

TRƯỜNG ĐẠI HỌC KHOA HỌC TỰ NHIÊN, ĐHQG - HCM  
KHOA CÔNG NGHỆ THÔNG TIN



# BÁO CÁO PROJECT 2

## TÌM HIỂU VÀ CÀI ĐẶT SYSCALL CỦA NACHOS

**Học phần:** Hệ điều hành

**Lớp:** 19\_21

**Giáo viên hướng dẫn:** Lê Viết Long

**Sinh viên thực hiện:**

- Lê Kiệt – 19120554
- Hồ Hữu Ngọc – 19120602
- Nguyễn Phát Minh – 19120586
- Hoàng Mạnh Khiêm – 19120543

# MỤC LỤC

I.	MÔ TẢ THƯ MỤC VÀ TẬP TIN CỦA NACHOS .....	1
a)	Thư mục nachos/gnu-decstation-ultrix .....	1
b)	Thư mục nachos/nachos-3.4/code/test .....	1
c)	Thư mục nachos-3.4/c++example .....	2
d)	Thư mục nachos-3.4/code/network .....	2
e)	Thư mục nachos-3.4/code/machine .....	2
f)	Thư mục nachos-3.4/code/userprog .....	5
g)	Thư mục nachos-3.4/code/bin .....	6
h)	Thư mục nachos-3.4/code/filesys .....	6
g)	Thư mục nachos-3.4/code/threads .....	7
II.	MÔ TẢ THIẾT KẾ VÀ CÁCH THỨC HOẠT ĐỘNG CỦA NACHOS [MINH] .....	9
1.	THIẾT KẾ .....	9
2.	QUÁ TRÌNH BIÊN DỊCH SYSTEM CALL .....	11
III.	EXCEPTIONS VÀ SYSTEM CALLS [KHIÊM + KIẾT] .....	12
1.	CÀI ĐẶT TRƯỚC .....	12
a)	Khai báo biến toàn cục gSynchConsole .....	12
b)	Các hàm System2User() và User2System() trong file exception.cc .....	12
c)	Hàm IncreasePC() trong exception.cc .....	12
d)	Khai báo các syscall .....	12
2.	CÀI ĐẶT SYSCALL .....	13
a)	Syscall ReadInt: int ReadInt() .....	13
b)	Syscall PrintInt: void PrintInt(int number) .....	14
c)	Syscall ReadChar: char ReadChar() .....	14
d)	Syscall PrintChar: void PrintChar(char character) .....	15
e)	Syscall ReadString: void ReadString (char[] buffer, int length) .....	15
f)	Syscall PrintString: void PrintString (char[] buffer) .....	15
3.	XỬ LÝ EXCEPTION .....	15
IV.	DEMO .....	15
1.	Syscall ReadInt và PrintInt .....	16
2.	Syscall ReadChar và PrintChar .....	17
3.	Syscall ReadString và PrintString .....	19
4.	Chương trình help .....	19
5.	Chương trình ascii .....	19
6.	Chương trình sort .....	20

## I. MÔ TẢ THƯ MỤC VÀ TẬP TIN CỦA NACHOS

- Nachos là một phần mềm mã nguồn mở (open-source) giả lập một máy tính ảo và một số thành phần cơ bản của hệ điều hành chạy trên máy tính ảo này nhằm giúp cho việc tìm hiểu và xây dựng các thành phần phức tạp hơn của hệ điều hành
  - o Máy ảo được giả lập có kiến trúc MIPS với hầu hết các thành phần và chức năng của một máy thật như: thanh ghi, bộ nhớ, bộ xử lý, bộ lệnh, chu kỳ thực thi lệnh, cơ chế ngắt, chu kỳ đồng hồ, ...
  - o Hệ điều hành Nachos chạy trên máy ảo Nachos hiện là một hệ điều hành đơn chương.
- Nachos là phần mềm giảng dạy cho phép sinh viên nghiên cứu và sửa đổi một hệ điều hành thực. Sự khác biệt duy nhất giữa Nachos và hệ điều hành "thực" là Nachos chạy như một quy trình Unix duy nhất, trong khi hệ điều hành thực chạy trên máy thật. Tuy nhiên, Nachos mô phỏng các cơ sở cấp thấp chung của các máy điển hình, bao gồm ngắt, bộ nhớ ảo và I / O thiết bị điều khiển ngắt.

### a) Thư mục nachos/gnu-decstation-ultrix

- Là thư mục chứa Cross-compiler được dùng để biên dịch các chương trình C thành các chương trình thực thi trên hệ điều hành Linux cho kiến trúc máy MIPS (COFF).
- Một cross-compiler ("Trình biên dịch chéo") là một trình biên dịch có khả năng tạo ra thực thi mã cho một nền tảng hệ điều hành khác so với cái mà trình biên dịch đang chạy. Ví dụ: trong đồ án này trình biên dịch chạy trên linux nhưng tạo mã chạy trên nachos là trình biên dịch chéo.
- Một trình biên dịch chéo là cần thiết để biên dịch mã cho nhiều nền tảng từ một máy chủ phát triển. Việc biên dịch trực tiếp trên nền tảng đích có thể không khả thi, ví dụ như trên các hệ thống nhúng có tài nguyên máy tính hạn chế.
- Trình biên dịch chéo khác với source-to-source compiler ("trình biên dịch nguồn sang nguồn"). Một trình biên dịch chéo dành cho việc tạo phần mềm đa nền tảng của mã máy, trong khi trình biên dịch nguồn sang nguồn dịch từ ngôn ngữ lập trình này sang ngôn ngữ lập trình khác trong mã văn bản. Cả hai đều là công cụ lập trình.

### b) Thư mục nachos/nachos-3.4/code/test

- Thư mục này sẽ chứa các chương trình viết bằng ngôn ngữ C và được biên dịch theo MIPS và chạy trong hệ điều hành nachos. Các chương trình thực thi này chạy trên một máy ảo mô phỏng SPIM được liên kết với các tệp thực thi trong nachos. Cách hoạt động của các chương trình sẽ được mô tả kỹ hơn ở những phần sau.
  - **File halt.c:** Chứa một chương trình C đơn giản để kiểm tra xem việc chạy chương trình người dùng có hoạt động hay không. Lệnh gọi hàm halt này thường được dùng trong chương trình do người dùng viết. Chỉ cần gọi một "system calls" này để tắt hệ điều hành.
  - **File start.c:** Là một chương trình hợp ngữ hỗ trợ chương trình người dùng chạy trên Nachos. Hợp ngữ này hỗ trợ system call đến nachos kernel.

- **File sort.c:** Chứa chương trình C đơn giản để sắp xếp một số lượng lớn các số nguyên. Mục đích là thử nghiệm hệ thống bộ nhớ ảo. Chương trình có thể đọc mảng chưa được sắp xếp từ một hệ thống tệp, và lưu trữ kết quả vào một tập tin hệ thống mong muốn
- **File matmult.c:** Chương trình C đơn giản để nhân hai ma trận. Mục đích là thử nghiệm hệ thống bộ nhớ ảo. Chương trình có thể đọc các ma trận từ một hệ thống tệp, và lưu trữ kết quả nhân hai ma trận vào một tập tin hệ thống mong muốn.
- Ngoài ra có các file \*.o là những file được tạo ra khi thực thi các file \*.c có tác dụng để chạy chương trình C trên nachos.

#### c) Thư mục nachos-3.4/c++example

- Thư mục này chứa một số file cấu trúc định nghĩa các lớp và hàm như list, stack,.....

#### d) Thư mục nachos-3.4/code/network

- Chứa các file liên quan đến việc truyền các gói tin thông qua liên kết mạng. Cung cấp việc phân phối các gói có kích thước giới hạn đến các máy khác trên mạng. Các gói tin cũng được đảm bảo không bị gián đoạn.

#### e) Thư mục nachos-3.4/code/machine

- Nachos mô phỏng một máy gần đúng với kiến trúc MIPS. Máy có thanh ghi, bộ nhớ và một cpu. Ngoài ra, đồng hồ mô phỏng theo hướng sự kiện cung cấp cơ chế lập lịch ngắt và thực thi tác vụ vào thời điểm sau đó.

#### ➤ File Machine.h và Machine.cc

- Hai file này chứa cấu trúc và định nghĩa của các lớp và các hàm và phân định nghĩa nhằm mục đích mô phỏng các thành phần của máy tính khi người dùng thực thi chương trình người dùng : bộ nhớ chính, thanh ghi, v.v..
- Chương trình người dùng được nạp vào “mainMemory”. Đối với nachos, việc này giống như một mảng các bytes. Tất nhiên, nhân của nachos cũng ở trong bộ nhớ. Cũng giống như hầu hết các ảo hiện nay, nhân được nạp vào một vùng bộ nhớ riêng biệt từ chương trình người dùng và truy cập tới bộ nhớ trung tâm không được dịch hoặc phân trang.
- Trong nachos, các chương trình người dùng được thực thi một lệnh tại một thời điểm bởi giả lập. Mỗi bộ nhớ tham chiếu sẽ được dịch, kiểm tra lỗi,...

