

Project 01

HỆ ĐIỀU HÀNH

Exceptions và các system calls đơn giản

Giảng viên hướng dẫn

Giảng viên lý thuyết

Lê Quốc Hòa

Giảng viên thực hành

Chung Thùy Linh

Thông tin nhóm

19127292

Nguyễn Thanh Tình

19127339

Phạm Chi Bảo

Lời cảm ơn

Hệ điều hành là thành phần quan trọng không thể thiếu trong hầu hết tất cả các thiết bị điện tử hiện nay. Nghiên cứu hệ điều hành sẽ giúp chúng ta có cái nhìn chuyên sâu về cách hoạt động của chúng. Tuy nhiên, đối với người mới bắt đầu, việc tìm hiểu những hệ điều hành mới và phổ biến nhất hiện nay sẽ rất khó khăn và tốn nhiều thời gian hơn. Vì lý do đó, **NachOS** ra đời là một hệ điều hành đơn giản để tìm hiểu và nghiên cứu, cung cấp kiến thức cơ bản để xây dựng các thành phần phức tạp hơn của các hệ điều hành sau này.

Chân thành cảm ơn thầy **Lê Quốc Hòa** và cô **Chung Thùy Linh** đã cung cấp cho cả lớp nói chung và nhóm chúng em nói riêng **Project 01 - Exceptions và các system calls đơn giản** - một đề án thật sự rất bổ ích và lý thú. Và cũng chân thành cảm ơn cô **Chung Thùy Linh** một lần nữa vì những tài liệu và những buổi seminar hướng dẫn chi tiết đầy hữu ích từ cô, giúp chúng em hoàn thành được đề án này.

Thành phố Hồ Chí Minh, ngày 13 tháng 3 năm 2022

Tập thể nhóm

Mục lục

Phần 1. Hiểu mã Hệ điều hành NachOS	3
Cài đặt, biên dịch, kiểm tra hoạt động	3
Các thư mục	4
Các tập tin quan trọng	5
Phần 2. Hiểu thiết kế Hệ điều hành NachOS	7
Phần 3. Exceptions và system calls	10
1. Xử lý exceptions	10
2. Tăng giá trị PC	11
Quy trình thêm một system call (yêu cầu 3 - 9)	12
Quy trình viết chương trình mức người dùng (yêu cầu 10 - 12)	12
3. System call: ReadNum	13
4. System call: PrintNum	13
5. System call: ReadChar	14
6. System call: PrintChar	14
7. System call: RandomNum	14
8. System call: ReadString	15
9. System call: PrintString	16
10. Chương trình: help	17
11. Chương trình: ascii	17
12. Chương trình: sort	17
Tài liệu tham khảo	18

Phần 1. Hiểu mã Hệ điều hành NachOS

Cài đặt, biên dịch và kiểm tra hoạt động

NachOS hiện nay là hệ điều ở mức một người dùng, chưa hỗ trợ đa chương, chạy bằng chương trình viết bằng ngôn ngữ C tại một thời điểm. Để dễ thao tác, thư mục chính để lập trình hệ điều hành NachOS sẽ là thư mục **NachOS-4.0/code**.

Cài đặt NachOS

Để cài đặt được NachOS-4.0, chúng ta sử dụng hệ điều hành Ubuntu, sau đó bật tính năng hỗ trợ cho hệ điều hành 32bit (NachOS chỉ chạy được trên nền 32bit), đồng thời cài đặt phiên bản GCC/G++ phù hợp.

Sau đó, đơn giản là tải các file mã nguồn của NachOS-4.0 và giải nén.

Biên dịch

Để biên dịch từ mã nguồn thành hệ điều hành NachOS có thể thực thi, chúng ta sẽ vào thư mục **build.linux**, sau đó gõ các câu lệnh sau:

```
$ make depend
```

```
$ make
```

Kiểm tra hoạt động

Chúng ta có thể kiểm tra hoạt động của NachOS bằng cách chạy chương trình **halt**, chương trình halt sẽ yêu cầu hệ điều hành NachOS tắt máy (máy ảo giả lập chạy NachOS).

