

**TRƯỜNG ĐẠI HỌC KHOA HỌC TỰ NHIÊN
ĐẠI HỌC QUỐC GIA THÀNH PHỐ HỒ CHÍ MINH**



PROJECT 2
HỆ ĐIỀU HÀNH NACHOS

Lớp: 19_1
Môn: Hệ điều hành
GVHD: ThS. Lê Viết Long

Niên khóa: 2021-2022

Mục lục

I.	Thông tin nhóm, phân công công việc và mức độ hoàn thành	3
1.	Thông tin nhóm:.....	3
2.	Bảng phân công công việc:.....	3
3.	Đánh giá mức độ hoàn thành:	5
II.	Nội dung báo cáo:	6
	Phần 1: Hiểu mã chương trình Nachos	6
	Phần 2: Hiểu thiết kế	20
1.	Tổng quan về NachOS	20
2.	Thiết kế:	20
	Phần 3: Exceptions và System calls	24
1.	Cấu hình sơ lược để thực hiện code nachos	24
2.	Viết các Exceptions và system calls.....	24
a.	Viết lại file exception.cc	24
b.	Viết lại cấu trúc điều khiển của chương trình để nhận các Nachos system calls ..	25
c.	Viết mã để tăng giá trị biến program counter	25
d.	Cài đặt system call int ReadInt().....	25
e.	Cài đặt system call int PrintInt(int number)	26
f.	Cài đặt system call char ReadChar()	26
g.	Cài đặt system call void PrintChar(char character)	27
h.	Cài đặt system call void ReadString(char[] buffer, int length).....	28
i.	Cài đặt system call void PrintString(char[] buffer)	28
k.	Cài đặt chương trình ascii	29
l.	Cài đặt chương trình sort	30
III.	Hình ảnh Demo	31
IV.	Tài liệu tham khảo	36
1.	Nguồn tham khảo từ tài liệu của thầy Long:.....	36
2.	Nguồn tham khảo trên Internet:	36

I. Thông tin nhóm, phân công công việc và mức độ hoàn thành

1. Thông tin nhóm:

MSSV	Họ và tên	Email
19120189	Lê Tiến Đạt	19120189@student.hcmus.edu.vn
19120190	Nguyễn Văn Đạt	19120190@student.hcmus.edu.vn
19120399	Nguyễn Tiến Toàn	19120399@student.hcmus.edu.vn
19120492	Đỗ Thái Duy	19120492@student.hcmus.edu.vn
19120603	Nguyễn Bá Ngọc	19120603@student.hcmus.edu.vn

2. Bảng phân công công việc:

MSSV	Học và tên	Nhiệm vụ	Tiến độ
19120189	Lê Tiến Đạt	<p>+ Viết code: Cài đặt system call <code>int ReadInt()</code>, <code>void PrintInt(int number)</code></p> <p>+ Viết báo cáo: Tìm hiểu về các mã nguồn của nachos (Phần 1). Viết cài đặt và thiết kế của 2 system call <code>ReadInt</code>, <code>PrintInt</code>.</p>	100%

19120190	Nguyễn Văn Đạt	<p>+ Viết code: Cài đặt system call <code>char ReadChar()</code>, <code>void PrintChar(char character)</code></p> <p>+ Viết báo cáo: Tìm hiểu về các mã nguồn của nachos (Phần 1). Viết cài đặt và thiết kế của 2 system call <code>ReadChar</code>, <code>PrintChar</code>.</p>	100%
19120399	Nguyễn Tiến Toàn	<p>+ Viết code: Cài đặt system call <code>void ReadString (char[] buffer, int length)</code>, <code>void PrintString (char[] buffer)</code></p> <p>+ Viết báo cáo: Tìm hiểu thiết kế, cách làm việc của hệ điều hành (Phần 2). Viết cài đặt và thiết kế của 2 system call <code>ReadString</code>, <code>PrintString</code>.</p>	100%
19120492	Đỗ Thái Duy	<p>+ Viết code: Viết lại file <code>exception.cc</code>, viết lại cấu trúc để nhận các system calls, viết hàm tăng program counter, format code.</p> <p>+ Viết báo cáo: Tìm hiểu thiết kế, cách làm việc của hệ điều hành (Phần 2). Viết các bước thực hiện của phần viết lại <code>exception.cc</code>, cấu trúc nhận system call, hàm tăng program counter.</p>	100%

19120603	Nguyễn Bá Ngọc	+ Viết code: Viết chương trình help, ascii, sort. + Viết báo cáo: Tìm hiểu về các mã nguồn của nachos (Phần 1). Viết cài đặt và thiết kế của 3 chương trình help, ascii, sort.	100%
----------	----------------	---	------

3. Đánh giá mức độ hoàn thành:

