biên dịch mã nguồn đối với Binutils & GCC , thường có câu lệnh sau

% ./configure (*Chú ý , phải có dấu ./ hay là đường dẫn tuyệt đối vì dấu ./ hay đường dẫn tuyệt đối này thể hiện đây là thư mục đang làm việc và configure ko phải là 1 lệnh*) là mở file tên **configure** kiểm tra xem đã đầy đủ điều kiện như thư viện , file , v..v… để cho quá trình cài đặt sau này chưa ? (thường thì bên trong 1 gói mã nguồn để cài đặt có file này) Trong tài liệu hướng dẫn , câu lệnh này có thêm các tham số là **--host=i686-pc-linux-gnu** tức là cái platform nền thực sự là Linux/x86 và **--target=decstation-ultrix** tức là cái platform nền chuyển qua là Linux/MIPS .. còn **-- prefix =...** là đường dẫn đến nơi cần chứa kết quả của quá trình biên dịch

Makefile là gì ? Thường khi bạn biên dịch 1 file **.c** trên linux như là **hello.c** , bạn vào Terminal , gõ %**gcc hello.c –o hello.out** thì nó sẽ biên dịch thành **hello.out** cho bạn . Đó là file thực thi đó . Tuy nhiên , đó là đối với các mã nguồn đơn giản . NachOS không đơn giản , bạn phải biên dịch phần này trước , phần kia trước , rồi link tè le xíu mại . Bạn không thể làm được điều đó . Makefile xuất hiện giúp bạn . Makefile là 1 file ghi lại đầy đủ trình tự biên dịch , ví như là biên dịch cái nào , link cái nào , cái nào trước , cái nào sau tè le thì trong Makefile có đủ hết . Việc ta sửa

lại **GCCDIR** = <đường dẫn của GCC 2.95.3>**/** trong file Makefile của thư mục code/**test có nghĩa là những gì (chương trình người dùng) trong thư mục code /test này sẽ được biên dịch bởi GCC 2.95.3 và coff2noff** (nhìn cuối Makefile đi , thấy coff2noff liền àh)

Vậy , khi biên dịch NachOS , theo trình tự mà trong file Makefile (Makefile trong thư mục **code** nha , ko phải Makefile trong thư mục **code/test** . Cái Makefile trong thư mục **code** là trình tự biên dịch cho toàn bộ phần mềm NachOS , còn Makefile trong **code/test** là trình tự biên dịch cho các file nằm trong code/test) đã định sẵn , Linux sẽ dùng **GCC 3x** biên dịch toàn bộ mã nguồn của NachOS , tạo ra

file thực thi **NachOS** trong **code/userprog**)để chạy . Tuy nhiên , vì ta sửa trong Makefile của thư mục **code/test** cái GCCDIR , nên khi Linux dùng GCC 3x biên dịch NachOS , nó sẽ chừa thư mục **code/test** ra và **Linux sẽ cho GCC 2.95.3 ra biên dịch thư mục code/test đó** !

➔ **Tóm lại , phần mềm NachOS (bao gồm tất cả các thư mục trong thư mục code – trừ thư mục code/test ra) được biên dịch bởi GCC 3x để có thể chạy trên Linux/x86 . Các file trong thư mục code/test sẽ được biên dịch bởi GCC 2.95.3 thành file.coff , rồi được phần mềm coff2noff chuyển từ đuôi .coff sang đuôi .noff**

Lệnh ./userprog/nachos –rs 1023 –x ./test/halt có ý nghĩa như vầy :

+./userprog/nachos là đường dẫn đến NachOS 

, có nghĩa là mở NachOS lên (giống như nhấp đôi vào NachOS để chạy á :D , nói thế thoy , chứ nhấp đôi vào nó ko chạy đâu , chỉ chạy trên màn hình Terminal thôi)

+ –rs 1023 –x , trong đó –rs và –x là các tham số , 1023 là một con số tùy chọn, chúng cách nhau bằng 1 khoảng trắng . Các bạn sẽ hiểu được chúng khi nghiên cứu kỹ ở phần sau ^__^ . Mật bí , nó nằm hết trong file **threads/main.cc**

+ ./test/halt là đường dẫn của file halt.noff đã được biên dịch xong xuôi

Nguyên câu lệnh này là , khởi động máy ảo MIPS và hệ điều hành NachOS lên để nạp và thực hiện chương trình halt (tắt máy)

NachOS chạy như là 1 tiến trình chạy trên LINUX !!

III - Tổng quan NachOS

(sau khi cài thành công nha ^__^)

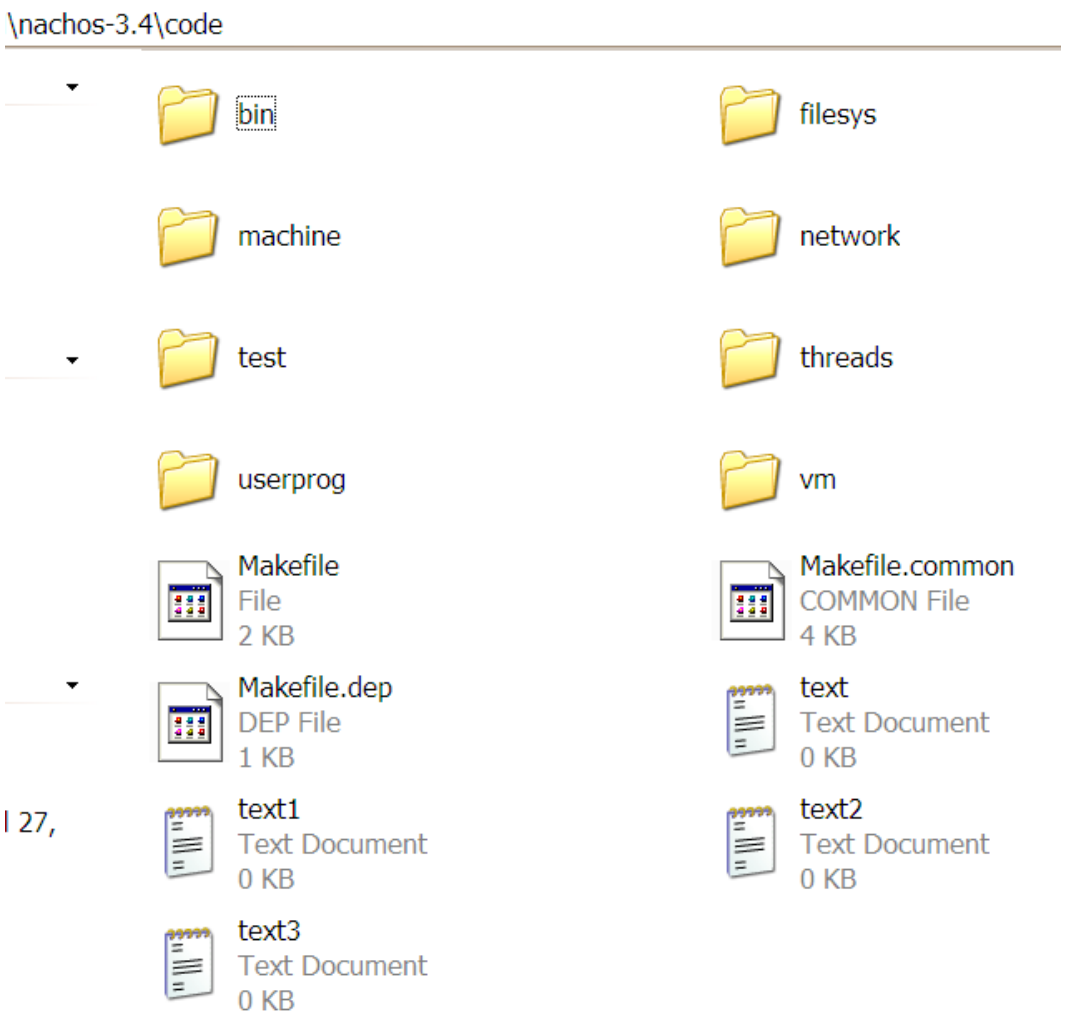
NachOS là một phần mềm giả lập một hệ điều hành chạy trên kiến trúc MIPS . Chỗ này rất khó nói … . Nó sẽ giả lập ra sao ? Giống như VMware àh .. uh , gần đúng rồi như mà nó ko có dữ dẫn như VMware , nó .. hiền hơn , hiền hơn nhiều luôn nhưng cũng đủ đè chết , đè bẹp dí sinh viên chúng ta ☺