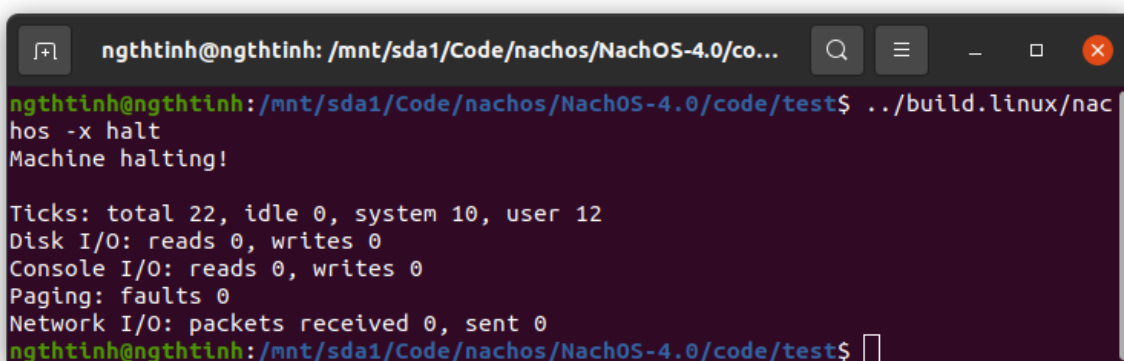
Đầu tiên vào thư mục **test**, mở Terminal tại đây và chạy câu lệnh:

```
$ make all
```

Sau đó, chạy câu lệnh gọi chương trình halt như sau:

```
$ ../build.linux/nachos -x halt
```

Kết quả sẽ như sau:

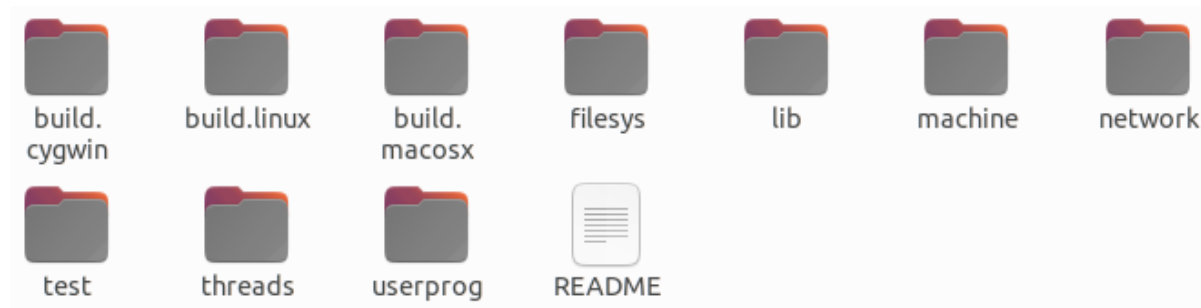


```
ngthtinh@ngthtinh: /mnt/sda1/Code/nachos/NachOS-4.0/co...
ngthtinh@ngthtinh:/mnt/sda1/Code/nachos/NachOS-4.0/code/test$ ../build.linux/nachos -x halt
Machine halting!

Ticks: total 22, idle 0, system 10, user 12
Disk I/O: reads 0, writes 0
Console I/O: reads 0, writes 0
Paging: faults 0
Network I/O: packets received 0, sent 0
ngthtinh@ngthtinh:/mnt/sda1/Code/nachos/NachOS-4.0/code/test$
```

Các thư mục

Thư mục chính của hệ điều hành NachOS nằm ở **NachOS-4.0/code**, trong đó mỗi thư mục đều đóng vai trò quan trọng cho hoạt động của NachOS.



build.cygwin

Chức năng: Biên dịch mã nguồn thành hệ điều hành NachOS có thể chạy trên Windows thông qua Cygwin.

Nội dung: Hai file makefile giúp quá trình biên dịch trở nên dễ dàng.

build.linux

Chức năng: Biên dịch mã nguồn thành hệ điều hành NachOS có thể chạy trên Linux. Đây là thư mục quan trọng cho đồ án này.

Nội dung: Chứa hệ điều hành NachOS, đồng thời chứa những thư viện hỗ trợ cho hoạt động của NachOS.

build.macosx

Chức năng: Biên dịch mã nguồn thành hệ điều hành NachOS có thể chạy trên Mac OS X.

Nội dung: Hai file makefile giúp quá trình biên dịch trở nên dễ dàng.

filesys

Chức năng: Chứa file hệ thống của hệ điều hành NachOS.

Nội dung: Các thư viện hệ thống của NachOS như filesys.

lib

Chức năng: Chứa các thư viện hỗ trợ cho hệ điều hành NachOS.

Nội dung: Các thư viện như bitmap để hỗ trợ phân trang,

machine

Chức năng: Giả lập các thành phần quan trọng của hệ điều hành NachOS, giả lập các phần cứng như đĩa, đồng hồ, v.v...

Nội dung: Chứa các thành phần dùng để giả lập phần cứng hệ điều hành NachOS như đĩa, đồng hồ, v.v...

network

Chức năng: Cung cấp khả năng giao tiếp với mạng cho NachOS.

Nội dung: Các thư viện hỗ trợ giao tiếp mạng.

test

Chức năng: Nơi cài đặt và lưu trữ các chương trình được lập trình ở mức người dùng, có thể gọi được các hàm hệ thống từ đây.