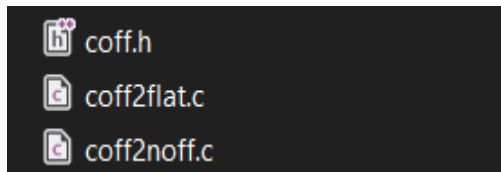
STT	Tên công việc	Mức độ hoàn thành
1	Phần 1: Hiểu mã chương trình Nachos	100%
2	Phần 2: Hiểu thiết kế	100%
3	Viết lại file exception.cc	100%
4	Viết lại cấu trúc điều khiển để nhận các Nachos system call	100%
5	Viết mã để tăng giá trị program counter	100%
6	Cài đặt system call int ReadInt()	100%
7	Cài đặt system call void PrintInt(int number)	100%
8	Cài đặt system call char ReadChar()	100%
9	Cài đặt system call void PrintChar(char character)	100%
10	Cài đặt system call void ReadString(char[] buffer, int length)	100%
11	Cài đặt system call void PrintString(char[] buffer)	100%
12	Viết chương trình help	100%
13	Viết chương trình ascii	100%
14	Viết chương trình sort	100%
	Toàn bộ project	100%

II. Nội dung báo cáo:

Phần 1: Hiểu mã chương trình Nachos

Các file/ thư mục trong chương trình của Nachos gồm:

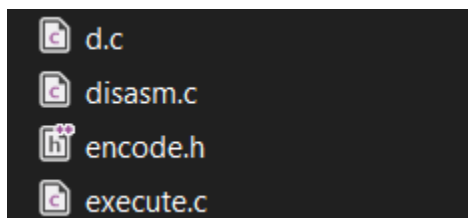
1. bin



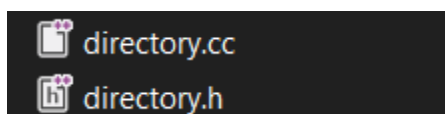
coff.h: cấu trúc dữ liệu mô tả định dạng của MIPS COFF.

coff2flat.c: Chương trình này đọc ở tệp định dạng COFF và xuất ra flat file - flat file sau đó có thể được sao chép trực tiếp vào bộ nhớ ảo và thực thi.

coff2noff.c: Chương trình này đọc ở 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 trong không gian địa chỉ ảo.



2. filesystem

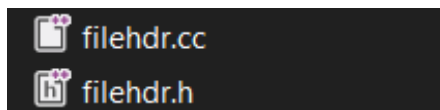


directory.h: Cấu trúc dữ liệu để quản lý một cây thư mục của tên file

Thư mục là một bảng gồm các cặp: <file name, sector #>, cung cấp tên của từng tệp trong thư mục và nơi tìm tiêu đề tệp của nó trên đĩa.

directory.cc: Các quy trình, phương thức quản lý một thư mục tên tệp.

Thư mục là một bảng các entry có độ dài cố định; mỗi entry đại diện cho một tệp duy nhất và chứa tên tệp, và vị trí của tiêu đề tệp trên đĩa. Kích thước cố định của mỗi directory entry có nghĩa là chúng ta có giới hạn có kích thước tối đa cố định cho tên tệp.



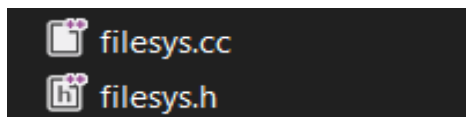
filehdr.h: Cấu trúc dữ liệu để quản lý tiêu đề tệp. Tiêu đề tệp mô tả vị trí trên đĩa để tìm dữ liệu trong tệp, cùng với thông tin khác về tệp (ví dụ: chiều dài, chủ sở hữu, v.v.).

filehdr.cc: Quy trình quản lý tiêu đề tệp đĩa.

Tiêu đề tệp được sử dụng để xác định vị trí trên đĩa nơi data của tệp được lưu trữ.

Tiêu đề tệp có thể được khởi tạo theo hai cách:

- Với một tệp mới, bằng cách sửa đổi cấu trúc dữ liệu trong bộ nhớ để trỏ đến các khối dữ liệu mới được cấp phát.
- Với tệp đã có trên đĩa, bằng cách đọc tiêu đề tệp từ đĩa.



filesys.h: Cấu trúc dữ liệu đại diện cho hệ thống tệp Nachos.

Hệ thống tệp là một tập hợp các tệp được lưu trữ trên đĩa, được tổ chức vào các thư mục. Các hoạt động trên hệ thống tệp phải thực hiện với "naming" - tạo, mở và xóa tệp, cho trước một tên tệp văn bản. Hoạt động trên một tệp "open" (đọc, ghi, đóng) sẽ được tìm thấy trong lớp OpenFile (openfile.h).

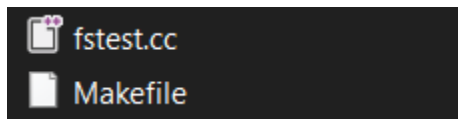
filesys.cc: Các quy trình quản lý hoạt động tổng thể của hệ thống tệp. Thực hiện các thủ tục để ánh xạ từ tên tệp văn bản thành tệp.

Mỗi tệp trong hệ thống tệp có:

- Tiêu đề tệp, được lưu trữ trong một khu vực trên đĩa (kích thước của cấu trúc dữ liệu tiêu đề tệp được sắp xếp có kích thước chính xác bằng 1 khu vực đĩa).
- Một số khối dữ liệu.
- Một entry trong thư mục hệ thống tệp.