#### ➤ File mippssim.cc và File mippssim.h

- Hai file này chứa cấu trúc và định nghĩa của các lớp và các hàm để mô phỏng tập lệnh số nguyên của MIPS R2/3000 processor.
- Cho đến nay, tất cả mã đã viết cho Nachos đều là một phần của nhân hệ điều hành. Trong hệ điều hành thực, hạt nhân không chỉ sử dụng các thủ tục bên trong mà còn cho phép các chương trình cấp người dùng truy cập vào một số quy trình của nó thông qua “system calls”. Máy ảo mô phỏng MIPS thì thực sự là thủ tục lớn và là một phần phân tán của Nachos. Quy trình này hiểu định dạng của các lệnh MIPS và hành vi mong đợi của các lệnh đó như được định nghĩa bởi kiến trúc MIPS. Khi trình mô phỏng MIPS đang thực hiện một chương trình người dùng, nó sẽ mô phỏng hành vi của một CPU MIPS thực bằng cách thực hiện một vòng lặp chặt chẽ, tìm nạp các lệnh MIPS từ bộ nhớ máy

được mô phỏng và “thực thi” . Trên thực tế, CPU mô phỏng giống với CPU thực (chip MIPS), nhưng chúng ta không thể chỉ chạy các chương trình người dùng như các quy trình UNIX thông thường, bởi vì chúng ta muốn kiểm soát hoàn toàn số lượng lệnh được thực thi tại một thời điểm, cách địa chỉ không gian. công việc và cách xử lý các ngắt và ngoại lệ (include system calls).

➤ **File console.cc và console.h**

- Hai file này chứa cấu trúc và định nghĩa của các lớp và các hàm nhằm mục đích mô phỏng thiết bị đầu cuối sử dụng UNIX files. Một thiết bị có 3 đặc tính:
  - Đơn vị dữ liệu theo byte
  - Đọc và ghi các bytes cùng một thời điểm
  - Các bytes đến bất đồng bộ
- Hai file này tạo cấu trúc dữ liệu để mô phỏng các hành vi của một thiết bị terminal nhập xuất. Một terminal gồm có hai phần. Một bàn phím nhập(input) và một màn hình hiển thị (output), mỗi phần trong chúng tạo ra hoặc chấp nhận các ký tự một cách tuần tự. Thiết bị phần cứng của bảng điều khiển thì không đồng bộ hóa. Khi một ký tự được ghi vào thiết bị, thì nó sẽ trả về ngay lập tức. Và hành động ngắt được gọi sau đó khi mà quá trình nhập xuất hoàn thành. Đối với mỗi lần đọc, một trình xử lý ngắt được gọi khi một ký tự đến.
- Trong file này có hai hàm chính quan trọng:
  - Getchar() : Đọc một ký tự từ bộ đệm đầu vào(nếu có). Trả về ký tự hoặc EOF nếu không có ký tự nào được lưu vào bộ đệm.
  - Putchar() : Ghi một ký tự vào màn hình mô phỏng, lên lịch ngắt

➤ **File Interrupt.cc và Interrupt.h:** Hai file này chứa cấu trúc và định nghĩa của các hàm cung cấp một quy trình (SetLevel) để bật hoặc tắt. Trên một máy tính thực, cần phải có một cơ chế để trả lại quyền điều khiển từ một chương trình người dùng đang chạy cho hệ điều hành. Tương tự, trong Nachos, cần có một cơ chế để trả lại quyền điều khiển cho hệ điều hành từ mô phỏng máy. Như trên một máy thực, điều này được thực hiện trong Nachos bằng các ngắt và ngoại lệ. Một ngoại lệ xảy ra khi chương trình người dùng đang chạy thực hiện lệnh gọi hệ thống hoặc khi nó cố gắng thực hiện một thao tác không được phép (chẳng hạn như chia cho 0 hoặc truy cập vào một phần bị cấm trong bộ nhớ của máy được mô phỏng). Khi một ngoại lệ xảy ra, mô phỏng máy gọi một hàm xử lý ngoại lệ, là một phần của hệ điều hành. Chức năng xử lý ngoại lệ Nachos có trong file ../exception.cc. Khi trình xử lý ngoại lệ được gọi, hệ điều hành có quyền kiểm soát một lần nữa. Thông thường, công việc của nó là tìm ra lý do tại sao ngoại lệ xảy ra và thực hiện một số hành động để giải quyết nó. Nếu ngoại lệ do lệnh gọi hệ thống gây ra, hệ điều hành sẽ thực hiện dịch vụ do chương trình người dùng yêu cầu, ví dụ: ghi một ký tự vào bảng điều khiển hoặc mở một tệp. Nếu chương trình thực hiện một hoạt động bất hợp pháp, hệ điều hành phải quyết định phải làm gì, ví dụ: nó có thể chỉ chấm dứt một chương trình người dùng cố gắng chia cho 0.

➤ **File translate.cc và translate.h**

- Hai file này chứa cấu trúc và định nghĩa của các lớp và các hàm tạo cấu trúc dữ liệu để quản lý việc chuyển đổi từ trang ảo sang trang vật lý được sử dụng để quản lý bộ nhớ vật lý thay cho các chương trình người dùng. Cấu trúc dữ liệu trong tệp này là "sử dụng kép" chúng phục vụ như một mục nhập bảng trang và một mục nhập trong bộ đệm. Bộ đệm được quản lý bằng phần mềm (TLB). Trong một bảng trang hoặc một TLB. Mỗi mục nhập xác định một ánh xạ từ một trang ảo đến một trang vật lý. Ngoài ra, có một số bit bổ sung để kiểm soát truy cập (hợp lệ) và một số bit cho thông tin sử dụng .
- Nachos hỗ trợ hai loại kiến trúc VM: bảng trang tuyến tính hoặc TLB được quản lý bằng phần mềm. Trong khi cái trước đơn giản hơn để lập trình, cái sau tương ứng chặt chẽ hơn với những gì máy hiện tại hỗ trợ. Nachos hỗ trợ cái này hay cái kia, nhưng không hỗ trợ đồng thời. Với các bảng tuyến tính, MMU chia một địa chỉ ảo thành số trang và các thành phần bù trang. Số trang được sử dụng để lập chỉ mục vào một mảng các mục trong bảng trang. Địa chỉ vật lý thực là sự ghép nối của số khung trang trong mục nhập bảng trang và phần bù trang của địa chỉ ảo.
- **File timer.cc và timer.h:** Hai file này chứa cấu trúc và định nghĩa của các lớp và các hàm đếm thời gian phần cứng tạo ra một ngắt CPU sau mỗi X mili giây xác định. Điều này có nghĩa là nó có thể được sử dụng để thực hiện phân chia thời gian hoặc có một luồng chuyển sang chế độ ngủ trong một khoảng thời gian cụ thể. Mô phỏng một bộ đếm thời gian phần cứng bằng cách tạo lịch cho một quá trình ngắt xảy ra.
- **File stats.h và stats.cc:** Hai file này chứa cấu trúc dữ liệu để thu thập thống kê về hiệu suất Nachos.
- **File network.h và network.cc:** Hai file này chứa cấu trúc dữ liệu để mô phỏng một kết nối mạng vật lý. Mạng cung cấp tính trừu tượng của phân phối gói, mỗi gói có kích thước cố định đến các máy khác trên mạng.
- **File disk.cc và disk.h**
  - Hai file này chứa cấu trúc và định nghĩa của các lớp và các hàm tạo cấu trúc dữ liệu để mô phỏng một đĩa vật lý. Đối tượng Disk mô phỏng hành vi của một đĩa thực. Đĩa chỉ có một đĩa duy nhất, với nhiều rãnh chứa các cung riêng lẻ. Mỗi rãnh chứa cùng một số sector và các khối được xác định duy nhất bằng số sector của chúng. Như với một đĩa thực, HĐH bắt đầu các hoạt động để đọc hoặc ghi một sector cụ thể, và một ngắt sau đó cho biết khi nào hoạt động đã thực sự hoàn thành. Disk Nachos chỉ cho phép một hoạt động đang chờ xử lý tại một thời điểm; hệ điều hành chỉ có thể bắt đầu các hoạt động mới khi thiết bị không hoạt động. Lưu ý hệ điều hành có trách nhiệm đảm bảo rằng các yêu cầu mới không được đưa ra trong khi đĩa đang bận phục vụ một yêu cầu trước đó.
  - Để mô phỏng sự chậm trễ điển hình trong việc truy cập đĩa, đối tượng Nachos Disk sẽ thay đổi động thời gian giữa thời điểm bắt đầu hoạt động I / O và ngắt hoàn thành I / O tương ứng của nó. Độ trễ thực tế phụ thuộc vào thời gian di chuyển đầu đĩa từ vị trí trước đó của nó sang rãnh mới, cũng như độ trễ quay gặp phải khi chờ khối mong muốn quay bên dưới đầu đọc / ghi. Nội dung trên đĩa được giữ nguyên khi máy gặp sự cố, nhưng nếu hoạt động của hệ thống tệp (ví dụ: tạo tệp) đang diễn ra khi hệ thống tắt, hệ thống tệp