Nội dung: Các chương trình mức người dùng viết bằng ngôn ngữ C, được biên dịch theo kiến trúc MIPS, chương trình halt là một ví dụ.

threads

Chức năng: Hỗ trợ đa chương, đa luồng cũng như các thư viện hỗ trợ tương tác với hệ điều hành.

Nội dung: Các thư viện hỗ trợ luồng, hỗ trợ tương tác với hệ điều hành.

userprog

Chức năng: Là một thư mục quan trọng của hệ điều hành NachOS, chứa các syscall để chương trình mức người dùng có thể thực hiện lời gọi hệ thống.

Nội dung: Chứa các file quan trọng của NachOS, đặc biệt, có hai file rất quan trọng đó là exception.cc để cài đặt các syscall và syscall.h để định nghĩa các syscall đó.

Các tập tin quan trọng

progtest.cc

Kiểm tra các thủ tục để chạy chương trình người dùng.

File này không còn tồn tại trong NachOS-4.0.

syscall.h

Thư mục: userprog.

Định nghĩa các system calls ở mức kernel mà chương trình người dùng có thể gọi, thông qua các define SC và các prototype.

exception.cc

Thư mục: userprog.

Xử lý các system calls và các exception khác ở mức người dùng. Những system calls này sẽ được coi là một ngoại lệ của hệ thống và được xử lý ở bên trong file exception.cc này.

bitmap.*

Thư mục: lib.

Các hàm xử lý cho lớp bitmap, nó hữu ích cho việc lưu vết ô nhớ vật lý, file này quan trọng cho đề án sau.

openfile.h

Thư mục: filesys.

Định nghĩa các hàm hệ thống file NachOS. Trong đề án này chúng ta sẽ sử dụng lời gọi thao tác với file trực tiếp từ Linux, trong đề án khác chúng ta sẽ triển khai hệ thống file trên ổ đĩa giả lập nếu kịp thời gian.

translate.*

Thư mục: machine.

Phiên bản NachOS chúng ta đang sử dụng, được giả sử rằng mỗi địa chỉ ảo cũng giống hệt địa chỉ vật lý, điều này giới hạn khả năng đa chương của hệ điều hành. Đề án sau chúng ta sẽ viết lại file để NachOS hỗ trợ đa chương.

machine.*

Thư mục: machine.

Mô phỏng các thành phần của máy tính khi thực thi chương trình người dùng như bộ nhớ chính, thanh ghi, v.v...

mipssim.cc

Thư mục: machine.

Mô phỏng tập lệnh của MIPS R2/3000 processor.

console.*

Thư mục: machine.

Mô phỏng thiết bị đầu cuối sử dụng UNIX files. Một thiết bị có đặc tính đơn vị dữ liệu theo byte, đọc và ghi các byte cùng một thời điểm, các bytes đến một cách bất đồng bộ.

synchconsole.*

Thư mục: userprog.

Nhóm hàm cho việc quản lý nhập xuất I/O theo dòng trong NachOS.

Phần 2. Hiểu thiết kế Hệ điều hành NachOS

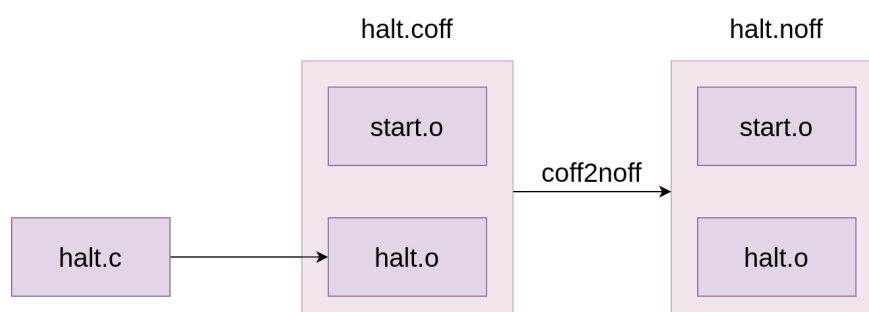
Để hiểu thiết kế và hoạt động hệ điều hành NachOS, chúng ta sẽ thử phân tích một chương trình ở mức người dùng (ví dụ như **halt.c**), rằng khi khởi chạy thì nó sẽ khởi động thế nào, hoạt động ra sao. Thông qua từng thư mục, từng file trong NachOS, file nào sẽ được gọi đến file nào, chạy ra sao, xử lý như thế nào. Giải đáp được các câu hỏi này, ta sẽ hiểu rõ thiết kế của NachOS.