Hệ thống tệp bao gồm một số cấu trúc dữ liệu:

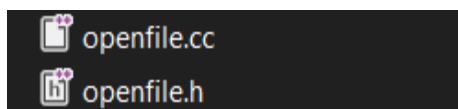
- Một bitmap của các sector đĩa trống.
- Một thư mục tên tệp và tiêu đề tệp.



fstest.cc: Các quy trình kiểm tra đơn giản cho hệ thống tệp.

- Sao chép - sao chép tệp từ UNIX sang Nachos
- In - ghi nội dung của tệp Nachos
- Perftest - một bài kiểm tra căng thẳng cho hệ thống tệp Nachos đọc và ghi một tệp thực sự lớn trong các phần nhỏ (sẽ không hoạt động trên hệ thống cơ sở!).

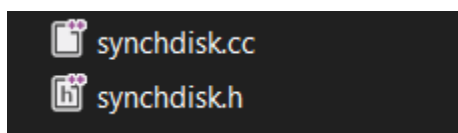
Makefile: dùng để gán hệ thống tệp. Xác định thiết lập giả sử đa chương trình và bộ nhớ ảo được thực hiện trước.



openfile.h: Cấu trúc dữ liệu để mở, đóng, đọc và ghi vào các tệp riêng lẻ.

Có hai cách triển khai. Một là "STUB" trực tiếp biến các thao tác trên tệp thành các thao tác UNIX cơ bản. Hai là triển khai "real", biến những các hoạt động vào đọc và ghi các yêu cầu khu vực đĩa.

openfile.cc: Quy trình quản lý tệp Nachos đang mở. Như trong UNIX, tệp phải được mở trước khi chúng ta có thể đọc hoặc ghi vào nó. Sau khi hoàn tất, chúng ta có thể đóng nó.

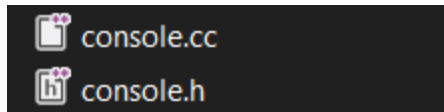


synchdisk.h: Cấu trúc dữ liệu để xuất giao diện đồng bộ sang thiết bị đĩa thô.

synchdisk.cc: Các quy trình truy cập đồng bộ vào đĩa. Đĩa vật lý là một thiết bị không đồng bộ (các yêu cầu đĩa trả về ngay lập tức và ngắt xảy ra sau đó). Đây là một lớp trên cùng của đĩa cung cấp một giao diện đồng bộ (các yêu cầu đợi cho đến khi yêu cầu hoàn thành). Sử dụng một semaphore để đồng bộ hóa trình xử lý ngắt với các yêu cầu đang chờ xử lý. Và, vì đĩa vật lý chỉ có thể xử lý một hoạt động tại một thời điểm, ta sử dụng khóa để thực thi loại trừ lẫn nhau.

3. machine

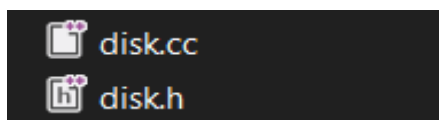
Mô phỏng các thành phần của máy tính khi thực thi chương trình người dùng: bộ nhớ chính, thanh ghi, v.v.



console.h: Chứa cấu trúc dữ liệu để mô phỏng hành vi của thiết bị đầu cuối I/O. Chứa class console.cc: mô tả lại console device, gồm constructor, destructor, hàm và các thuộc tính liên quan.

console.cc: Đặc tả các constructor, destructor và các hàm của class Concole, gồm:

- **GetChar():** Đọc một ký tự từ bộ đệm đầu vào, nếu có. Trả lại ký tự hoặc EOF nếu không có ký tự nào được lưu vào bộ đệm.
- **ChackCharAvail():** Được gọi định kỳ để kiểm tra xem một ký tự có sẵn cho nhập liệu từ bàn phím mô phỏng.
- **WriteDone():** Quy trình nội bộ được gọi khi đến lúc gọi trình xử lý ngắt để thông báo cho hạt nhân Nachos rằng ký tự đầu ra đã hoàn thành.
- **PutChar():** Ghi một ký tự vào màn hình mô phỏng, lên lịch ngắt sẽ xảy ra trong tương lai và trả về.

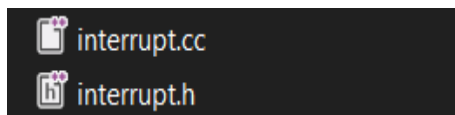


disk.h: Cấu trúc dữ liệu để mô phỏng một đĩa vật lý. Đĩa vật lý có thể chấp nhận (mỗi lần một) yêu cầu đọc, ghi một sector.

disk.cc: Đặc tả constructor, destructor và các phương thức của class disk gồm:

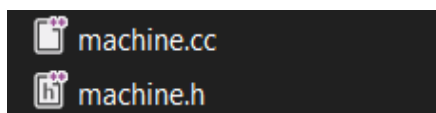
- **PrintSector():** xuất data ổ đĩa.
- **ReadRequest() / WriteRequest():** mô phỏng yêu cầu đọc/ghi một sector.
- **HandleInterrupt():** thông báo cho yêu cầu đĩa đã được thực hiện xong.
- **TimeToSeek():** Trả về thời gian cần thiết để đặt đầu đĩa lên đúng giá trên đĩa.
- **ModuloDiff():** Trả về số lượng các sector có độ trễ luân chuyển giữa sector mục tiêu "to" và vị trí sector hiện tại "from".