Bây giờ , mình nói với bạn , bạn hãy tạo cho mình 1 cái hộp có đáy để chứa đĩa CD . Bạn có thể bỏ đĩa CD và hộp và lấy đĩa CD ra khỏi hộp . Bạn sẽ hình dung cái gì . Ồh , đó là STACK đúng ko nào . Vậy , để trừu tượng hóa cái STACK đó = ngôn ngữ C++ , bạn làm gì ? Có phải bạn sẽ viết 1 lớp tên là HopDiaCD đại loại như sau ko ?

```
Class HopDiaCD
{
Private :
     Int top;
     Int bottom;
     CD quality[100];
Public :
     int Push(CD x);
     int Pop (CD &x);
//…. Vân vân
};
```

Đó , ờ đây , NachOS cũng giả lập như vậy . Một thực thể vi xử lý , bộ nhớ … của máy MIPS sẽ được giả lập , được mô phỏng thông qua ngôn ngữ C++ .

Các bạn nhìn vào hình sau đây Đây là thư mục code của NachOS , chứa hầu hết các đoạn mã nguồn mô phỏng NachOS :



Các bạn chú ý từng thư mục có nhiệm vụ riêng của nó , tất cả liên kết với nhau tạo thành một thể thống nhất – **Hệ thống NachOS !**

IV - Thành phần của hệ thống NachOS và 2 chế độ User MODE , System MODE (Kernel MODE)

Theo mình nghiên cứu thì **Hệ thống NachOS** chia làm 3 phần chính như sau (Không tính phần mạng nhé, phần này mình ko nghiên cứu):

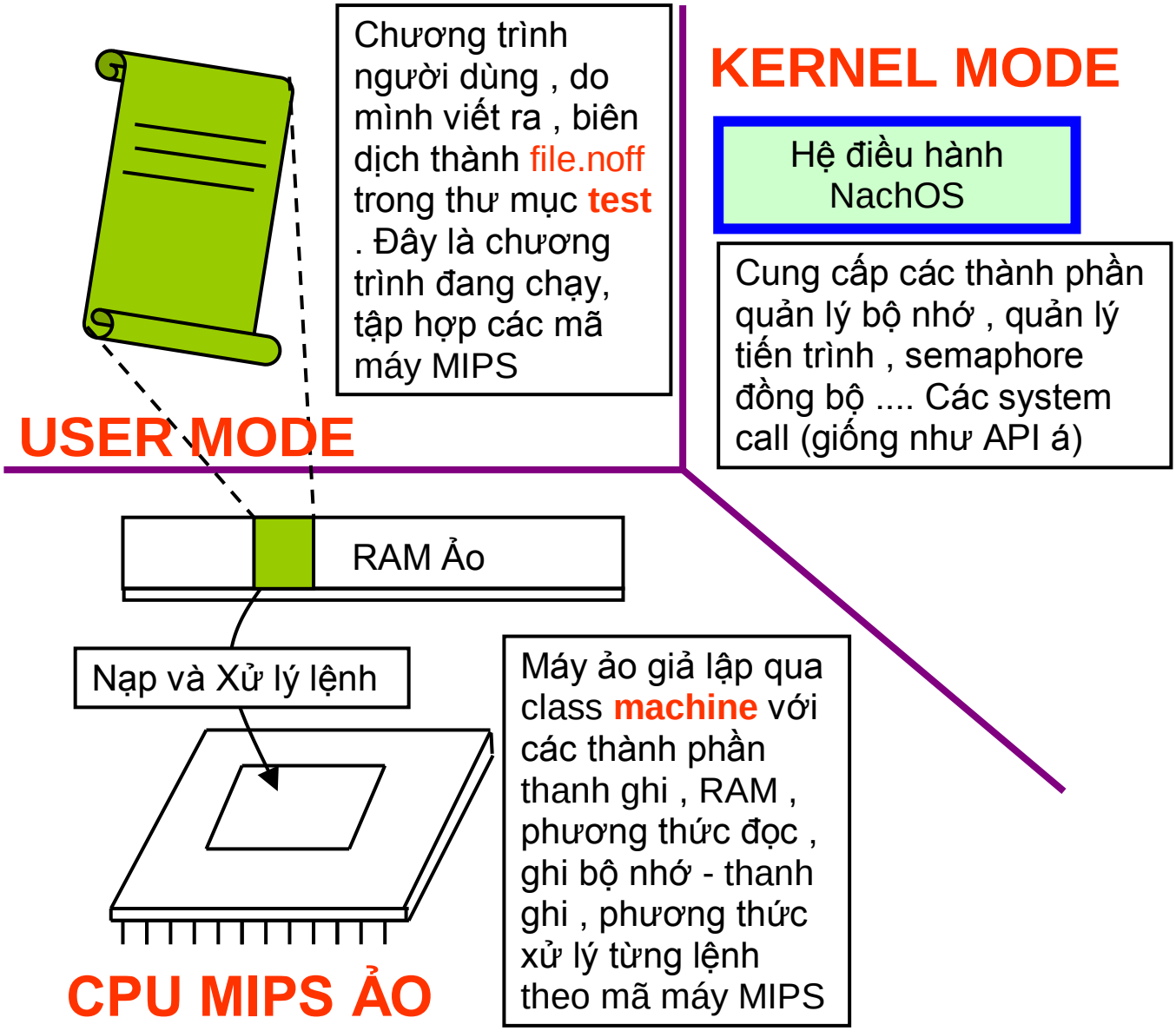
+ 1* ***Chương trình ứng dụng*** (…code/test) -- Nhà lập trình phần mềm ứng dụng sẽ thao tác trên đây