Về chương trình **halt.c**, chương trình chỉ gọi một hàm duy nhất **Halt()** để tắt máy ảo NachOS:

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

Trước khi chương trình này có thể thực thi được bởi NachOS, nó phải được biên dịch (không cần biên dịch lại cả NachOS). NachOS cung cấp một môi trường giả lập để chạy các chương trình C, xem qua Makefile trong thư mục test, ta thấy chương trình biên dịch thông qua GCC tạo ra các file object, sau đó chuyển sang định dạng đặc biệt của NachOS nhờ **coff2noff**. Cụ thể:

- Chương trình **halt.c** được biên dịch thành file object **halt.o**.
- Tập tin **halt.o** này sẽ được liên kết với tập tin **start.o** để tạo thành tập tin **halt.coff**, là định dạng thực thi được trên hệ điều hành Linux cho kiến trúc máy MIPS. (File **start.o** được sử dụng nằm mục đích đơn giản hóa thay vì dùng toàn bộ thư viện C).
- Tập tin **halt.coff** được chuyển thành **halt.noff**, là định dạng thực thi được trên hệ điều hành NachOS cho kiến trúc máy MIPS, sử dụng tiện ích được cung cấp sẵn **coff2noff**.



NachOS cung cấp một CPU giả lập, thực tế CPU giả lập này giống hệt CPU thật (MIPS-32 bit chip), nhưng chúng ta không thể chỉ thực thi chương trình như một tiến trình bình thường của UNIX, bởi vì chúng ta muốn kiểm soát có bao nhiêu lệnh được thực hiện trong một đơn vị thời gian, không gian địa chỉ làm việc như thế nào, các interrupt và exceptions (system calls) được xử lý như thế nào.

Đi sâu vào hoạt động của NachOS, ta cần hiểu **kernel mode (system space)** và **user mode (user space)**. Mỗi chương trình trong hệ thống phải có các thông tin cục bộ của nó, bao gồm program counter, registers, stack pointers, và các file system handler. Mặc dù user program truy cập các thông tin cục bộ của nó, nhưng hệ điều hành điều khiển các truy cập này, hệ điều hành đảm bảo các yêu cầu từ user program tới kernel không làm cho hệ điều hành sụp đổ. Việc chuyển quyền điều khiển từ user mode thành system mode được thực hiện thông qua system calls, software interrupt, system trap. Trước khi gọi một lệnh trong hệ thống thì các tham số truyền vào cần được nạp vào các thanh ghi của CPU. Để chuyển một biến mang giá trị, tiến trình chỉ việc ghi giá trị vào thanh ghi. Để chuyển một biến tham chiếu, thì giá trị lưu trong thanh ghi được xem như là “user space pointer”. Bởi vì user space pointer không có ý nghĩa đối với kernel, mà chúng ta cần là chuyển nội dung từ user space vào kernel sao cho ta có thể xử lý dữ liệu này. Khi trả thông tin từ system về user space, thì các giá trị phải đặt trong các thanh ghi của CPU.

Khi thực thi một chương trình trên NachOS, thì chương trình **start** luôn được thực thi trước, và quá trình thực thi của halt sẽ được thực hiện như sau:

- Gọi hàm **main** trong **halt**.

start.s:

```
...  
jal main # nhảy đến main trong halt  
...
```

- Khi đang thực hiện **main** trong **halt**, gặp hàm **Halt()**, quay về **start.s**:

halt:

```
...  
jal Halt # nhảy đến Halt trong start.s  
...
```

- Nạp mã của Syscall Halt vào thanh ghi số 2, sau đó thực hiện trao quyền điều khiển cho hệ thống.

Sau khi thực hiện xong syscall, nhảy đến \$31 để tiếp tục chương trình.

start.s:

```

Halt:
    addiu $2, $0, SC_Halt # Nạp mã SC_Halt
    syscall # Trao quyền cho hệ thống xử lý
    jal $31 # Tiếp tục chương trình
.end Halt

...

```

- Hiện tại vẫn còn đang ở **User mode**, lúc này hệ điều hành nhận yêu cầu trao quyền, nó được xếp vào loại OP_SYSCALL trong **missim.cc**, lúc này một ngoại lệ được đẩy lên:

mipssim.cc:

```

...
case OP_SYSCALL:
    RaiseException(SyscallException, 0);
...