có thể bị hỏng. Đầu mô phỏng chứa Num Tracks (32) rãnh, mỗi rãnh chứa SectorsPerTrack (32). Các sector riêng lẻ có kích thước SectorSize (128) byte. Ngoài ra, Disk còn chứa một bộ đệm "track buffer". Ngay sau khi tìm kiếm một track mới, đĩa bắt đầu đọc các sector, đặt chúng vào bộ đệm. Bằng cách đó, một yêu cầu đọc tiếp theo có thể tìm thấy dữ liệu đã có trong bộ nhớ bộ nhớ đệm làm giảm độ trễ truy cập.

- **File sysdep.h và sysdep.cc:** Hai file này cung cấp một giao diện hệ thống để hệ điều hành nachos có thể sử dụng thay vì gọi trực tiếp các hàm của thư viện UNIX, để đơn giản hóa việc chuyển giữa các phiên bản UNIX và thậm chí các hệ thống khác, chẳng hạn như MSDOS và Macintosh.
- **File synchcons.h và synchcons.cc:** Hai file này định nghĩa nhóm hàm cho việc quản lý nhập xuất I/O trên console theo dòng trong Nachos. Do nachos là hệ điều hành trên máy ảo của linux nên cần sự hỗ trợ này để tương tác được với terminal của linux.

#### **f) Thư mục nachos-3.4/code/userprog**

- Là thư mục chứa các chương trình C do người dùng viết. Các chương trình này cung cấp các lớp cho phép Nachos tải và thực thi các chương trình người dùng đơn luồng trong các không gian địa chỉ riêng biệt.
- **File addrspace.h và addrspace.c:** Hai file này chứa cấu trúc và định nghĩa của các hàm tạo cấu trúc dữ liệu để theo dõi việc thực thi các chương trình người dùng. CPU cấp người dùng một không gian địa chỉ được lưu và khôi phục trong luồng.
- **File bitmap.h và bitmap.cc:** Hai file này chứa các cấu trúc dữ liệu xác định một bitmap - một mảng các bit mà mỗi bit trong số đó có thể bật hoặc tắt. Được biểu diễn dưới dạng một mảng các số nguyên không dấu, trên đó chúng ta thực hiện số học module để tìm bit mà chúng ta quan tâm. Bitmap có thể được tham số hóa với số lượng bit đã được quản lý.
- **File exception.cc** ( là nơi cài đặt để xử lý các syscall nằm trong /code/userprog/ )
  - Xử lý ngoại lệ khi mà chương trình do người dùng viết mà người dùng thực hiện điều gì đó mà CPU không thể xử lý. Ví dụ: truy cập bộ nhớ không tồn tại, lỗi số học,
  - Hiện tại, chỉ xử lý bằng cách gọi lệnh hệ thống Halt () để ngắt chương trình. Nếu một tình huống người dùng đang thực thi yêu cầu một system call ( bằng cách thực hiện một trap instruction) thì máy ảo sẽ chuyển đổi bảng điều khiển cho bộ xử lý trung tâm bằng cách gọi hàm Exceptionhandler bên trong file exception.cc
- **File proptest.cc:** Chứa các quy trình để kiểm tra xem rằng Nachos có thể tải và thực thi một chương trình người hay không. Ngoài ra, còn có các quy trình để kiểm tra thiết bị phần cứng Console.
- **File syscall.h** (nơi khai báo system call cho người dùng nằm trong /code/userprog/ )
  - Chứa các thủ tục ở kernel mà chương trình người dùng có thể gọi. File này kết nối các chương trình người dùng và kernel Nachos.
  - Ý nghĩa : Lệnh này để gọi xử lý system call. Khi nạp từng lệnh, máy ảo giải thích mã lệnh này, nhận opcode của lệnh là OP\_SYSCALL nó sẽ gọi hàm Raise Exception. Trong hàm này, đơn giản 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 Exceptionhandle



chỉ có xử lý mỗi trường hợp cho system call Halt – tắt máy mà thôi, còn các system call khác thì chưa có.

Ví dụ : Khi được cung cấp hàm Halt(), người lập trình phần mềm không cần quan tâm bên trong có gì mà chỉ cần gọi hàm Halt() để tắt máy.

- **File Makefile, Makefile.common và Makefile.dep:** Makefile này được sử dụng để xây dựng hệ thống Nachos, bao gồm:
  - Mô phỏng máy MIPS và một hệ điều hành đơn giản. Có một Makefile riêng, trong thư mục ".../test" được sử dụng để xây dựng các chương trình thử nghiệm Nachos (chạy trên máy mô phỏng). Có một số thư mục "xây dựng", mỗi thư mục cho mỗi loại.
  - Chức năng của từng file :
  - Makefile chứa lệnh build Nachos để tạo ra các file thực thi cho người dùng nằm trong /code/ (Sửa gmake thành make) (Linux related)
  - Makefile.common thêm class (sẽ tạo ra file thực thi của lớp trong /code/userprog/) nằm trong /code/ (Bỏ class.h và class.cc vào /code/threads/ và build lại)

#### g) Thư mục nachos-3.4/code/bin

- Thư mục này chứa các chương trình hỗ trợ chuyển đổi định dạng giữa các file để nachos có thể thực hiện việc chạy chương trình ở bên trong file đó. Trong Unix, các tệp "a.out" được lưu trữ ở định dạng "coff". Nachos yêu cầu các tệp thực thi phải ở định dạng "Noff" đơn giản hơn. Để chuyển đổi các tệp nhị phân của một định dạng này sang định dạng khác, cần sử dụng chương trình coff2noff.
- **File coff2noff.c:** File này chứa chương trình đọc tệp định dạng COFF và xuất ra tệp định dạng NOFF. Định dạng NOFF về cơ bản chỉ là một phiên bản đơn giản hơn của tệp COFF, ghi lại vị trí của mỗi phân đoạn trong tệp NOFF và vị trí của nó đi trong không gian địa chỉ ảo.
- **File coff2flat.c:** Chứa chương trình đọc một tệp định dạng COFF và xuất ra một tệp phẳng, tệp phẳng sau đó có thể được sao chép trực tiếp vào bộ nhớ ảo và thực thi. Nói cách khác, các đoạn mã đối tượng khác nhau được tải ở độ lệch thích hợp trong tệp phẳng.
- **File coff.c:** Chứa cấu trúc dữ liệu mô tả định dạng của file MIPS COFF.
- **File encode.h:** Chứa định nghĩa các hằng được sử dụng trong chương trình
- **File noff.h:** Chứa cấu trúc dữ liệu xác định định dạng của các đối tượng bên trong Nachos.

#### h) Thư mục nachos-3.4/code/filesys

- Thư mục này chứa các file, lớp và chương trình do người lập trình hệ điều hành tạo nên.
- **File directory.h và directory.cc:** Chứa cấu trúc dữ liệu để quản lý một thư mục tên tệp giống UNIX. Thư mục là một bảng gồm các cặp: <name, sector> cung cấp tên của từng tệp trong thư mục và nơi để tìm tiêu đề tệp của nó.
- **File filehdr.h và filehdr.cc**
  - Hai file này chứa cấu trúc dữ liệu để quản lý tiêu đề của tệp đĩa. Tiêu đề tệp mô tả vị trí trên đĩa để tìm dữ liệu trong tệp, cùng với các thông tin khác về tệp (ví dụ: chiều dài, chủ sở hữu, v.v.)