- **ComputeLatency()**: trả về thời gian để read/write một disk sector.
- **UpdateLast()**: Giữ track của sector được request gần nhất.



interrupt.h: Cấu trúc dữ liệu để mô phỏng ngắt phần cứng ở low - level.

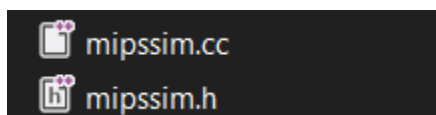
interrupt.cc: Đặc tả constructor, destructor và các thuộc tính và phương thức của class interrupt. Phần cứng cung cấp một quy trình (SetLevel) để bật hoặc tắt các interrupt. Để mô phỏng phần cứng, chúng ta cần theo dõi tất cả các interrupt mà thiết bị phần cứng sẽ gây ra và khi nào chúng được cho là xảy ra. Mô-đun này cũng theo dõi thời gian mô phỏng. Thời gian chỉ tăng lên khi những điều sau xảy ra: interrupts được kích hoạt lại, một lệnh người dùng được thực thi, không có gì trong ready queue.



machine.h: cấu trúc dữ liệu mô phỏng việc thực thi các chương trình người dùng chạy trên Nachos. Gồm 2 class: Instruction-để định nghĩa một lệnh, Machine-định nghĩa mô phỏng nơi vận hành của phần cứng của host, như được thấy bởi các chương trình người dùng - thanh ghi CPU, bộ nhớ chính,...

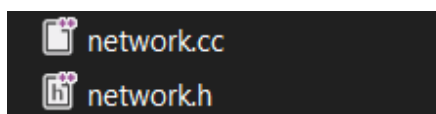
machine.cc: Gồm các quy trình để mô phỏng việc thực thi các chương trình người dùng, gồm:

- **constructor, destructor** và các phương thức của class Machine.
- **checkEndian()**: Kiểm tra rằng máy chủ thực sự sử dụng định dạng mà nó nói, để lưu trữ các byte của một số nguyên. Dừng lại khi có lỗi.



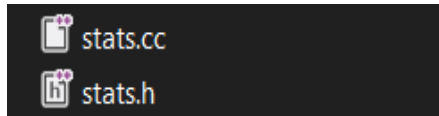
Cấu trúc các dữ liệu, thanh ghi của hệ thống mô tả tập lệnh MIPS.

Mã này đã được điều chỉnh từ Ousterhout's MIPSSIM package.

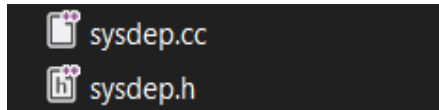


network.h: Cấu trúc dữ liệu để mô phỏng một kết nối mạng vật lý.

network.cc: Quy trình mô phỏng giao diện mạng, sử dụng ổ cắm UNIX để phân phối các gói giữa nhiều lệnh gọi nachos.



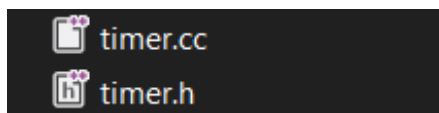
2 file **stats.c** và **stats.h** là cấu trúc dữ liệu để thu thập thông kê về hiệu suất Nachos.



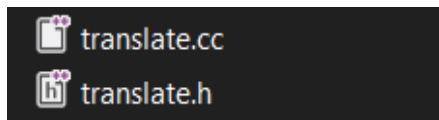
sysdep.h: Giao diện phụ thuộc vào hệ thống. Nachos sử dụng các quy trình được xác định ở đây, thay vì gọi trực tiếp các hàm của thư viện UNIX, để đơn giản hóa việc chuyển giữa các phiên bản của UNIX và thậm chí với các hệ thống khác, chẳng hạn như MSDOS và Macintosh.

sysdep.cc: Thực hiện giao diện phụ thuộc vào hệ thống. Nachos sử dụng các quy trình được định nghĩa ở đây, thay vì gọi trực tiếp thư viện UNIX, để đơn giản hóa việc chuyển giữa các phiên bản của UNIX và thậm chí với các hệ thống khác, chẳng hạn như MSDOS.

Trên UNIX, hầu hết tất cả các quy trình này đều là trình bao bọc đơn giản cho các lệnh gọi hệ thống UNIX cơ bản.



2 file **timer.cc** và **timer.h**: Cấu trúc dữ liệu để mô phỏng bộ đếm thời gian phần cứng.