```

- Hàm **RaiseException()** được cài đặt trong **machine.cc** sẽ trao quyền cho hệ thống, mọi thứ sẽ được xử lý trong file **exception.cc** sau đó:

machine.cc:

```

void RaiseException(ExceptionType which, ...)
{
    ...
    ExceptionHandler(which);
    ...
}

```

- Lúc này đã vào được **System mode**, hệ điều hành kiểm tra phần xử lý của syscall này và thực hiện các chỉ thị:

exception.cc:

```

if ((which == SyscallException) && (type == SC_Halt))
{
    ...
    interrupt->Halt();
    ...
}

```

- Tuy nhiên, có một điều cần lưu ý, là sau khi thực hiện xong system call thì hệ điều hành sẽ trao quyền trở lại cho người dùng, CPU sẽ thực hiện nạp lệnh tiếp theo. Nếu lúc này PC vẫn còn giá trị cũ, CPU sẽ đọc lại chỉ thị mà đã được thực hiện, từ đó xảy ra vòng lặp vô tận. Vì thế khi cài đặt syscall, lập trình viên cần lưu ý tăng giá trị của Program Counter lên.

Phần 3. Exceptions và system calls

1. Xử lý exceptions

Đọc file machine/machine.h

Các Exceptions được thể hiện dưới dạng enum có tên là **ExceptionType**, chúng bao gồm các exceptions sau:

- **NoException**: Mọi thứ đều ổn.
- **SyscallException**: Ngoại lệ phát sinh từ một chương trình gọi syscall.
- **PageFaultException**: Lỗi trang.
- **ReadOnlyException**: Ghi vào vùng nhớ chỉ ghi.
- **BusErrorException**: Lỗi bus.
- **AddressErrorException**: Lỗi địa chỉ.
- **OverflowException**: Lỗi tràn số vì cộng trừ.
- **IllegalInstrException**: Lỗi chỉ thị.
- **NumExceptionTypes**: Tổng số các loại exceptions.

Hầu hết các exceptions trên đều là **run-time errors**, khi các exceptions này xảy ra thì user program không thể được phục hồi. Trường hợp đặc biệt duy nhất là **NoException** sẽ trả quyền điều khiển về hệ điều hành, còn **SyscallExceptions** sẽ được xử lý bởi các hàm chúng ta viết cho user system calls. Với tất cả exceptions khác, hệ điều hành hiển thị ra một thông báo lỗi và Halt hệ thống.

Viết lại file exception.cc

Để xử lý các exceptions, ta sẽ đến với hàm **ExceptionHandler**.

Ta sẽ cần phải căn lề lại các dòng code để dễ nhìn hơn.

Hiện tại, NachOS chỉ hỗ trợ **SyscallException**, những exceptions còn lại, NachOS chỉ đơn giản là hiển thị thông báo "**Unexpected user mode exception**". Vậy chúng ta cần bổ sung các exceptions được liệt kê ở trên vào điều kiện rẽ nhánh **switch** (which).

- **NoException**: trả quyền điều khiển cho hệ điều hành bằng cách return.
- **SyscallException**: Hiện tại giữ nguyên.
- Các exceptions còn lại:
 - Hiển thị thông báo lỗi.
 - Halt hệ thống.
 - Đánh dấu **ASSERTNOTREACHED**.

2. Tăng giá trị PC

Tất cả các system calls (không phải Halt) sẽ yêu cầu NachOS tăng Program Counter trước khi system call trả kết quả về. Nếu không lập trình đúng phần này thì NachOS sẽ bị vòng lặp gọi thực hiện system call này mãi mãi. Cũng như các hệ thống khác, MIPS xử lý dựa trên giá trị của Program Counter, vì vậy chúng ta cần phải viết mã để tăng giá trị biến Program Counter.

Tìm đoạn mã tăng giá trị biến Program Counter

Đoạn mã này đã được cài đặt sẵn ngay trên chính trong file [exception.cc](#), cụ thể là ở [SC_Add](#). (Chỉ có ở NachOS 4.0 được tải về từ moodle môn học).

Phân tích [SC_Add](#):

- Đầu tiên thông báo [DEBUG](#) là hiện tại đang cộng hai số.
- Thực hiện cộng hai số từ thanh ghi số 4 và thanh ghi số 5.
- Lưu kết quả vào thanh ghi số 2.
- **Chỉnh sửa điểm trả về (Modify return point).**

Chính bước chỉnh sửa điểm trả về này, là công đoạn tăng Program Counter. Cụ thể hơn, công đoạn tăng Program Counter sẽ bao gồm các bước sau:

- Đặt [PrevPCReg](#) thành giá trị của [PCReg](#).
- Tăng thanh ghi [PCReg](#) lên 4 đơn vị. (Lý do tại sao lại 4 đơn vị, là vì NachOS là hệ điều hành 32 bit, kích thước địa chỉ mỗi lệnh sẽ là 4 bytes!)
- Đặt [NextPCReg](#) thành giá trị của [PCReg](#) nhưng cộng thêm 4 đơn vị.
- Các biến [PrevPCReg](#), [PCReg](#), [NextPCReg](#) được khai báo ở [machine.h](#).