+ 2* ***Cỗ máy ảo*** *MIPS* (CPU , Registers , RAM , I/O , FileSys …v…v…)

+ 3* ***Hệ điều hành NachOS*** (là phần con của Hệ thống NachOS thôi nha) -- Nhà lập trình nhân hệ điều hành sẽ làm việc chỗ này

Xin mời bạn nhìn vào hình sau đây . Hình sau đây ko đúng lắm nhưng mình vẽ thế này để cho bạn dễ hình dung [**xin mời coi lại hình tổng quan NachOS ở tài liệu đính kèm , nhớ là vừa coi hình này vừa coi hình đính kèm nhá**]

Đây là HỆ THỐNG NACHOS



+Đầu tiên phải nói đến **cỗ máy ảo** này , tất cả đều nằm trong thư mục

code/machine  và chủ yếu ở file **machine.h**

//Lớp lệnh , miêu tả mã máy cho MIPS , coi sơ thoy , ko cần quan tâm

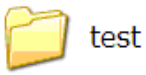
```
class Instruction {
  public:
    void Decode();       // decode the binary representation of the instruction

    unsigned int value; // binary representation of the instruction

    char opCode;      // Type of instruction.  This is NOT the same as the
                  // opcode field from the instruction: see defs in mips.h
    char rs, rt, rd; // Three registers from instruction.
    int extra;        // Immediate or target or shamt field or offset.
                      // Immediates are sign-extended.
};
```

//Lớp machine , giả lập 1 máy ảo , chỉ xem những hàm cần thiết nhá , tô xanh lá

```
class Machine {
  public:
    Machine(bool debug);
    ~Machine();
    void Run();                 // Run a user program
    int ReadRegister(int num);      // read the contents of a CPU register
    void WriteRegister(int num, int value);
                        // store a value into a CPU register
//Vòng lặp nạp và giải mã, thi hành lệnh
    void OneInstruction(Instruction *instr);
                        // Run one instruction of a user program.

    bool ReadMem(int addr, int size, int* value);
    bool WriteMem(int addr, int size, int value);
                        // Read or write 1, 2, or 4 bytes of virtual
                        // memory (at addr).  Return FALSE if a
                        // correct translation couldn't be found.
```

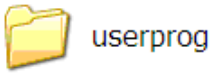
```
    ExceptionType Translate(int virtAddr, int* physAddr, int size,bool writing);
                        // Translate an address, and check for
                        // alignment.  Set the use and dirty bits in
                        // the translation entry appropriately,
                        // and return an exception code if the
                        // translation couldn't be completed.

    void RaiseException(ExceptionType which, int badVAddr);
                        //Lập 1 ngắt chuyển từ User MODE sang Kernel MODE

//Đây chính là cái RAM trong hình trên này
    char *mainMemory;         // physical memory to store user program,
                        // code and data, while executing
    int registers[NumTotalRegs]; // CPU registers, for executing user programs

//Còn đây là chia bộ nhớ theo Page này , sẽ dùng đến khi làm đa chương đó
    TranslationEntry *pageTable;
    unsigned int pageTableSize;

};
```

+ Cái hình tờ giấy màu xanh dương là **chương trình ứng dụng người dùng do mình viết và biên dịch** thành file **.noff** , lúc thành **noff** thì nó toàn là mã máy òi



, nó nằm trong thư mục test .

Khi gọi lên chạy , nó sẽ được nạp vào bộ nhớ RAM ảo của MIPS (*vẫn là RAM thật thôi , nhưng nó là một vùng nhớ do cái máy ảo MIPS quản lý nên gọi là RAM ảo á mà , chỗ* `char*mainMemory;` *là Ram của máy MIPS ảo đó*) , và cái CPU MIPS ảo sẽ lần lượt nạp các lệnh từ RAM ảo của nó vào mà xứ lý (yêu cầu coi hàm `OneInstruction` của lớp Machine) , cứ thế mà nạp , decode –giải mã , rồi thi hành thôi … mọi chuyện sẽ bình thường khi đến lúc gặp **system call** (nói sau)

+ **Hệ điều hành NachOS** cấu thành từ một vài file mã nguồn , dùng để xứ lý các system call , và được đặt trong thư mục **userprog , thread** ..v..vv… chủ yếu trong **userprog** 

+ Các bạn nhìn lên sơ đồ hệ thống NachOS , cách bạn có thấy dòng chữ USERMODE và KERNEL MODE không ? Xét trên vùng RAM thật của máy tính :

- **User MODE** : vùng nhớ của những chương trình ứng dụng chạy trên NachOS/MIPS
- **Kernel MODE** : vùng nhớ của hệ điều hành NachOS

Chú ý rằng , hệ điều hành NachOS này ko được máy MIPS xử lý lệnh , mà nó được máy thiệt x86 xử lý từng lệnh [tại sao ? Thứ 1 là vì GCC 3x trên linux biên dịch chú NachOS này , chú CPU MIPS ảo đâu có hiểu , thứ 2 là vùng nhớ này nằm riêng ra, ko nằm trong bộ nhớ RAM ảo , thì làm sao CPU MIPS ảo xử lý được] . Chỗ này mình vẫn đang thắc mắc . Trong máy thật , hệ điều hành và chương trình ứng dụng đều nạp lên chung 1 bộ nhớ , còn ở đây , hệ điều hành tách rời … tách rời luôn cả cái máy luôn =))