- Mỗi tệp Nachos có cấu trúc FileHeader được liên kết. FileHeader tương tự như Unix inode ở chỗ nó chứa tất cả thông tin cần thiết về một tệp, chẳng hạn như kích thước hiện tại của tệp và các con trỏ đến các khối đĩa vật lý của nó. Cụ thể, FileHeader của Nachos chứa kích thước hiện tại của tệp tính bằng byte, số lượng khu vực đã được cấp cho tệp và một dãy số khu vực xác định cụ thể nơi chứa các khối dữ liệu của tệp. Khi một tệp ban đầu được tạo, người gọi chỉ định kích thước của tệp. Tại thời điểm tạo tệp, Nachos phân bổ đủ các sector để phù hợp với kích thước được yêu cầu. Do đó, khi dữ liệu được nối vào tệp, không cần phân bổ các địa chỉ khác. Trường kích thước hiện tại cho biết lượng tệp hiện đang chứa dữ liệu có ý nghĩa gì.
- **File filesys.h và filesys.cc:** Hai tập tin này tập hợp các file được lưu trữ trên đĩa, được tổ chức thành các thư mục. Các thao tác trên file hệ thống phải có cú pháp “tên lệnh”—tạo, mở, xóa, được đặt tên file văn bản. Các thao tác trên một tập tin như (read, write, close) đều nằm ở lớp openfile.
- **File fstest.cc:** Chứa các quy trình kiểm tra đơn giản cho hệ thống tệp. Trong đó gồm : Sao chép tệp từ UNIX sang Nachos, đọc và ghi nội dung của tệp Nachos.
- **File openfile.h và openfile.cc:** Hai file này định nghĩa các hàm trong hệ thống file của nachos. Trong đồ án này sử dụng lời gọi thao tác với file trực tiếp từ linux. Nhưng trong hệ điều hành UNIX, một tập tin phải được mở trước khi thực hiện các hành động khác như đọc và viết. Một khi đã hoàn thành, có thể đóng tập tin. Trong nachos việc đọc tập tin thực hiện bằng cách xóa cấu trúc dữ liệu
- **File synchdisk.h và synchdisk.cc**
  - Hai file này chứa cấu trúc và định nghĩa của các hàm tạo cấu trúc dữ liệu để xuất giao diện đồng bộ sang thiết bị đĩa thô.
  - Đối tượng SynchDisk nằm phía trên đĩa thô, cung cấp giao diện rõ ràng cho các tác vụ của nó. Cụ thể, nó cung cấp các hoạt động chặn luồng gọi cho đến khi lệnh ngắt được hoàn thành I / O tương ứng diễn ra. Ngược lại, đối tượng Disk cung cấp cơ chế bắt đầu hoạt động I / O, nhưng không cung cấp một cách thuận tiện để chặn người gọi khi yêu cầu hoàn tất. Ngoài ra, SynchDisk cung cấp tính năng loại trừ lẫn nhau, để nhiều luồng có thể gọi đồng thời các quy trình SynchDisk một cách an toàn( Đĩa không thể phục vụ nhiều hơn một yêu cầu cùng một lúc). Thay vì sử dụng trực tiếp đối tượng Disk, hầu hết các ứng dụng sẽ thấy thích hợp khi sử dụng.

### i) **Thư mục nachos-3.4/code/threads**

- Thư mục này chứa các xử lý về tiến trình của nachos
- Một tiến trình bao gồm một chương trình, dữ liệu của nó và tất cả thông tin trạng thái (bộ nhớ, thanh ghi, bộ đếm chương trình, tệp đang mở, v.v.) được liên kết với nó.
- Đôi nhiều tiến trình điều khiển thực thi đồng thời trong một quy trình duy nhất. Các tiến trình điều khiển riêng lẻ này được gọi là luồng. Theo mặc định, các quy trình chỉ có một luồng duy nhất được liên kết với chúng, mặc dù có thể có tác dụng lớn hơn nếu có một số luồng đồng thời được thực hiện. Tất cả các luồng của một quy trình cụ thể chia sẻ cùng một không gian địa chỉ.

- Một sự khác biệt lớn giữa luồng và quy trình là các biến toàn cục được chia sẻ giữa tất cả các luồng. Bởi vì các luồng thực thi đồng thời với các luồng khác, chúng phải lo lắng về việc đồng bộ hóa và loại trừ lẫn nhau khi truy cập bộ nhớ dùng chung.
- Các luồng Nachos thực thi và chia sẻ cùng một mã (mã nguồn Nachos) và chia sẻ các biến toàn cục giống nhau.
- Bộ lập lịch Nachos duy trì một cấu trúc dữ liệu được gọi là danh sách sẵn sàng, theo dõi các luồng đã sẵn sàng thực thi. Các luồng trong danh sách sẵn sàng đã sẵn sàng để thực thi và có thể được chọn để thực thi bởi bộ lập lịch bất kỳ lúc nào.
- **File bool.h:** Chứa chương trình định nghĩa hàm bool( True là 1 và False là 0)
- **File list.h:** Chứa cấu trúc dữ liệu để quản lý danh sách giống LISP. Như trong LISP, một danh sách có thể chứa bất kỳ kiểu cấu trúc dữ liệu nào dưới dạng một mục trong danh sách: khối điều khiển luồng, các ngắt đang chờ xử lý, v.v.
- **File main.cc:** Chứa mã Bootstrap để khởi tạo kernel system. Cho phép gọi trực tiếp vào các chức năng của hệ điều hành nội bộ, để đơn giản hóa việc gỡ lỗi và kiểm tra. Trong thực tế, mã bootstrap sẽ chỉ khởi tạo cấu trúc dữ liệu, và khởi động chương trình người dùng để in lời nhắc cần thiết.
- **File scheduler.h:** Chứa cấu trúc dữ liệu cho bộ điều phối luồng và bộ lập lịch chứa danh sách các luồng đã sẵn sàng chạy.
- **File synch.h và synch.cc:** Hai file này chứa cấu trúc và định nghĩa của các lớp và các hàm tạo cấu trúc dữ liệu để đồng bộ hóa các luồng. Ba loại đồng bộ hóa được định nghĩa ở đây: semaphores, khóa và các biến điều kiện. Việc thực hiện cho semaphores được đưa ra; đối với hai phần sau, chỉ thủ tục giao diện được cung cấp - chúng sẽ được triển khai như một phần của nhiệm vụ đầu tiên. Tất cả các đối tượng đồng bộ hóa có "tên" là một phần của quá trình khởi tạo. Điều này chỉ dành cho mục đích gỡ lỗi.
- **File synchlist.h và synchlist.cc:** Chứa cấu trúc dữ liệu để truy cập đồng bộ dạng danh sách. Được triển khai bằng cách bao quanh phần trừu tượng hóa của danh sách với các quy trình đồng bộ hóa.
- **File system.h và system.cc:** Chứa tất cả các biến toàn cục được sử dụng trong Nachos đều được định nghĩa tại đây. Quản lý thư viện hệ thống nằm trong /code/threads/ có cả lớp synconsole cần được thêm để dùng.
- **File thread.h và thread.cc:** Hai file này chứa cấu trúc và định nghĩa của các lớp và các hàm tạo cấu trúc dữ liệu để quản lý luồng. Một luồng đại diện cho việc thực thi tuần tự mã trong một chương trình. Vì vậy, trạng thái của một luồng bao gồm bộ đếm chương trình, các thanh ghi bộ xử lý và ngăn xếp thực thi. Một ngăn xếp có kích thước cố định cho mỗi luồng, nên có thể làm tràn ngăn xếp - ví dụ: bằng cách đệ quy đến một mức quá sâu. Lý do phổ biến nhất cho sự cố này là phân bổ cấu trúc dữ liệu lớn trên ngăn xếp. Cho nên cần cẩn thận khi viết chương trình xử lý cho phần này. Luồng đi sẽ được nói rõ hơn trong phần sau.
- **File threadtest.cc:** Tạo ra một testcase để kiểm tra luồng đi bằng cách Tạo hai luồng và chuyển đổi qua lại giữa chúng bằng cách gọi Thread :: Yield, để minh họa hoạt động bên trong của hệ thống luồng.