Viết hàm tăng Program Counter

Copy đoạn mã tăng Program Counter ở trong [SC_Add](#), và viết thành một hàm, đặt tên là [IncreasePC\(\)](#). Từ nay khi muốn tăng giá trị của Program Counter, chỉ việc gọi hàm [IncreasePC\(\)](#).

Vị trí đặt hàm [IncreasePC\(\)](#): Quy trình của từng system calls sẽ như sau:

- Xử lý yêu cầu.
- Lưu kết quả (nếu cần)
- Tăng Program Counter.
- Return.

Quy trình thêm một system calls (yêu cầu 3 - 9)

Từ yêu cầu 3 đến yêu cầu 9, đều có thể được gọi chung là thêm system calls. Quy trình này giống nhau ở mọi system calls. Các công việc này bao gồm:

- Trong tập tin [userprog/syscall.h](#), thêm prototype cho syscall, đồng thời define mã cho syscall này.
- Trong tập tin [test/start.s](#), thêm một đoạn code nạp mã syscall này vào thanh ghi \$2 và thực hiện gọi syscall, sau đó nhảy đến thanh ghi \$31. Hoặc đơn giản hơn, copy một đoạn lệnh có sẵn cho một system call và chỉnh sửa lại tên.
- Trong tập tin [userprog/exception.cc](#), thêm phần xử lý system call vào [switch](#) (type). Cấu trúc của đoạn mã cho system call này sẽ như sau:
 - Thông báo **DEBUG** là thực hiện system call.
 - Cài đặt xử lý system call.
 - Chuẩn bị kết quả system call (nếu cần).
 - Tăng Program Counter, sau đó return.
 - Đánh dấu **ASSERTNOTREACHED()**.
 - Break system call.
- Sau khi thực hiện xong, biên dịch lại NachOS.

Quy trình viết chương trình mức người dùng (yêu cầu 10 - 12)

Từ yêu cầu 10 đến yêu cầu 12, đều có thể được gọi chung là viết chương trình mức người dùng. Quy trình này giống nhau ở mọi chương trình mức người dùng. Các công việc này bao gồm:

- Viết chương trình mức người dùng bằng ngôn ngữ C trong thư mục test. Lưu ý: cần include thư viện [syscall.h](#), đồng thời, ở cuối mỗi chương trình cần **Halt()** hệ thống.
- Chỉnh sửa Makefile:
 - Thêm chỉ thị trong tab **all**.
 - Thêm chỉ thị biên dịch mã nguồn từ ngôn ngữ C thành file object, sau đó kết hợp với file start.o để tạo thành file coff, cuối cùng dùng tiện ích coff2noff để biên dịch thành file noff.
- Trong thư mục test, gõ **make all**.
- Chạy chương trình ở mức người dùng.

3. System call: ReadNum

Cài đặt system call **int ReadNum()**.

System call ReadNum sẽ sử dụng lớp **SynchConsoleIn** để đọc một số nguyên do người dùng nhập vào. Nếu giá trị người dùng nhập không phải là số nguyên thì trả về số 0. Có một vấn đề, đó là lớp **SynchConsoleIn** chỉ có hàm **GetChar()** hỗ trợ đọc từng ký tự, vì thế ta cần tạo một chuỗi **buffer** để lưu trữ số nhập từ console.

Người dùng có thể gây sập hệ điều hành nếu:

- Nhập số nguyên quá lớn.
- Nhập ký tự thay vì nhập số nguyên.
- Không nhập gì cả ngoài việc nhấn Enter.

Vậy, để giải quyết bài toán này, ta cần làm các bước sau đây:

- Tạo buffer rỗng.
- Đọc các ký tự từ console bằng lớp **SynchConsoleIn**.
- Nếu số nguyên quá lớn, trả về 0.
- Ngược lại, dùng vòng lặp để chuyển chuỗi về số. Lưu ý: Trong quá trình lặp mà phát hiện có ký tự không phải ký tự số, lập tức dừng vòng lặp và trả kết quả về 0.
- Lưu kết quả vào thanh ghi \$2.

4. System call: PrintNum

Cài đặt system call **void PrintNum(int number)**.

System call PrintNum sẽ sử dụng lớp **SynchConsoleOut** để xuất một số nguyên ra màn hình. Có một vấn đề, đó là lớp **SynchConsoleOut** chỉ có hàm **PutChar()** hỗ trợ xuất từng ký tự, vì thế ta cần in từng chữ số ra màn hình.