V - Quá trình biên dịch chương trình người dùng trên NachOS

Yêu cầu : mở file

Giao_tiep_giua_HDH_Nachos_va_chuong_trinh_nguoi_dung.pdf ra coi

Bạn phải nhớ 1 điều rằng , khi nghiên cứu NachOS , bạn phải đứng ở 2 vai trò khác nhau : Nhà lập trình phần mềm ứng dụng, Nhà lập trình nhân hệ điều hành

Bạn sẽ là nhà lập trình phần mềm ứng dụng , bạn sử dụng những hàm mà nhà lập trình nhân hệ điều hành cung cấp , giống như bạn xài mấy hàm API , chỉ hiểu tác dụng của nó , còn nội dung hàm API ra sao thì bạn ko biết

Bạn sẽ là nhà lập trình nhân hệ điều hành , bạn phải viết nội dung các hàm API , cung cấp cho nhà lập trình phần mềm ứng dụng sử dụng

Sau đây , mình sẽ nói đến quá trình biên dịch chương trình người dùng (nằm trong thư mục test)– những chương trình do các nhà lập trình phần mềm ứng dụng viết

Các chương trình người dùng trên NachOS nằm trong thư mục **code/test** sẽ được biên dịch như sau (trích dẫn kèm các câu lệnh trong Makefile):

- Khi biên dịch NachOS , **GCC 2.95.3** biên dịch các file mã nguồn **XXX.c** trong thư mục code/test

```
GCCDIR = ../../../gnu-decstation-ultrix/decstation-ultrix/2.95.3/
```

GCC 2.95.3 biên dịch cho các file mã nguồn này , GCC 2.95.3 ko có hỗ trợ biên dịch các thư viện như stdio.h , conio.h … . Vì thế các file XXX.c này được viết = C chuẩn và không thể sử dụng bất kỳ một hàm nào như printf , scanf … . Tức chương trình của ta viết chỉ +-*/ , while , if , khai báo int, long , float … mà thoy .Chúng ta muốn in ra màn hình hay nhập vào từ bàn phím phải thông qua System Call (nói sau)

- Trong code/test có sẵn 1 file hợp ngữ **start.s** (đây được coi là file thư viện) . Khi biên dịch , nó sẽ biên dịch file mã nguồn **XXX.c** thành file hợp ngữ **XXX.s**

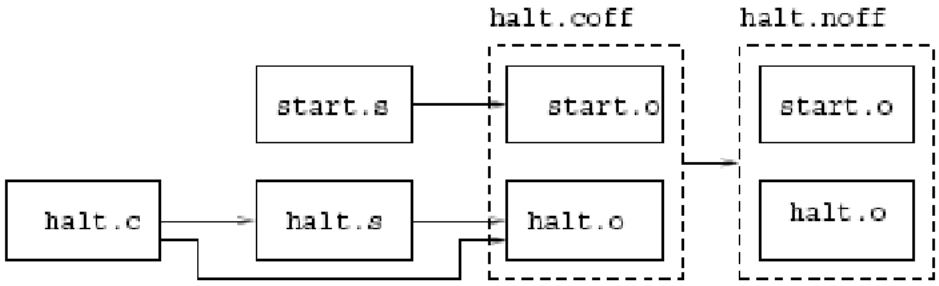
- Sau đó lấy file **XXX.s** này link với file **start.s** tạo thành 1 file tổng hợp **XXX.noff** (bao gồm XXX.o và start.o) . Đây là file thực thi cho Linux/MIPS

```
halt.o: halt.c
	$(CC) $(CFLAGS) -c halt.c
halt: halt.o start.o
	$(LD) $(LDFLAGS) start.o halt.o -o halt.coff
	../bin/coff2noff halt.coff halt
```

- Sau đó dùng phần mềm **coff2noff** (trong thư mục bin)chuyển qua file XXX.noff. Đây là file thực thi cho NachOS/MIPS

```
halt.o: halt.c
	$(CC) $(CFLAGS) -c halt.c
halt: halt.o start.o
	$(LD) $(LDFLAGS) start.o halt.o -o halt.coff
	../bin/coff2noff halt.coff halt
```

Bạn thấy đấy , nhìn hình sau , một ví dụ khi biên dịch file mã nguồn halt.c:



Khi viết 1 chương trình phần mềm ứng dụng mới , để biên dịch nó , bạn nhớ thêm vào Makefile

```
all: halt shell matmult sort createfile <tên file>
… … … …

<tên_file>.o: <tên_file>.c
	$(CC) $(CFLAGS) -c <tên_file>.c
<tên_file>: <tên_file>.o start.o
	$(LD) $(LDFLAGS) start.o <tên_file>.o -o <tên_file>.coff
	../bin/coff2noff <tên_file>.coff <tên_file>
```

VI - System call

Là gì ? Là lời nhờ vả của chương trình người dùng khi nó không làm được điều gì đó , cần đến sự giúp đỡ của hệ điều hành :D . Cứ cho nó là hàm API đi ^_^

Bạn nhớ chú ý , hiểu được bí kíp system call này là có thể ngộ được đạo NachOS , dĩ bất biến , ứng vạn biến , tung hoành ngang dọc , danh trấn giang hồ , cái gì sau này dẫu có phức tạp , thiên biến vạn hóa , xét cho cùng cũng chỉ là thứ bèo bọt …

… nổ xíu thế thoy 😐

Để hiểu được 1 system call , ta phải chạy thử 1 chương trình có gọi system call .

Yêu cầu : mở file

Giao_tiep_giua_HDH_Nachos_va_chuong_trinh_nguoi_dung.pdf ra coi

Lần theo dấu vết

Chúng ta sẽ coi cách mà chương trình halt.noff được thực thi như trong tài liệu vẽ

Halt.c

```
#include "syscall.h"
int main()
{
   Halt();
   /* not reached */
}
```

Nhận xét : *Trong chương trình **Halt** này , chẳng có +-*/, hay khai báo biến , while, if gì cả mà chỉ có đúng 1 hàm tên là Halt() . Hàm này được gọi là system call (lời gọi hệ thống) có nhiệm vụ là tắt máy ảo . Như đã định nghĩa trên nó là lời nhờ vả của chương trình người dùng khi nó không làm được điều gì đó , cần đến sự giúp dỡ của hệ điều hành . Chương trình **Halt** ko thể tự nó tắt máy được , mà nó phải cần đến hệ điều hành tắt máy cho nó*

Sau khi được nạp vào RAM ảo , CPU MIPS bắt đầu nạp các lệnh của halt.noff để thực thi từng lệnh một .