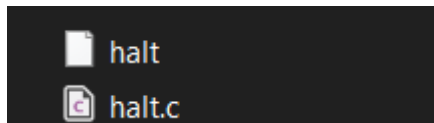
Cấu trúc dữ liệu để quản lý bản dịch từ trang ảo # -> trang vật lý #, được sử dụng để quản lý bộ nhớ vật lý thay mặt cho các chương trình người dùng.

Các cấu trúc dữ liệu trong tệp này là "dual-use" - chúng vừa đóng vai trò là mục nhập bảng trang, vừa là mục nhập trong bộ đệm tìm bản dịch (TLB) do phần mềm quản lý.

Dù bằng cách nào, mỗi mục nhập đều có dạng: <trang ảo #, trang vật lý #>.

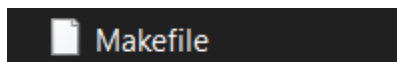
4. test

Thư mục chứa các file code của các chương trình thử xem hệ điều hành hoạt động như thế nào.



halt.c: Chương trình đơn giản để kiểm tra xem việc chạy chương trình của người dùng có chạy đúng hay không. Chương trình chỉ gọi đến 1 syscall để dừng hệ điều hành (ở đây có tên là “halt”).

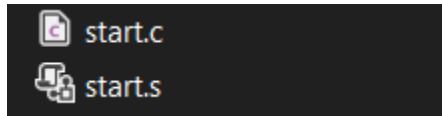
Lưu ý: vì một số lý do, các chương trình người dùng có sử dụng cấu trúc dữ liệu global (vector, matrix,..) đôi khi không hoạt động trong môi trường Nachos. Vì vậy, hãy cẩn thận khi sử dụng. Một lựa chọn là phân bổ cấu trúc dữ liệu như biến tự động trong một quy trình nhưng nếu làm cách này, hãy cẩn thận phân bổ một ngăn xếp đủ lớn để chứa các biến tự động này.



MakeFile: xây dựng các chương trình người dùng tạo chạy được trên Nachos. Một số điều cần lưu ý:

- Nachos giả định rằng vị trí của quy trình khởi động chương trình (vị trí kernel nhảy đến khi chương trình khởi động) là ở vị trí 0. Điều này có nghĩa là: file start.o phải là file .o đầu tiên được truyền qua vào ld, để quy trình "Start" được tải vào vị trí 0 mới có thể bắt đầu chương trình.
- Nếu như sử dụng trình biên dịch chéo, cần trỏ tới đúng file exe và thay đổi flags thành ld và xây dựng chương trình như sau:
 - + GCCDIR = ../../../gnu-decstation-ultrix/decstation-ultrix/2.95.3/
 - + LDFLAGS = -T script -N
 - + ASFLAGS = -mips2
 - + CPPFLAGS = \$(INCDIR)
- Nếu như không sử dụng trình biên dịch thì:
 - + GCCDIR =

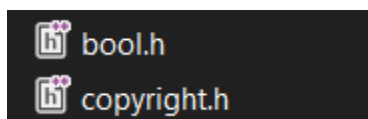
```
+ LDFLAGS = -N -T 0
+ ASFLAGS =
+ CPPFLAGS = -P $(INCDIR)
```



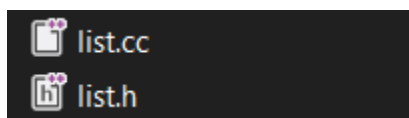
start.c: Chương trình hỗ trợ ngôn ngữ hợp ngữ cho các chương trình người dùng chạy trên Nachos. Vì ta không muốn kéo toàn bộ thư viện C, ta chỉ định nghĩa những gì chúng ta cần cho một chương trình người dùng ở đây, cụ thể là Start và syscall.

start.s: Khởi tạo chạy chương trình C bằng cách gọi "main". Lưu ý: start phải đứng đầu tiên, để nó được tải ở vị trí 0. Để nhân của Nachos luôn bắt đầu một chương trình bằng cách nhảy đến vị trí 0.

5. threads



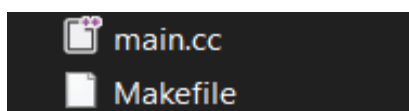
bool.h: define giá trị FALSE là 0 và TRUE là 1.



list.h: Cấu trúc dữ liệu để quản lý danh sách kiểu LIPS, một danh sách có thể chứa bất kỳ kiểu cấu trúc dữ liệu nào dưới dạng một item trong danh sách: khối điều khiển luồng, ngắt đang chờ xử lý, v.v. Đó là lý do tại sao mỗi item là "void *", hay nói cách khác, "trỏ đến bất cứ điều gì".

list.cc: Quy trình quản lý danh sách "things" được liên kết đơn lẻ.

Một "ListElement" được cấp phát cho mỗi item được đưa vào danh sách; nó được khử phân bổ khi item đó bị loại bỏ. Điều này có nghĩa là chúng ta không cần phải giữ một con trỏ "next" trong mọi đối tượng mà chúng ta muốn đưa vào danh sách.



main.cc: Mã Bootstrap để khởi tạo nhân cho hệ điều hành. Cho phép gọi trực tiếp vào các chức năng bên trong của hệ điều hành để làm đơn giản việc tìm lỗi và thử nghiệm. Trong thực tế, mã bootstrap sẽ khởi tạo luôn cấu trúc dữ liệu và khởi động chương trình của người dùng để in lời nhắc đăng nhập.