Vậy, để cài đặt được system call này, ta thực hiện như sau:

- Đọc con số cần được in ra màn hình từ thanh ghi \$4.
- Nếu là số 0, in ký tự '0' bằng lớp **SynchConsoleOut**. Hoàn thành.
- Ngược lại:
 - Nếu là số âm, in ký tự '-' trước, sau đó lấy trị tuyệt đối của số này.
 - Thực hiện một vòng lặp, để in các chữ số của số này theo thứ tự từ trái sang phải. Trong đề án này, nhóm sử dụng một biến **exp** có giá trị là 10^x , nhờ đó có thể dễ dàng tách từng thành phần của con số một cách dễ dàng.

5. System call: ReadChar

Cài đặt system call **char ReadChar()**.

System call ReadChar sẽ sử dụng lớp **SynchConsoleIn** để đọc một ký tự do người dùng nhập vào.

Người dùng có thể gây sập hệ điều hành không? Câu trả lời là không, vì dù nhập thế nào cũng không sập hệ điều hành.

System call này khá đơn giản:

- Lớp **SynchConsoleIn** đã có sẵn hàm **GetChar()**.
- Dùng hàm này để nhập ký tự từ console.
- Sau đó lưu vào thanh ghi \$2.

6. System call: PrintChar

Cài đặt system call **void PrintChar(char character)**.

System call PrintChar sẽ sử dụng lớp **SynchConsoleOut** để xuất một ký tự ra màn hình.

System call này khá đơn giản:

- Lớp **SynchConsoleOut** đã có sẵn hàm **PutChar()**.
- Lấy giá trị là ký tự cần in từ thanh ghi \$4.
- Đưa vào hàm này để in ra console.

7. System call: RandomNum

Cài đặt system call **int RandomNum()**. System call RandomNum() sẽ trả về một số nguyên dương ngẫu nhiên.

Để thực hiện được system call này, ta cần sự hỗ trợ của hàm **rand()**. Cần lưu ý sử dụng **srand(time(NULL))** trước để tránh mọi số được sinh ngẫu nhiên đều là một số duy nhất.

Quá trình cài đặt rất đơn giản, bao gồm các bước như sau:

- Gọi **srand()**.
- Tạo một biến kết quả.
- Gán biến kết quả này bằng giá trị của hàm **rand()**.
- Lưu kết quả vào thanh ghi \$2.

8. System call: ReadString

System space to User space

Buffer là vùng nhớ thuộc **user space**, khi người dùng nhập chuỗi thì nội dung được lưu trữ ở **kernel space**, chúng ta cần viết một hàm tương ứng để chuyển dữ liệu từ **kernel space** qua **user space**.

Công cụ đặc lực để hỗ trợ thao tác này là: **kernel->machine->WriteMem()**.

Để thực hiện được công việc này, nhóm đã tham khảo đoạn code cài đặt hàm **System2User** trong [2] **Giao tiếp giữa Nachos và chương trình người dùng.pdf**, và ý tưởng của hàm này là:

- Kiểm tra độ dài buffer, nếu buffer có độ dài nhỏ hơn hoặc bằng 0 thì không cần thực hiện tiếp.
- Tạo một vòng lặp duyệt buffer (vòng lặp dừng khi buffer được đọc hết hoặc đọc tới ký tự '\0')
 - Lấy ký tự từ trong buffer ra.
 - Dùng **kernel->machine->WriteMem()** để ghi vào user space.
- Trả về kết quả là số byte thực tế đã copy.

System call: ReadString

Cài đặt system call **void ReadString(char buffer[], int length)**.

Chú ý rằng thanh ghi chỉ chứa số, vì vậy khi truyền vào string sẽ là địa chỉ.

Người dùng có thể gây sập hệ điều hành nếu:

- Nhập chuỗi rỗng.
- Nhập chuỗi quá dài.

Để giải quyết khả năng gây sập hệ điều hành trên, chỉ cần tuân theo quy tắc:

- Vẫn xem chuỗi rỗng là chuỗi, nhưng mọi thao tác trên chuỗi này không thực hiện, vì độ dài chuỗi là 0.
- Nhập chuỗi nhưng tuân theo độ dài tối đa đã cung cấp **length**. Ngoài việc dừng nhập chuỗi với phím Enter, ta cũng nên dừng nhập chuỗi khi đã đạt đến giới hạn này.

Cài đặt system call này cũng không quá phức tạp, bao gồm các bước sau:

- Tạo buffer, dùng lớp **SynchConsoleIn** để đọc từng ký tự cho buffer này.
- Dùng hàm **System2User** để đưa buffer này đến địa chỉ chuỗi.
- Làm sạch buffer.

9. System call: PrintString

User space to System space

Chuỗi là vùng nhớ thuộc **user space**, khi người dùng muốn in chuỗi thì nội dung được lưu trữ ở buffer trong **kernel space**, chúng ta cần viết một hàm tương ứng để chuyển dữ liệu từ **kernel space** qua **user space**.