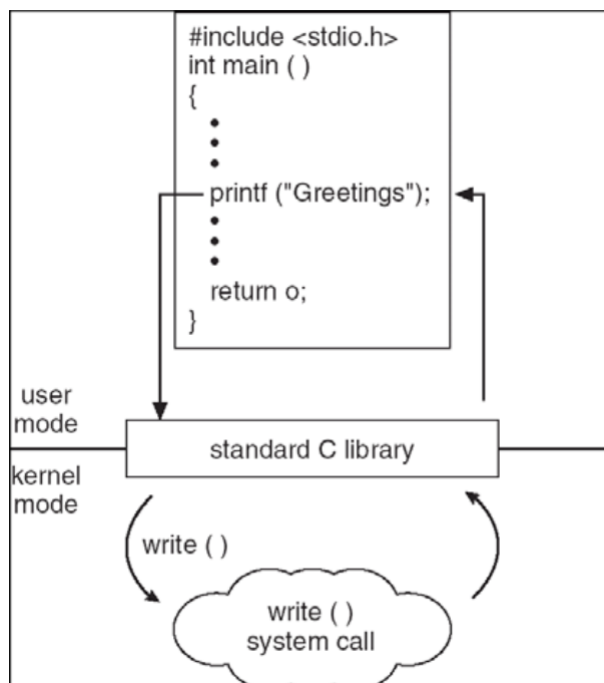
- **File utility.h và utility.cc:** Chứa chương trình thông báo lỗi trong nachos. Các quy trình gỡ lỗi cho phép người dùng bật đã chọn thông báo gỡ lỗi, có thể điều khiển từ các đối số dòng lệnh được chuyển tới Nachos (-d). Các cờ gỡ lỗi được xác định trước là:
- '+' - bật tất cả thông báo gỡ lỗi
  - 't' - hệ thống luồng
  - 's' - semaphores, lock và condition
  - 'i' - giả lập ngắt
  - 'm' - giả lập máy (USER\_PROGRAM)
  - 'd' - giả lập đĩa (FILESYS)
  - 'f' - hệ thống tệp (FILESYS)
  - 'a' - khoảng trống địa chỉ (USER\_PROGRAM)
  - 'n' - mô phỏng mạng (NETWORK)

## II. MÔ TẢ THIẾT KẾ VÀ CÁCH THỨC HOẠT ĐỘNG CỦA NACHOS [MINH]

### 1. THIẾT KẾ

- NachOS là 1 hệ thống giả lập chạy trên nền Linux. NachOS giả lập lại bộ lệnh MIPS với hầu hết các thành phần và chức năng của một máy thật. Các giả lập này được viết bằng ngôn ngữ C

**Mô hình:**



- Có thể thấy được 2 mode chính:
  - User mode: vùng nhớ của những chương trình ứng dụng chạy trên NachOS/MIPS
  - Kernel mode: vùng nhớ của hệ điều hành NachOS

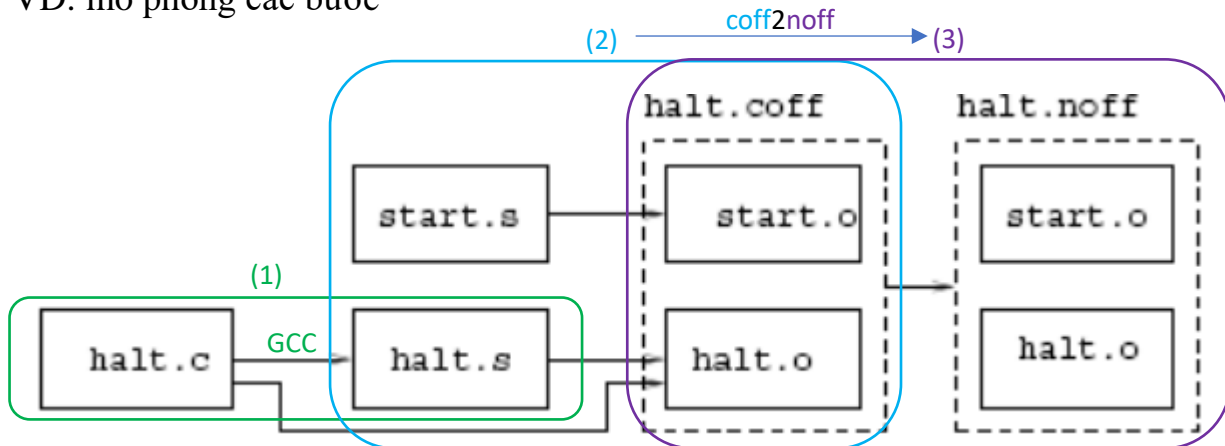
- Cách hệ thống NachOS biên dịch chương trình người dùng:
  - (1) Từ 1 file **<filename>.c** của usermode à trình biên dịch **GCC** (cross-compiler) biên dịch tập tin để tạo tập tin mới có đuôi **<filename>.s** là mã hợp ngữ chạy trên kiến trúc MIPS
  - (2) **<filename>.s** liên kết với tập tin **start.s** (đây được coi là file thư viện) để tạo thành tập tin nhị phân mã máy **<filename>.coff** bao gồm **<filename>.o** và **start.o** là định dạng thực thi trên hệ điều hành **Linux** trên kiến trúc 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
```

- (3) Cuối cùng, để thực thi trên hệ điều hành **NachOS** trên kiến trúc máy MIPS thì **<filename>.coff** phải được chuyển thành **<filename>.noff** nhờ tiện ích “**coff2noff**” trong thư mục **/code/bin**. File **<filename>.noff** cũng sẽ bao gồm **<filename>.o** và **start.o** và sẽ thực thi được trên NachOS