halt.noff = tập mã máy MIPS = start.o + hatl.o

Sau đây , mình sẽ dùng hợp ngữ trong start.s và halt.s để diễn tả phần tương ứng .Khi chạy halt.noff , nó sẽ chạy từ trên xuống dưới , nó sẽ chạy vào phần mã máy của start trước

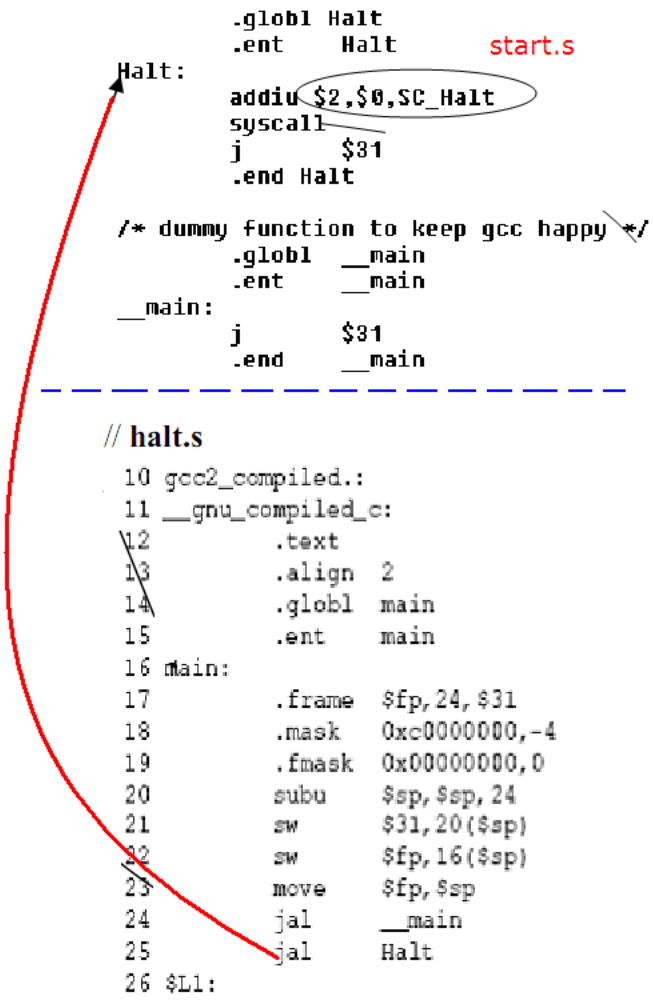
Đầu tiên nó nhảy vào hàm main () của halt.c bằng lệnh jal **main**

```
__start:                                    start.s
        jal     main
        move    $4,$0
        jal     Exit    /* if we return from main, exit(0) */
        .end __start

        .globl Halt
        .ent    Halt
Halt:
        addiu $2,$0,SC_Halt
        syscall
        j       $31
        .end Halt

/* dummy function to keep gcc happy */
        .globl  __main
        .ent    __main
__main:
        j       $31
        .end    __main
```

// **halt.s**

```
10 gcc2_compiled.:
11 __gnu_compiled_c:
12         .text
13         .align  2
14         .globl  main
15         .ent    main
16 main:
```

Sau đó , dựa vào tên của hàm system call là Halt nó sẽ nhảy đến **cái nhãn (label)** – **tên hàm chính là cái nhãn , đơn giản vậy thôi** - tương ứng với hàm đó .



Tại vị trí label này , ta thấy có dòng lệnh

addiu $2,$0,SC_Halt

Ý nghĩa : Cho mã của system call Halt là SC_Halt (là 1 số nguyên) vào thanh ghi thứ 2

syscall

Ý nghĩa : Lệnh này để gọi xử lý system call . Khi nạp từng lệnh , máy ảo giải mã lệnh này , nhận thấy opCode của lệnh là OP_SYSCALL nó sẽ gọi hàm Raise Exception . Trong hàm này , đơn giản là chỉ chuyển ngắt qua system mode , thực hiện xử lý system call đó , sau đó chuyển lại qua user mode để chạy tiếp chương

trình. Hiện tại , trong hàm Exceptionhandle chỉ có xử lý mỗi trường hợp cho system call Halt – tắt máy mà thôi , còn các system call khác thì chưa có

➔ Nhiệm vụ của một nhà phát triển nhân hệ điều hành là tạo ra nhiều các system call cung cấp cho nhà phát triển phần mềm có thể viết chương trình

➔ Từ trên , ta có thể nhận xét được rằng , file start.s tương tự như một thư viện lập trình được cung cấp cho nhà lập trình phần mềm ứng dụng sử dụng . Hễ nhà viết nhân hệ điều hành tạo ra các system call mới thì họ sẽ cung cấp cho nhà lập trình phần mềm các hàm system call này trong file start.s

Các bạn nhận thấy rằng , khi được cung cấp hàm Hatl() , nhà lập trình phần mềm chẳng cần quan tâm bên trong đó có gì , muốn tắt máy , chỉ cần gọi Halt() là ok!

Đối với nhà lập trình nhân hệ điều hành thì có phần phức tạp hơn , họ phải viết **nội dung mà hệ điều hành phải xử lý khi nhận được system call** Halt() này , rồi họ phải **thêm vào trong thư viện hàm start.s** để nhà lập trình phần mềm có thể sử dụng. Vai trò của file start.s sẽ được làm rõ hơn trong phần sau đây , cách tạo 1 system call mới !

Tạo 1 system call mới như thế nào ?

<Để tạo 1 system call mới , bạn phải nắm rõ đường đi từ lời gọi hàm cho đến lúc nó dược thực hiện trước như đã nói ở phần trên>

Để tạo 1 system call mới cho nhà lập trình phần mềm có thể sử dụng , bạn phải xem xét các file , và trải qua các bước sau (**Coi mấy cái mũi tên sau cùng**):

(mình sẽ dùng **ví dụ tạo 1 system call Create ,hàm này dùng tạo file**)

- Khai báo 1 system call mới trong /code/userprog/syscall.h

  Định nghĩa mã của system call

  ```
  #define SC_Create 4
  ```

  Khai báo tên hàm của system call đó

  ```
  int Create(char* name);
  ```

- Thêm các đoạn mã trong start.s và start.c , phần này để khi người dùng gọi hàm (thư viện hàm) . Nó sẽ gọi đến system call tương ứng như sau :

  ```
  .globl Create
  .ent   Create
  Create://Cái nhãn,dùng để nhảy đến trong hợp ngữ,tên nhãn=tên hàm
  addiu $2,$0,SC_Create //Cho mã SC_Create=4 vào thanh ghi 2
  syscall //Gọi hệ điều hành xử lý giùm system call Create này
  j      $31
  .end Create
  ```