Tóm tắt main(int argc, char **argv):

Khởi động nhân hệ điều hành.

Kiểm tra đối số dòng lệnh

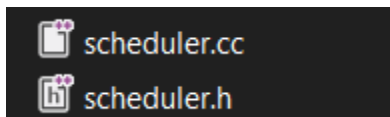
Khởi tạo cấu trúc dữ liệu

(tùy chọn) Gọi quy trình kiểm tra

- "argc" là số đối số dòng lệnh (bao gồm cả tên của lệnh) - ví dụ: "nachos -d +" -> argc = 3.
- "argv" là một mảng các chuỗi, một chuỗi cho mỗi đối số dòng lệnh.
ví dụ: "nachos -d +" -> argv = {"nachos", "-d", "+"}

LƯU Ý: nếu main trả về giá trị thì chương trình "nachos" sẽ thoát (như bao chương trình khác) Nhưng có thể có các luồng khác trong danh sách chờ. Ta chuyển đổi tới các luồng đó bằng cách cho main (luồng hiện tại) đã kết thúc, ngăn chặn main trả về bất cứ giá trị nào.

Makefile: Đây là một GNU Makefile. Nên phải sử dụng "gmake" chứ không phải "make".
Dùng để các phân luồng, cần được thực thi trước tiên.

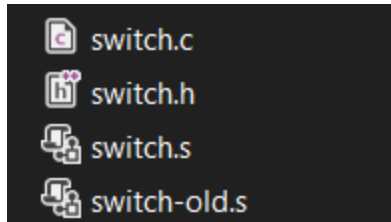


scheduler.h: Cấu trúc dữ liệu cho bộ điều phối luồng và bộ lịch trình. Chủ yếu là danh sách các thread đã sẵn sàng để chạy.

scheduler.cc: Các quy trình để chọn thread tiếp theo để chạy và gửi tới thread đó.

Các quy trình này giả định rằng các ngắt đã bị vô hiệu hóa. Nếu ngắt bị vô hiệu hóa, chúng ta có thể giả định loại trừ lẫn nhau.

LƯU Ý: không thể sử dụng Locks để cung cấp tính năng loại trừ lẫn nhau ở đây, vì nếu chúng ta cần đợi khóa và khóa đang bận, chúng ta sẽ gọi **FindNextToRun()** và điều đó sẽ đưa chúng ta vào một vòng lặp vô hạn.



switch.h: Các định nghĩa cần thiết để thực hiện chuyển đổi các context.

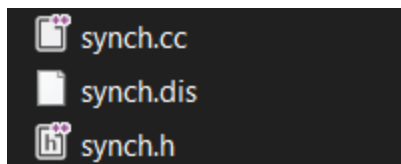
Chuyển đổi context vốn dĩ phụ thuộc vào máy, vì các thanh ghi được lưu, cách thiết lập khung câu lệnh ban đầu, v.v., tất cả đều cụ thể đối với kiến trúc bộ xử lý.

Tập này hiện hỗ trợ kiến trúc DEC MIPS và SUN SPARC.

switch.c: Các quy trình chuyển đổi context phụ thuộc vào máy.

Chương trình xác định hai quy trình cho mỗi kiến trúc:

- **ThreadRoot (InitialPC, InitialArg, WhenDonePC, StartupPC).** ThreadRoot được gọi từ quy trình SWITCH () để bắt đầu một tiểu trình lần đầu tiên.
- **SWITCH (oldThread, newThread):** oldThread - Luồng hiện tại đang chạy, nơi trạng thái thanh ghi CPU. newThread - Luồng mới sẽ được chạy, nơi trạng thái thanh ghi CPU sẽ được tải từ đó.



synch.h: Cấu trúc dữ liệu để đồng bộ hóa các luồng.

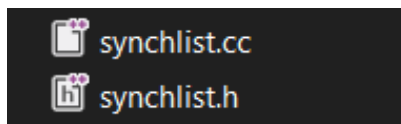
Ba loại đồng bộ hóa được định nghĩa ở đây: semaphores, locks và condition variables.

Việc triển khai cho các semaphores được đưa ra; đối với hai phần sau, chỉ giao diện thủ tục được đưa ra - chúng sẽ được thực hiện như một phần của nhiệm vụ đầu tiên.

Lưu ý rằng tất cả các đối tượng đồng bộ hóa đều có "name" như một phần của quá trình khởi tạo. Điều này chỉ dành cho mục đích gỡ lỗi.

synch.cc: Quy trình đồng bộ hóa các luồng.

Bất kỳ việc thực hiện quy trình đồng bộ hóa nào cũng cần một số hoạt động nguyên tử. Chương trình cho rằng Nachos đang chạy trên bộ xử lý đơn và do đó tính nguyên tử có thể được cung cấp bằng cách tắt ngắt. Trong khi ngắt bị vô hiệu hóa, không có chuyển đổi ngữ cảnh nào có thể xảy ra và do đó luồng hiện tại được đảm bảo giữ CPU xuyên suốt cho đến khi ngắt được kích hoạt lại.