Công cụ đặc lực để hỗ trợ thao tác này là: **kernel->machine->ReadMem()**.

Để thực hiện được công việc này, nhóm đã tham khảo đoạn code cài đặt hàm **User2System** trong [2] **Giao tiếp giữa Nachos và chương trình người dùng.pdf**, và ý tưởng của hàm này là:

- Kiểm tra độ dài giới hạn buffer, nếu giới hạn buffer có độ dài nhỏ hơn hoặc bằng 0 thì không cần thực hiện tiếp.
- Tạo một vòng lặp duyệt chuỗi ở mức người dùng (vòng lặp dừng khi chuỗi được đọc hết hoặc đọc tới ký tự '\0')
 - Lấy ký tự từ trong chuỗi ra.
 - Dùng **kernel->machine->WriteMem()** ghi vào buffer system space.
- Trả về địa chỉ của buffer cho system.

System call: PrintString

Cài đặt system call **void PrintString(char buffer[])**.

Chú ý rằng thanh ghi chỉ chứa số, vì vậy khi truyền vào string sẽ là địa chỉ.

Cài đặt system call này cũng không quá phức tạp, bao gồm các bước sau:

- Đọc địa chỉ của chuỗi.
- Sử dụng hàm **User2System()** để copy vùng nhớ từ user space xuống kernel space, để có dữ liệu cho system call.
- Tạo một vòng lặp duyệt từng ký tự:
 - Nếu là ký tự bình thường, sử dụng lớp **SynchConsoleOut** để in ra ký tự đó lên console.
 - Nếu là ký tự '\0', thoát vòng lặp.
- Giải phóng buffer.
- Hoàn tất việc in chuỗi.

Vậy là nhóm đã thực hiện cài đặt thành xong các system call. Chúng sẽ là những công cụ hỗ trợ đặc lực cho việc xây dựng chương trình ở mức người dùng.

10. Chương trình: help

Viết chương trình **help**, dùng để in ra các dòng giới thiệu cơ bản về nhóm và mô tả vắn tắt về chương trình sort và ascii.

Chương trình này rất đơn giản, vì thật ra chỉ việc gọi system call **PrintString()**.

Các thông tin nhóm đã in:

- Môn học, lớp học, tên trường.
- Giảng viên hướng dẫn.
- Thông tin thành viên trong nhóm, gồm MSSV và Họ tên.
- Thông tin vắn tắt về hai chương trình: **ascii** và **sort**.

11. Chương trình: ascii

Viết chương trình **ascii** để in ra bảng mã ascii (bắt buộc phải in các ký tự đọc được, các mã ký tự không đọc được không cần phải in ra).

Sau khi in tất cả các ký tự có mã từ 0 đến 255, nhóm nhận thấy rằng các ký tự đọc được có mã từ 33 đến 126, vì thế nhóm đã dùng hàm **PrintChar()** để in chúng.

12. Chương trình: sort

Viết chương trình **sort** với yêu cầu sau: Cho phép người dùng nhập vào một mảng n số nguyên với n là số do người dùng nhập vào ($n \leq 100$).

Sử dụng thuật toán bubble sort để sắp xếp mảng trên theo chiều tăng dần hoặc giảm dần tùy vào người dùng lựa chọn.

Chương trình này cài đặt rất dễ dàng, bao gồm các bước sau đây:

- Khai báo mảng, biến kích thước mảng n, biến chạy, biến tạm.
- Nhập n.
- Nếu n không hợp lệ ($n < 0$ hoặc $n > 100$), yêu cầu người dùng nhập lại.
- Nhập mảng.
- Sắp xếp mảng bằng thuật toán Bubble sort.
- In mảng đã sắp xếp.

Nhờ cài đặt các system call một cách đúng đắn và hợp lý, việc cài đặt các chương trình ở mức người dùng cũng trở nên dễ dàng và đơn giản. Đồng thời, việc xử lý các trường hợp có thể gây sụp hệ điều hành đã khiến chương trình hoạt động ổn định và chính xác.

Tài liệu tham khảo

- [1] How to install NachOS
https://www.fit.hcmus.edu.vn/~ntquan/os/setup_nachos.html
- [2] Project 01 - Exceptions và các system calls đơn giản
Chi tiết hướng dẫn thực hiện đồ án 01 của cô Chung Thùy Linh
- [3] Giao tiếp giữa hệ điều hành NachOS và chương trình
Tài liệu hướng dẫn thực hiện đồ án 01
- [4] Cách viết một system call
Tài liệu hướng dẫn thực hiện đồ án 01



Cảm ơn thầy cô đã đọc tài liệu này.