- Hệ điều hành khi nhận được 1 system call , phải phân loại chúng thuộc về system call dạng gì , tắt máy , ghi file ,tạo file , in ra màn hình , nhập từ bàn phím hay sao .v…v…v . Sau đó , bạn bạn phải code ra cách xử lý chúng . Đến đây coi như bạn tạo xong 1 system call rồi đó .

```
void
ExceptionHandler(ExceptionType which) //Kiểu Exception
{
    int type = machine->ReadRegister(2); // Đọc thanh ghi thứ 2 để biết mã System call là gì

    switch (which){

    case NoException:
      return;
```

```
    case SyscallException: //Nếu Exception which là System Call thì …
      switch (type){ // type thuộc dạng System Call nào đây …

      case SC_Halt: // Nếu là system call Halt
      DEBUG(dbgAddr, "\n Shutdown, initiated by user program.");
      printf ("\n\n Shutdown, initiated by user program.");
      interrupt->Halt(); //Thực hiện Halt máy …
      break;

      case SC_Open:
            //Thực hiện xử lý mở file
      break;

      case SC_Close:
            //Thực hiện xử lý đóng file
      break;

      case SC_Read:
            //Thực hiện xử lý đọc file
      break;

      case SC_Write:
            //Thực hiện xử lý ghi file
      break;

      case SC_Create:
            //Thực hiện xử lý tạo file
      break;

  … vân vân vân …
     default:
            printf("\n SyscallException:  Unexpected system call %d", type);
            //break; Mặc định là Halt máy
            interrupt->Halt();
      }

      //printf("\n Incrementing PC.");
      DEBUG(dbgAddr, "\n Incrementing PC.");
      AdvanceProgramCounter();
      break;

      default:
      printf("\n Unexpected user mode exception (%d %d)", which, type);
      interrupt->Halt();
    }
}
```

VÍ DỤ : Trình tự + những quá trình cần thực hiện của hàm Create:

- Yêu cầu các bạn phải mở tài liệu tiếng Anh và đọc sơ qua về hệ thống file trong NachOS . Mình chỉ nói sơ qua đây thôi , NachOS có 2 hệ thống file , 1 là hệ thống file của nó và 2 là dùng hệ thống file của Linux (Unix) . Hiện tại , NachOS đang xài hệ thống file thứ 2 , nên việc

tạo file , thông qua lớp fileSystem , lớp fileSystem này lại dùng hàm của Linux để tạo file :D !!

- Đầu tiên chương trình người dùng muốn tạo 1 file tên char* X = "abc.txt". Chuỗi X này đang nằm đâu đó trong vùng nhớ của chương trình người dùng , là USER MODE đó. Chương trình người dùng sẽ dùng đến system call Create(X); để nhờ vả hệ điều hành NachOS làm giùm
- ***Bây giờ thì ta xem xét chi tiết việc xử lý hàm tạo 1 file (Create) thì thế nào ?***
- Chương trình đi ngủ , hệ điều hành NachOS vươn vai thức dậy , đọc thanh ghi R2 , và nó biết nhiệm vụ là tạo ra 1 file . Nhưng hỡi ôi , làm sao nó biết tạo file tên gì bây giờ , nó chỉ hiểu các biến trong phạm vi vùng nhớ của nó , tức KERNEL MODE thôi , chứ còn trong USER MODE thì nó biết chỗ nào đâu là lấy … hix hix … Thế là, mình , một nhà lập trình nhân hệ điều hành sáng giá , dựa trên tính chất sau đây !!

  Tham số thứ nhất của hàm nằm trong thanh ghi R4

  Tham số thứ hai của hàm nằm trong thanh ghi R5

  Tham số thứ ba của hàm nằm trong thanh ghi R6 …

  Đó , dựa vào đó , ta chỉ cần cho NachOS **đọc thanh ghi R4** là có thể biết được **vị trí bắt đầu** của cái chuỗi đó ở đâu trong vùng nhớ của USER MODE
- Sau đó , qua hàm **User2System** , hệ điều hành NachOS copy cái chuỗi X là "abc.txt" thuộc USER MODE đó qua chuỗi Y của vùng nhớ KERNEL MODE (chuỗi Y giống hệt như chuỗi X luôn). NachOS sẽ tạo 1 file mới dựa trên chuỗi Y này
- Xử lý èo èo xong , hệ điều hành NachOS đi ngủ , chương trình người dùng chạy tiếp ! Tiếp tục công việc nạp , giải mã , và thực thi lệnh …

Rồi , thế là bạn đã tạo xong 1 system call mới là Create(char x); rồi đó !!! Mọi nhà lập trình phần mềm đều có thể sử dụng system call này của bạn để viết phần mềm ^__^ !! Các system call khác thì sẽ tương tự y như thế !!*

VII - Tiến trình

Tiến trình được nạp vào bộ nhớ như thế nào ?

Xin mời các bạn quay lại câu lệnh

./userprog/nachos –rs 1023 –x ./test/halt

Trong đó ./userprog/nachos thể hiện rằng mở chương trình nachos lên , nạp vào bộ nhớ rồi chạy , y như bất kỳ một chương trình nào khác trên Linuz như Kwrite , FireFox …

./test/halt thể hiện rằng mở chương trình halt lên để nạp vào bộ nhớ máy MIPS ảo rồi chạy

Tiến trình chạy cùng khi mở NachOS lên này , gọi là tiến trình đầu tiên . Hiện tại , NachOS là hệ điều hành đơn nhiệm , chỉ cho phép chạy 1 tiến trình thôi .