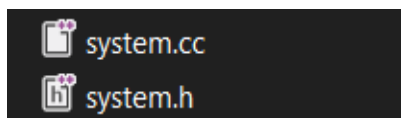
synchlist.h: Cấu trúc dữ liệu để truy cập đồng bộ vào danh sách.

Thực hiện bằng cách bao quanh việc trừu tượng hóa Danh sách với các quy trình đồng bộ hóa.

synchlist.cc: Quy trình truy cập đồng bộ vào danh sách.

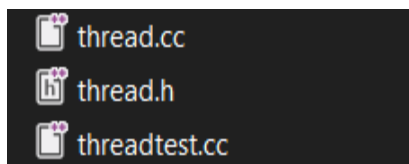
Thực hiện bằng cách bao quanh việc trừu tượng hóa Danh sách với các quy trình đồng bộ hóa.

Được triển khai theo kiểu "monitor" - bao quanh mỗi thủ tục bằng một cặp thu nhận và giải phóng khóa, sử dụng tín hiệu điều kiện và chờ đồng bộ hóa.



system.h: Tất cả các biến toàn cục được sử dụng trong Nachos đều được định nghĩa tại file này.

system.cc: Quy trình khởi tạo và dọn dẹp Nachos.



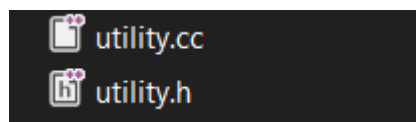
thread.h: Các cấu trúc dữ liệu để quản lý các luồng. Một luồng đại diện cho việc thực thi tuần tự mã trong một chương trình. Vì vậy, trạng thái của một luồng bao gồm bộ đếm chương trình, các thanh ghi bộ xử lý và ngăn xếp thực thi.

thread.cc: Các quy trình để quản lý các luồng. Có bốn hoạt động chính:

- **Fork** - tạo một luồng để chạy một thủ tục đồng thời với trình gọi (điều này được thực hiện trong hai bước - đầu tiên cấp phát đối tượng Luồng, sau đó gọi Fork trên nó)
- **Finish** - được gọi khi quá trình chia tách kết thúc, để dọn dẹp
- **Tield** - từ bỏ quyền kiểm soát đối với CPU cho một chuỗi sẵn sàng khác
- **Sleep** - từ bỏ quyền kiểm soát đối với CPU, nhưng luồng hiện đã bị chặn. Nói cách khác, nó sẽ không chạy lại, cho đến khi được đưa trở lại hàng đợi sẵn sàng một cách rõ ràng.

threadtest.cc: Các trường hợp kiểm tra đơn giản để gán luồng.

Tạo hai luồng và yêu cầu chúng chuyển đổi ngữ cảnh qua lại giữa chúng bằng cách gọi Thread::Yield, để minh họa hoạt động bên trong của hệ thống luồng.



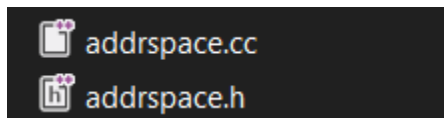
utility.h: Các định nghĩa hữu ích khác, bao gồm các quy trình gỡ lỗi.

Các quy trình gỡ lỗi cho phép người dùng bật các thông báo gỡ lỗi đã chọn, có thể điều khiển được từ các đối số dòng lệnh được chuyển đến Nachos (-d). Bạn được khuyến khích thêm cờ gỡ lỗi của riêng mình. 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)

utility.cc: Quy trình gỡ lỗi. Cho phép người dùng kiểm soát xem có in các câu lệnh DEBUG hay không, dựa trên đối số dòng lệnh.

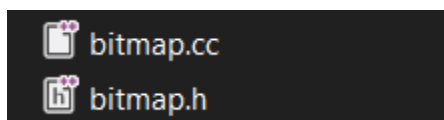
6. userprog



addrspace.h: Các cấu trúc dữ liệu để theo dõi việc thực thi các chương trình của người dùng (không gian địa chỉ).

Trạng thái CPU cấp người dùng được lưu và khôi phục trong luồng thực thi chương trình người dùng.

addrspace.cc: Các quy trình quản lý không gian địa chỉ (thực thi chương trình người dùng).

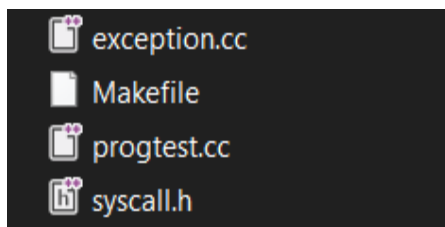


bitmap.h: Cấu trúc dữ liệu xác định một bitmap - một mảng bit, mỗi bit có thể được bật hoặc tắt.