```
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
```

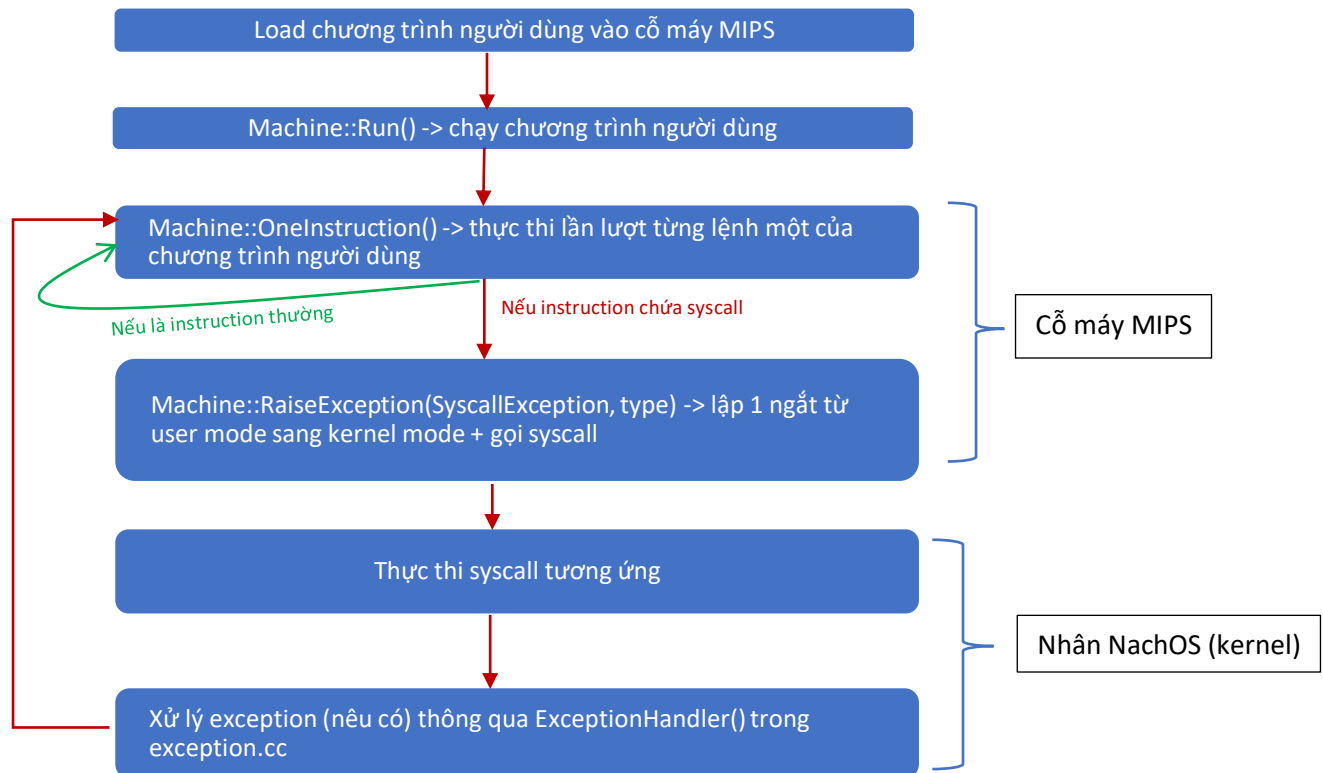
VD: mô phỏng các bước



- User Mode sử dụng bộ nhớ của máy thật và Kernel Mode sử dụng bộ nhớ máy ảo. Dù là bộ nhớ máy ảo nhưng thực chất là vẫn là của máy thật do máy ảo MIPS quản lí.
- Cỗ máy ảo chính là class **Machine** trong tập tin **machine.h** trong đường dẫn **code/machine**. Tại đây, máy ảo được giả lập với các thành phần các thanh ghi (**registers[NumTotalRegs]**), RAM, phương thức đọc, ghi bộ nhớ - thanh ghi (**ReadMem/ WriteMem - ReadRegister/ WriteRegister**), phương thức xử lý từng lệnh theo mã máy MIPS. Trong class **Machine** cũng định nghĩa ra các hàm phục vụ cho quá trình nạp, giải mã và thi hành lệnh và người

dùng không được phép gọi tới các hàm này. Bộ nhớ máy ảo được mô phỏng qua biến **char\* mainMemory**

- Các file nguyên gốc từ có đuôi **.c** sau khi chuyển thành file có đuôi **.noff** sẽ được nạp vào bộ nhớ ảo của MIPS, từ bộ nhớ ảo thì CPU ảo sẽ nạp lên các lệnh để xử lí (đoạn OneInstruction), sau đó sẽ đến phần **system call**
- Các hàm trong lớp Machine không xài riêng lẻ mà sẽ kết hợp với các chương trình người dùng để thực thi syscall mong muốn. Quy trình đó như sau



## 2. QUÁ TRÌNH BIÊN DỊCH SYSTEM CALL

- Muốn thực thi thì ta phải trải qua các bước biên dịch như trên để có được tập tin **.noff**. Sau khi có được **.noff** ta sẽ đi như sau. Ví dụ, để thực thi syscall **halt**:
  - o CPU MIPS sẽ nạp lệnh của **halt.noff** từ bộ nhớ ảo để thực thi. Mà như đã đề cập thì file **halt.noff** bao gồm **halt.o** và **start.o** nên nó sẽ chạy từ trên xuống dưới, nghĩa là phần mã máy của **start** trước (tức **start.s**)
  - o File **start.s** sẽ luôn được chạy trước các chương trình khác, trong **start.s** ta sẽ *jal* đến main của chương trình **halt.s**.
  - o Trong main của **halt.s** *jal* đến nhãn **Halt** (tên hàm cũng giống tên nhãn) tương ứng trong **start.s**.
  - o Tại vị trí nhãn này sẽ gặp lệnh **addiu \$2, \$0, SC\_Halt** để ghi mã của system call **Halt** là **SC\_Halt** (là 1 số nguyên) vào thanh r2 (mã system call phải tự cài đặt lúc ban đầu).

- Gọi xử lý **system call**.
  - Truyền lệnh **system call** cho **ExceptionHandler** để thực hiện chuyển qua **kernel mode**.
  - Khi ở **kernel mode** lấy số **system call** ở thanh ghi mà vừa này lưu để kiểm tra lệnh nào.
  - Nhận được mã **system call** của **Halt** thì thực hiện lệnh và chuyển lại **user mode** để thực hiện phần còn lại.
- Tóm tắt quy trình biên dịch syscall thông qua chương trình người dùng

### III. EXCEPTIONS VÀ SYSTEM CALLS [KHIÊM + KIỆT]

#### 1. CÀI ĐẶT TRƯỚC

##### a) Khai báo biến toàn cục gSynchConsole

- Sau khi thêm lớp SynchConsole như hướng dẫn trong file “[4] Cách Thêm 1 Lớp Vào Nachos.pdf”, tiến hành các bước để thêm 1 biến toàn cục
  - Bước 1: vào **.code/threads/system.h** tại macro **#ifndef USER\_PROGRAM** để khai báo thư viện chứa lớp SynchConsole và khai báo biến gSynchConsole là 1 biến extern. Vì ta thao tác mọi hoạt động liên quan tới lớp SynchConsole thông qua biến này nên biến này sẽ có kiểu là con trỏ trỏ tới lớp SynchConsole
  - Bước 2: vào **.code/threads/system.cc** (đây là file dùng cho việc khởi tạo và giải phóng bộ nhớ các biến con trỏ toàn cục) tại các macro **#ifndef USER\_PROGRAM** để lần lượt khai báo (không cần từ khóa extern), cấp phát và hủy vùng nhớ cho biến gSynchConsole

##### b) Các hàm System2User() và User2System() trong file exception.cc

- Định nghĩa hàm của 2 hàm này được trích trong trang 7 file “[2] Giao tiếp giữa HDH Nachos và chương trình người dùng.pdf”
- System2User(): copy nội dung từ vùng nhớ đệm trong kernel space sang user space
- User2System(): copy nội dung từ vùng nhớ đệm trong user space sang kernel space

##### c) Hàm IncreasePC() trong exception.cc

- Hàm có chức năng tăng program counter. Đầu tiên ghi đè giá trị của thanh PC trước đó (**PrevPCReg**) thành giá trị thanh PC hiện tại (**PCReg**), sau đó gán giá trị thanh ghi hiện tại thành giá trị thanh ghi tiếp theo (**NextPCReg**) mà chương trình sắp thực thi; cuối cùng, ghi đè giá trị thanh ghi tiếp theo là giá trị thanh ghi tiếp sau đó nữa (vì mỗi instruction có kích thước 4 byte và PC luôn trỏ tới byte đầu trong 4 byte đó nên cần tịnh tiến thanh ghi NextPCReg lên 4 để đi tới địa chỉ của instruction tiếp theo sau NextPCReg)

##### d) Khai báo các syscall

- Khai báo các hằng (ví dụ: **#define SC\_ReadInt**) và nguyên mẫu các hàm trong file **.code/userprog/syscall.h**



- Trong file code/test/start.c và code/test/start.s thêm mỗi 1 đoạn chương trình MIPS tương ứng với 1 syscall riêng

## 2. CÀI ĐẶT SYSCALL

- Các định nghĩa của các syscall được đặt trong file exception.cc. trước khi xử lý cho từng syscall, ta khai báo các biến dùng chung:
  - o **maxBuffer** = 255: số lượng ký tự tối đa mà người dùng nhập vào
  - o **char\* buffer[maxBuffer+1]**: lưu nội dung người dùng nhập từ màn terminal, vì nội dung nhập luôn kết thúc bởi ký tự '\0' nên kích thước thật sự của buffer là maxBuffer+1

### a) Syscall ReadInt: int ReadInt()

- Input: không có
- Output: số đã nhập nếu như số nhập vào là 1 số nguyên hợp lệ; ngược lại trả về 0. Ngoài ra có xử lý thêm các tác vụ sau
  - o Nếu số nằm ngoài (-2147483648 , 2147483647], tức tràn số, thì sẽ trả về 0.
  - o Nếu là số chấm động và sau dấu chấm là 1 dãy số 0 (Ví dụ: 12.000) thì trả về số phần nguyên của dấu chấm động (ví dụ: 12). Ngược lại, nếu là số thực bình thường (ví dụ: 12.001) thì sẽ trả về 0 báo hiệu đây không là số nguyên hợp lệ
- Đầu tiên, đọc số nhập từ màn hình thông qua phương thức Read của biến toàn cục gSynchConsole thuộc lớp SynchConsole, lưu số byte đọc được trong biến **nBytes** và nội dung nhập vào **buffer**. Ta cũng khai báo một biến **long long number** để lưu kết quả cuối cùng, biến này có kiểu **long long** để hỗ trợ cho việc tính số lớn và kiểm tra tràn số nếu số ngoài phạm vi lưu trữ của kiểu int