Một tiến trình bao gồm nhiều thành phần :

```
class Thread {
  private:
    // NOTE: DO NOT CHANGE the order of these first two members.
    // THEY MUST be in this position for SWITCH to work.
    int* stackTop;                  // the current stack pointer
    int machineState[MachineStateSize];  // all registers except for stackTop

  public:
    Thread(char* debugName);        // initialize a Thread debugName chính là tên
của Thread đó ~ đường dẫn tương đối đến file thực thi của Thread
    Thread(char* debugName, int priority);// init with a given priority
    ~Thread();                      // deallocate a Thread
                                // NOTE -- thread being deleted
                                // must not be running when delete
                                // is called

    // basic thread operations

    void Fork(VoidFunctionPtr func, int arg);   // Make thread run (*func)(arg)

    void Yield();                       // Relinquish the CPU if any
                                // other thread is runnable Pause
    void Sleep();                       // Put the thread to sleep and
                                // relinquish the processor
    void Finish();                      // The thread is done executing

    void CheckOverflow();               // Check if thread has
                                // overflowed its stack
    void setStatus(ThreadStatus st) { status = st; }
    char* getName() { return (name); }
    void Print() { printf("%s, ", name); }
```

```
  private:
    // some of the private data for this class is listed above

    int* stack;                 // Bottom of the stack
                                // NULL if this is the main thread
                                // (If NULL, don't deallocate stack)
    ThreadStatus status;              // ready, running or blocked
    char* name; //Tên của tiến trình = đường dẫn tương đối đến file chương trình

    void StackAllocate(VoidFunctionPtr func, int arg);
                                // Allocate a stack for thread.
                                // Used internally by Fork()

#ifdef USER_PROGRAM
// A thread running a user program actually has *two* sets of CPU registers --
// one for its state while executing user code, one for its state
// while executing kernel code.

    int userRegisters[NumTotalRegs];      // user-level CPU register state

  public:
    void SaveUserState();           // save user-level register state
    void RestoreUserState();        // restore user-level register state

    AddrSpace *space;               //Vùng nhớ của tiến trình trên RAM ảo.
#endif
};
```

Để coi kỹ hơn cách tạo dựng một tiến trình đầu tiên , mời bạn vào threads/main.cc Trong threads/main.cc sẽ gọi đến hàm StartProcess trong **userprog/progtest.cc** Hàm StartProcess này làm những công việc sau:

- Nạp file.noff vào bộ nhớ ảo của máy MIPS (class AddrSpace)
- Trỏ vùng nhớ của currentThread vào vùng nhớ vừa mới nạp ở trên (con trỏ AddrSpace)
- Khởi tạo các thanh ghi
- Cho chạy máy ảo

```
void
StartProcess(char *filename)
{
    OpenFile *executable = fileSystem->Open(filename);//Mở file .noff
    AddrSpace *space;

    if (executable == NULL) {
      printf("Unable to open file %s\n", filename);
      return;
    }
    space = new AddrSpace(executable);//Nạp file .noff vào bộ nhớ
    currentThread->space = space; //*****Chú ý - Nói sau*****

    delete executable;              // close file
```

```
    space->InitRegisters();         // set the initial register values
    space->RestoreState();          // load page table register

    machine->Run();                 // jump to the user progam
    ASSERT(FALSE);                  // machine->Run never returns;
                                // the address space exits
                                // by doing the syscall "exit"
}
```

Ở trên là cách tiến trình đầu tiên được nạp vào hệ thống . Các tiến trình không phải là đầu tiên , cũng tương tự như trên nhưng sẽ khác 1 chút xíu về **nơi khai báo hàm StartProcess (còn tùy , thường các tiến trình sau gọi StartProcess trong exception.cc , hàm này do mình copy từ hàm StartProcess gốc trong userprog/progtest.cc)**

Đa chương

Để NachOS biến thành một hệ điều hành đa nhiệm , bạn phải nạp thêm 2,3… tiến trình khác cho chúng chạy "song song". Để làm được điều đó , bạn phải chú ý những điều sau

- Từ tiến trình đầu tiên , bạn sẽ tạo ra các tiến trình con qua system call Exec()
- Chú ý số lượng tiến trình , kích thước của 1 chương trình nạp vào RAM , vì RAM của máy ảo MIPS hơi nhỏ , bạn có thể reisze lại RAM này (**machine.h**)
- Ta phải quan tâm đến việc nạp chương trình vào bộ nhớ sao cho ta có thể nạp các tiến trình vào bộ nhớ mà không xảy ra việc nạp đè lên vùng nhớ của tiến trình khác . Điều đó có thể gây nên sụp đổ hệ thống

  ➔ Giải quyết bằng kỹ thuật phân trang : Paging

  Xem trong hàm khởi tạo **AdrressSpace** , việc nạp file .noff vào tiến trình , ta sẽ chỉnh lại cách nạp file .noff vào bộ nhớ. Chỉnh lại thế nào đây ? Cái đó bạn phải tự suy nghĩ , đó cũng thật đơn giản thôi mà , với điều kiện bạn phải nắm kỹ kỹ thuật Paging . 1 trang (Page) thì ứng với 1 khung trang (Frame) . Khung trang trong NachOS tên là PhysPages – trang vật lý . Cách làm giống như tìm các lỗ trống để nhét các hòn bi vào mà thôi . Gợi ý : Bạn sử dụng

class BitMap để đánh dấu các khung trang còn trống , class BitMap giống như mảng các cờ hiệu thôi mà , bạn tự tìm hiểu nhé

- Và vấn đề cuối cùng liên quan đến hàm Thread::Fork (đọc kỹ hơn trong tài liệu tiếng Anh .nachos_study_book)!!
    - Các bạn chú ý trong thân hàm StartProcess , có dòng mà mình gạch đỏ chú ý ở trên `currentThread->space = space` . Như mình đã nói ở điều trên cùng : "Từ tiến trình đầu tiên , bạn sẽ tạo ra các tiến trình con qua system call Exec()" . Như vậy , currentThread ở đây đang trỏ đến tiến trình cha đầu tiên này , nếu mà trúng câu lệnh này , thì toàn bộ vùng nhớ của tiến trình cha sẽ bị trỏ hết vào con , thế thì không được. Để giải quyết vấn đề này , trong Exec , ta tạo 1 new Thread x rồi cho x->**Fork**(*con trỏ đến hàm StartProcess (`VoidFunctionPtr`) , ID của tiến trình – nếu có (`int`)*). Hàm Fork này làm cái gì ?? Nó làm những thứ sau
        - Cấp phát vùng nhớ Stack cho tiến trình . Trong lúc cấp phát vùng nhớ Stack này , Fork gán con trỏ hàm `VoidFunctionPtr` , và số nguyên `int` (**số nguyên này là tham số của hàm được trỏ bởi `VoidFunctionPtr`** , bạn có thể tùy biến là một con trỏ trỏ đến các kiểu dữ liệu khác tùy bạn thiết kế hàm) vào các **thanh ghi đặc biệt**
        - Chuyển tiến trình vào ReadyList