- Tiếp theo kiểm tra số có là số âm bằng việc kiểm tra `buffer[0] == '-'` ? nếu đúng thì bắt đầu duyệt buffer từ chỉ số 1 để lọc ra các chữ số; ngược lại duyệt từ chỉ số 0
- Duyệt từng ký tự trong buffer:
  - o Nếu ký tự đang xét là '.' nghĩa là có 2 trường hợp, trường hợp 1 là số chấm động (VD: 12.001), trường hợp 2 là số nguyên (VD: 12.000). Tiến hành duyệt các ký tự sau dấu '.' để xét có chữ số nào khác 0 hay không, nếu có chữ số như vậy thì rơi vào trường hợp 1 à không phải số nguyên nên trả về 0; ngược lại, kết thúc duyệt buffer vì đã xét tới dấu '.'
  - o Nếu ký tự đang xét không phải ký tự số, tức là nằm ngoài đoạn ASCII ['0', '9'] (trừ ký tự '.' phía trên) thì trả về 0. Ngược lại cộng dồn từng ký tự số vào biến **number**
- Kết thúc duyệt buffer, kiểm tra số là số âm, nếu đúng thì nhân thêm **number** với -1
- Cuối cùng, kiểm tra có tràn số không, tức ngoài (-2147483648 , 2147483647] thì trả về 0; ngược lại trả về kết quả **number**. Giá trị trả về được lưu trong thanh ghi r2 qua phương thức `machineWriteRegister(2, number)`. Ta ghi kết quả trả về vào thanh ghi r2



vì đây là thanh ghi lưu mã syscall, đồng thời lưu kết quả trả về của syscall nếu có. Sau đó tăng program counter và kết thúc syscall

#### **b) Syscall PrintInt: void PrintInt(int number)**

- Input: số nguyên
- Output: không có. In số nguyên truyền vào ra terminal
- Đầu tiên, đọc vào tham số **number** được lưu trong thanh ghi r4 qua việc gọi hàm `machineàReadRegister(4)`. Ta kiểm tra `number == 0` hay không, nếu đúng thì in ra màn hình thông qua `gSynchConsoleWrite("0", 1)` và kết thúc syscall ngay
- Đếm số chữ số:
  - o Trước khi kiểm tra điều này cần kiểm tra `number` có âm hay không vì ta không muốn đếm cả dấu '-', nếu là âm thì tạm thời chuyển về dương để đếm số chữ số. Gán **isNegative** = true để khúc sau chuyển ngược lại về số âm trả ra màn hình và gán **startIndex** = 1 để dành ra `buffer[0]` cho dấu '-'
  - o Dùng biến tạm **tmp** = **number**, chia dần **tmp** cho 10 và cùng lúc đếm **nDigits** thêm 1, làm như vậy tới khi **tmp** = 0 thì dừng đếm. Cũng chính vì vậy mà cần so sánh **number** == 0 ở bước đầu tiên vì nếu không, khi tới bước này **tmp** = 0 à **nDigits** = 0 là sai
- Chia dần `number` cho 10 và gán ngược vào `buffer[i]`, với `startIndex+nDigits-1 ≥ i ≥ startIndex`. Kết thúc duyệt (khi **number** = 0), `buffer` sẽ chứa các ký tự số của **number** ban đầu
- In ra terminal:
  - o Nếu là số âm (**isNegative** == true) thì gán **buffer[0]** = '-' và xuất ra màn hình **nDigits+1** ký tự (+1 do có cả dấu '-') của `buffer` qua việc gọi `gSynchConsoleàWrite(buffer, nDigits + 1);`
  - o Ngược lại, nếu là số dương thì xuất ra màn hình **nDigits** ký tự của `buffer` qua việc gọi `gSynchConsoleàWrite(buffer, nDigits);`
- Tăng program counter và kết thúc syscall

#### **c) Syscall ReadChar: char ReadChar()**

- Input: không có
- Output: ký tự nhập từ màn hình. Nếu nhập vào 1 chuỗi string thì sẽ trả về ký tự đầu chuỗi string đó. Nếu không nhập gì thì in thông báo lỗi ký tự rỗng
- Đầu tiên, đọc input người dùng thông qua phương thức `gSynchConsoleàRead(buffer, maxBuffer)` và lưu số ký tự đọc được vào **nBytes** nội dung đọc sẽ ghi vào **buffer**
- Kiểm tra **nBytes** == 0 ? nếu đúng nghĩa là người dùng không nhập gì cả à xuất thông báo lỗi ký tự rỗng và tăng PC rồi kết thúc syscall

- Ngược lại chỉ lấy và trả về ký tự đầu, tức **buffer[0]**, trong input người dùng thông qua phương thức `machine→WriteRegister(2, buffer[0])` để ghi ký tự này vào thanh ghi r2 rồi tăng program counter, kết thúc syscall

**d) Syscall PrintChar: void PrintChar(char character)**

- Input: 1 ký tự kiểu `char`
- Output: không có. In ký tự truyền vào ra terminal
- Đầu tiên, đọc vị trí bắt đầu của đối số tại thanh ghi r4 qua `machine→ReadRegister(4)` và lưu vào biến `character` kiểu `char`, sau đó in nội dung biến này lên terminal thông qua `gSynchConsole→Write(&character, 1)`. Tăng program counter và kết thúc syscall

**e) Syscall ReadString: void ReadString (char[] buffer, int length)**

- Input: mảng **buffer**, chiều dài tối đa của chuỗi **length**.
- Output: Không có
- Đầu tiên đọc địa chỉ vùng nhớ tại thanh ghi r4 qua **address** = `machine→ReadRegister(4)`, đọc độ dài tối đa của chuỗi tại thanh ghi r5 **length** = `machine→ReadRegister(5)`. Sử dụng hàm `User2System` để sao chép chuỗi từ userspace lên kernelspace. Khi người dùng nhập chuỗi, nội dung sẽ được lưu tại kernel space, đọc chuỗi từ kernel space vào biến **string**. Dùng hàm `System2User` để sao chép chuỗi về userspace. Giải phóng các vùng nhớ phụ được sử dụng, tăng progra counter và kết thúc syscall.

**f) Syscall PrintString: void PrintString (char[] buffer)**

- Input: chuỗi **buffer**.
- Output: Xuất chuỗi **buffer** ra màn hình console.
- Đầu tiên lấy địa chỉ của vùng nhớ **buffer** tại thanh ghi r4 **address** = `machine→ReadRegister(4)`. Sử dụng hàm `User2System` sao chép chuỗi từ userspace qua kernelspace. Tính toán chiều dài thực của chuỗi qua vòng lặp `while` sau đó in chuỗi ra màn hình console `gSynchConsole→Write(string, length + 1)`. Giải phóng các vùng nhớ phụ được sử dụng, tăng progra counter và kết thúc syscall.

### 3. XỬ LÝ EXCEPTION

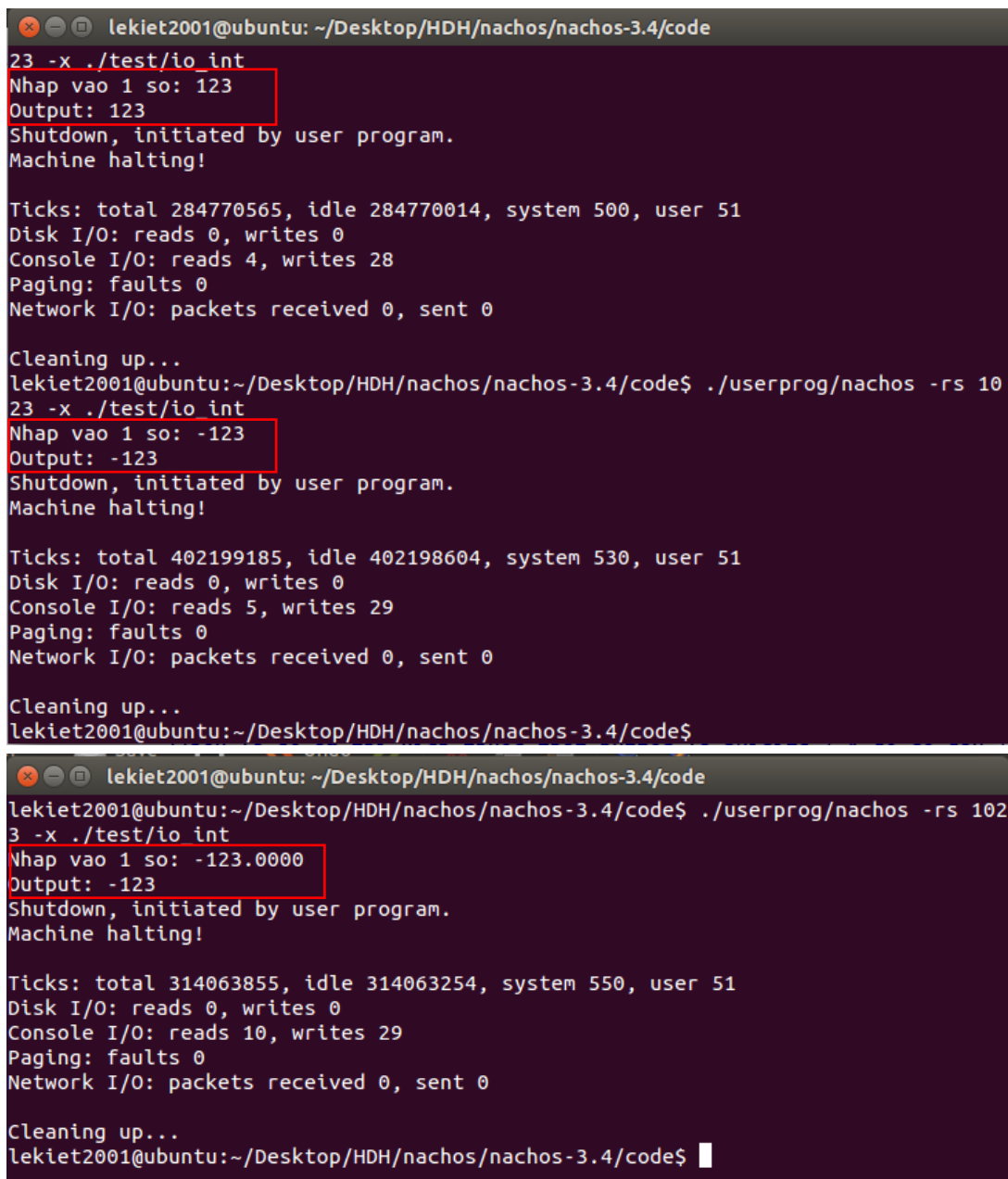
- Với exception **NoException**, chương trình trả quyền điều khiển cho hệ điều hành qua “lệnh ngắt” `interrupt` thuộc class `Interrupt`, “lệnh ngắt” này sẽ thiết lập lại trạng thái của NachOS là `SystemMode` qua hàm `setStatus()` - tức là trả quyền điều khiển về cho kernel
- Với exception **SyscallException**, xử lý hàm syscall tương ứng bằng cách `switch case` cho biến `type`
- Với các exception còn lại sẽ tiến hành in ra 1 thông báo lỗi và halt hệ thống

### IV. DEMO

- Link: <https://youtu.be/RXU5XGhJNs8>

## 1. Syscall ReadInt và PrintInt

- Khi nhập 1 số nguyên hợp lệ (123, -123 và -123.0000)



```
lekiet2001@ubuntu: ~/Desktop/HDH/nachos/nachos-3.4/code
23 -x ./test/io_int
Nhập vào 1 số: 123
Output: 123
Shutdown, initiated by user program.
Machine halting!

Ticks: total 284770565, idle 284770014, system 500, user 51
Disk I/O: reads 0, writes 0
Console I/O: reads 4, writes 28
Paging: faults 0
Network I/O: packets received 0, sent 0

Cleaning up...
lekiet2001@ubuntu:~/Desktop/HDH/nachos/nachos-3.4/code$ ./userprog/nachos -rs 10
23 -x ./test/io_int
Nhập vào 1 số: -123
Output: -123
Shutdown, initiated by user program.
Machine halting!

Ticks: total 402199185, idle 402198604, system 530, user 51
Disk I/O: reads 0, writes 0
Console I/O: reads 5, writes 29
Paging: faults 0
Network I/O: packets received 0, sent 0

Cleaning up...
lekiet2001@ubuntu:~/Desktop/HDH/nachos/nachos-3.4/code$

lekiet2001@ubuntu:~/Desktop/HDH/nachos/nachos-3.4/code$ ./userprog/nachos -rs 102
3 -x ./test/io_int
Nhập vào 1 số: -123.0000
Output: -123
Shutdown, initiated by user program.
Machine halting!

Ticks: total 314063855, idle 314063254, system 550, user 51
Disk I/O: reads 0, writes 0
Console I/O: reads 10, writes 29
Paging: faults 0
Network I/O: packets received 0, sent 0

Cleaning up...
lekiet2001@ubuntu:~/Desktop/HDH/nachos/nachos-3.4/code$
```

- Khi nhập 1 số nguyên không hợp lệ (12a và 123.001)

```
lekiet2001@ubuntu: ~/Desktop/HDH/nachos/nachos-3.4/code
lekiet2001@ubuntu:~/Desktop/HDH/nachos/nachos-3.4/code$ ./userprog/nachos -rs 1
3 -x ./test/io_int
Nhập vào 1 số: 12a
Output: 0
Shutdown, initiated by user program.
Machine halting!

Ticks: total 1345021445, idle 1345020914, system 480, user 51
Disk I/O: reads 0, writes 0
Console I/O: reads 4, writes 26
Paging: faults 0
Network I/O: packets received 0, sent 0

Cleaning up...
lekiet2001@ubuntu:~/Desktop/HDH/nachos/nachos-3.4/code$ ./userprog/nachos -rs 1
3 -x ./test/io_int
Nhập vào 1 số: 123.001
Output: 0
Shutdown, initiated by user program.
Machine halting!

Ticks: total 267597635, idle 267597074, system 510, user 51
Disk I/O: reads 0, writes 0
Console I/O: reads 8, writes 26
Paging: faults 0
Network I/O: packets received 0, sent 0
```

## 2. Syscall ReadChar và PrintChar

- Khi nhập 1 ký tự 'c' và khi nhập 1 chuỗi "he dieu hanh"

```
lekiet2001@ubuntu: ~/Desktop/HDH/nachos/nachos-3.4/code
lekiet2001@ubuntu:~/Desktop/HDH/nachos/nachos-3.4/code$ ./userprog/nachos -rs 1
023 -x ./test/io_char
c
c
Shutdown, initiated by user program.
Machine halting!

Ticks: total 114966703, idle 114966548, system 120, user 35
Disk I/O: reads 0, writes 0
Console I/O: reads 2, writes 1
Paging: faults 0
Network I/O: packets received 0, sent 0

Cleaning up...
lekiet2001@ubuntu:~/Desktop/HDH/nachos/nachos-3.4/code$ ./userprog/nachos -rs 1
023 -x ./test/io_char
he dieu hanh
h
Shutdown, initiated by user program.
Machine halting!

Ticks: total 421810993, idle 421810718, system 240, user 35
Disk I/O: reads 0, writes 0
Console I/O: reads 13, writes 1
Paging: faults 0
Network I/O: packets received 0, sent 0
```

- Khi không nhập gì cả

```
lekiet2001@ubuntu: ~/Desktop/HDH/nachos/nachos-3.4/code
lekiet2001@ubuntu:~/Desktop/HDH/nachos/nachos-3.4/code$ ./userprog/nachos -rs 1
023 -x ./test/io_char
Ky tu rong!
Shutdown, initiated by user program.
Machine halting!

Ticks: total 36923593, idle 36923458, system 100, user 35
Disk I/O: reads 0, writes 0
Console I/O: reads 1, writes 1
Paging: faults 0
Network I/O: packets received 0, sent 0

Cleaning up...
lekiet2001@ubuntu:~/Desktop/HDH/nachos/nachos-3.4/code$
```

### 3. Syscall ReadString và PrintString

```
lekiet2001@ubuntu: ~/Desktop/HDH/nachos/nachos-3.4/code
lekiet2001@ubuntu:~/Desktop/HDH/nachos/nachos-3.4/code$ ./userprog/nachos -rs 1
023 -x ./test/io string
le kiet 19120554
le kiet 19120554
Shutdown, initiated by user program.
Machine halting!

Ticks: total 357133170, idle 357132677, system 460, user 33
Disk I/O: reads 0, writes 0
Console I/O: reads 17, writes 17
Paging: faults 0
Network I/O: packets received 0, sent 0

Cleaning up...
lekiet2001@ubuntu:~/Desktop/HDH/nachos/nachos-3.4/code$ ./userprog/nachos -rs 1
023 -x ./test/io string
9
9
Shutdown, initiated by user program.
Machine halting!

Ticks: total 342974504, idle 342974347, system 120, user 37
Disk I/O: reads 0, writes 0
Console I/O: reads 2, writes 2
Paging: faults 0
Network I/O: packets received 0, sent 0
```

### 4. Chương trình help

- Source code: ../code/test/help.c
- Sử dụng syscall: PrintString

```
hoangmanhkhien@ubuntu: ~/Desktop/nachos/nachos-3.4/code
23 -x ./test/help

1. Giới thiệu về nhóm:
   - Le Kiet (19120554)
   - Ho Huu Ngoc (19120602)
   - Hoang Manh Khien (19120543)
   - Nguyen Phat Minh (19120586)

2. Chương trình sort: dùng thuật toán Bubble Sort để sắp xếp mảng n số nguyên theo thứ tự tăng dần, n (0 <= n <= 100) là số nguyên do người dùng nhập vào

3. Chương trình ascii: In bảng mã ASCII ra màn hình.

Shutdown, initiated by user program.
Machine halting!

Ticks: total 39202, idle 35000, system 4130, user 72
Disk I/O: reads 0, writes 0
Console I/O: reads 0, writes 350
Paging: faults 0
Network I/O: packets received 0, sent 0

Cleaning up...
hoangmanhkhien@ubuntu:~/Desktop/nachos/nachos-3.4/code$ S
```

### 5. Chương trình ascii

- Source code: ../code/test/ascii.c

- Sử dụng syscall: PrintString, PrintInt, PrintChar
- In ra 256 ký tự ASCII. Vì output khá dài nên chương trình này được demo trong video demo đi kèm

## 6. Chương trình sort

- Source code: ../code/test/sort.c
- Sử dụng các syscall: PrintString, PrintInt, ReadInt.
- Chức năng: sắp xếp mảng n phần tử theo thứ tự tăng hoặc giảm dần dựa vào input người dùng. Nếu n nhập vào không hợp lệ (tức ngoài [1,100]) hoặc thứ tự sắp xếp nhập vào không hợp lệ (tức không phải **1: tăng dần** hoặc **2: giảm dần**) thì thoát chương trình. Ngược lại sẽ bắt người dùng nhập mảng n phần tử, nếu trong quá trình nhập người dùng nhập 1 ký tự không phải số thì sẽ coi như nhập số 0. Cuối cùng sắp xếp và in mảng kết quả
- Vì chương trình có nhiều xử lý nên nhiều output khác nhau. Vì vậy sẽ được demo trong video đi kèm