➔ Vậy trong thân hàm Exec() , hàm StartProcess không được thực thi , mà nó chỉ thực thi khi mà tiến trình mới được điều phối nắm CPU để chạy . Tức , trong Exec() , chỉ tạo ra 1 Thread con mới , trong trắng hoàn toàn , nằm trong ReadlyList , Thread con chỉ được nạp file .noff của nó vào bộ nhớ và khởi tạo các thanh ghi khi nó nắm quyền cầm CPU . Sau đó , CPU lần lượt sẽ được hệ điều hành điều phối theo cơ chế Round Robin , CPU cho tiến trình cha đầu tiên xài 1 chút , hết lượt đến tiến trình con xài một chút rồi lại quay sang cha cho đến khi 1 tiến trình kết thúc thì nó tắt máy luôn (Bởi vì hiện giờ

mình chỉ quan tâm là nó chạy được đa chương thôi , chứ chưa quan tâm đến vấn đề đồng bộ hóa mà)

➔ Vậy ta có thể kết hợp 2 hàm Fork và StartProcess này lại làm 1 cho tiện được ko ? Tôi không muốn Thread mới tạo iu quí của tôi phải đợi đến lúc nó nhận điều phối thì mới được nạp vào bộ nhớ và khởi tạo các thanh ghi . Tôi muốn những điều đó được làm hết trong hàm Exec cơ … ➔ Bạn có thể nghiên cứu phần khởi tạo các thanh ghi trong AddressSpace ☺ , việc này sẽ làm thay đổi các thanh ghi của máy MIPS ảo , dẫn đến tiến trình cha (currentThread) bị sụp đổ ☺ , tuy nhiên , có lẽ khả năng của mình hạn hẹp , chưa thông được hết , có thể vẫn có cách làm đó , bạn thử nghĩ xem ☺

VIII - Lời kết và những câu hỏi

Chà, đến đây là bạn đã nắm được những khái niệm căn bản của NachOS rồi , tới 30% lận đó , phần còn lại bạn có thể vô tư tìm hiểu

Hãy suy nghĩ tiếp đi nào … nào là **quản lý** hệ thống file , nhập xuất I/O , **quản lý** tiến trình , đồng bộ hóa …

Hãy nhớ một điều , bạn là nhà lập trình nhân hệ điều hành vĩ đại , bạn có quyền thay đổi mọi thứ theo ý thích của mình , ví dụ như bạn nhận thấy class Thread cần phải có một ID định danh cho từng tiến trình thì bạn sẽ thêm vào đó int ID chẳng hạn …

Bạn còn là một nhà sáng tạo phần mềm ứng dụng tuyệt vời , bạn có thể sử dụng các hàm system call của NachOS để viết chương trình nhập xuất tạo file … , đồng bộ hóa các chương trình chạy đồng thời , thực hiện các bài toán đồng bộ hóa kinh điển…

Chỉ có điều , hãy sáng tạo và thay đổi mọi thứ một cách khôn ngoan 😐

Sau đây mình trả lời một số câu hỏi hay thắc mắc nhất trong quá trình làm NachOS

Vấn đề biến toàn cục:

1) Tôi có thể tạo biến toàn cục thế nào ?

- Bạn mở thread/system.h và system.cc , bạn có thấy biến machine được khai báo và hủy thế nào không , làm theo nó đi ^_^

Vấn đề tạo một class mới:

1) Tôi có thể tạo 1 class mới của tôi thế nào ?

- Bạn hãy tạo newclass.h và newclass.cc , tùy vào ý nghĩa của chúng mà bạn có thể bỏ vào thư mục thích hợp , thread, userprog …
- Trong newclass.h , nếu bạn có sử dụng lớp nào khác thì bạn #include file .h của lớp đó vào . Trong file .cc bạn chỉ cần #include "newclass.h" và "system.h" nếu bạn có sử dụng đến biến toàn cục

- Ở đây ,mình muốn lớp newclass của mình có thể sủ dụng được bởi bất cứ mã nguồn nào trong thư mục **userprog**

- Mở tập tin Makefile.common (trong nachos-3.4/**code**) , cuối đoạn bắt đầu dòng USERPROG_H = thêm vào đường dẫn tương đối đến ../…/newclass.h , chú ý , nhớ thêm \ cho dòng trên , **dòng cuối cùng không có **

- Cuối đoạn USERPROG_C = thêm tương tự như trên đường dẫn tương đối đến newclass.cc

- Cuối đoạn USERPROG_O = thêm vào newclass.o

➔ Làm như thể để khi biên dịch , GCC 3.x biên dịch các file đó , có thể sử dụng chúng ở trong thư mục **userprog**

Vấn đề đa chương:

1) NachOS điều phối theo cơ chế nào ?

- NachOS sử dụng cơ chế điều phối Round Robin với quantum là một con số ngẫu nhiên có seed ~ **-rs 1023** (hay số nào cũng được tùy bạn , -rs 200, -rs 100 …). Nó sẽ chọn tiến trình theo kiểu **First In First Serve**

Vấn đề đồng bộ :

1) Nếu tôi gọi 1 system call, trong system call đó, tức trong KERNEL MODE, tôi bị lặp vô hạn thì có bao giờ máy nó quay về USER MODE (giai đoạn nạp và xử lý từng lệnh) lại được không ?

- Không ☺ , bị lặp thì nó lặp hoài luôn

2) Vậy thế sao trong Semaphore::P() tôi thấy có vòng lặp while lặp hoài mà , nếu như 1 tiến trình khi Semaphore::P() mà bị ngủ thì sao ?

- Vòng lặp while() trong Semaphore::P() không bị lặp vô hạn đâu , nếu như tiến trình vào vòng lặp đó , hàm hợp ngữ SWITCH sẽ chuyển CPU qua cho tiến trình khác sử dụng , bạn thử Debug xem nào ☺ [xem trong nachos studybook phần tiến trình]. Và mình lưu ý rằng , ở đây , bạn chỉ cần việc sử dụng P() và V() thôi , có thể không cần quan tâm bên trong đó thế nào , cứ hiểu chúng y như lúc học lý thuyết ,tức là gọi P() , nếu ngủ thì CPU sẽ chuyển ngay cho tiến trình khác luôn , khi được đánh thức dậy thì nó chạy

tiếp ngay khúc dưới của P() . Nhớ quan tâm thêm cả vaule của biến semaphore nữa nhé.

3) Tôi thấy hàm Dow và Up trong lý thuyết ngược với P() và V() trong NachOS

- Xin mời bạn mở nachos_study_book ra coi , trong đó có ghi rõ , 2 phương pháp này hoàn toàn giống nhau về ý nghĩa ^_^ (Khỏi coi cũng dc , cứ biết thế thoy, coi hồi điên lun á ☺)

Còn lại một số câu hỏi khác , các câu hỏi khác mình thấy hoàn toàn có thể tìm thấy câu trả lời thỏa đáng trong tài liệu tiếng Anh hay trên web , các bạn ráng tìm nhé ^__^

GoodLuck !!!

The End