Được biểu diễn dưới dạng một mảng các số nguyên không dấu, trên đó chúng ta thực hiện số học modulo để tìm bit mà chúng ta quan tâm. Bitmap có thể được tham số hóa với số lượng bit được quản lý.

bitmap.c: Các quy trình quản lý một bitmap, gồm các hàm như:

- **Mark():** Đặt bit "thứ n" trong một bitmap.
- **Clear():** Xóa một bit của bitmap.
- **Test():** kiểm tra 1 bit, trả về TRUE nếu bit “thứ n” đã được đặt.
- **Find():** Trả về số của bit đầu tiên đã được xóa. Nếu không có bit nào rõ ràng, trả về -1.
- **NumClear():** Trả về số bit clear trong bitmap. (Nói cách khác, có bao nhiêu bit không được phân bổ?)
- **Print():** In nội dung của bitmap để gỡ lỗi.
- **FetchFrom():** Khởi tạo nội dung của bitmap từ tệp Nachos.
- **WriteBack():** Lưu nội dung của bitmap vào file Nachos.



exception.cc: Đây được coi như điểm truy cập vào nhân Nachos từ các chương trình người dùng. Có 2 cách điều có thể khiến bộ điều khiển trở lại đây từ mã người dùng:

- syscall - Mã người dùng trực tiếp yêu cầu để gọi một thủ tục trong nhân Nachos (Như Halt, shell, sort,..)
- exceptions - Mã người dùng yêu cầu thực hiện điều gì đó mà CPU không thể xử lý.
Ví dụ: truy cập bộ nhớ không tồn tại, lỗi số học,..

Các Interrupts (cũng có thể khiến bộ điều khiển trở lại nhân Kernel từ mã người dùng) thì được xử lý ở nơi khác.

Bộ xử lý Exceptions (ExceptionHandler): Điểm truy cập vào nhân Nachos được gọi khi một chương trình người dùng đang thực thi và sẽ thực hiện 1 syscall hoặc tạo ngoại lệ (exception) địa chỉ hoặc số học.

- Ngoại lệ địa chỉ (addressing exception): xuất hiện khi người dùng gọi, trở tới vị trí không tồn tại.
- Ngoại lệ số học (arithmetic exception): xuất hiện khi người dùng muốn thực hiện các phép tính CPU không tính toán được hay vi phạm quy tắc tính toán.

Nhớ phải tăng số lượng của program counter trước khi trả về nếu không sẽ rơi vào vòng lặp liên tục gọi system call đó.

syscall.h: chứa định nghĩa các thủ tục và các mã system calls liên quan của chương trình.

MakeFile: Tập tin MakeFile này chứa mã Kernel dùng để xây dựng các thành phần cần thiết (dependencies) và trình tự biên dịch cho đối số đa chương (multiprogramming). Phần thiết lập định nghĩa (defines) giả sử đa chương được thực hiện trước tệp system. Nếu không, hãy sử dụng định nghĩa "filesys first".

Phần 2: Hiểu thiết kế

1. Tổng quan về NachOS

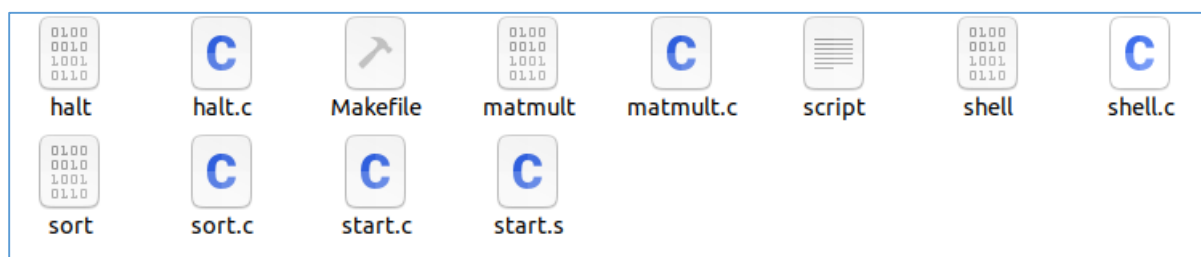
NachOS 3.4 chạy trên môi trường Linux với kiến trúc x86. Về cơ bản, NachOS chỉ là một phần mềm trên Linux, ta cần dùng GCC để biên dịch chương trình này. Quá trình biên dịch NachOS (thứ tự, các liên kết...) được quy định bởi file Makefile trong thư mục code. **Compiler GCC** của Linux được sử dụng để biên dịch toàn bộ mã nguồn của NachOS, tạo ra file thực thi NachOS trong thư mục /code/userprog để sử dụng.

2. Thiết kế:

1) Hệ thống NachOS được chia làm 3 thành phần chính:

- Chương trình ứng dụng (thư mục /code/test): do người lập trình ứng dụng viết.

Các file có sẵn trong thư mục này là:



Trong đó, **Makefile** chịu trách nhiệm quy định quá trình **GCC 2.95.3** biên dịch các file chương trình người dùng thành các file .coff rồi được chuyển thành file .noff để thực thi nhờ chương trình **coff2noff** (thư mục /bin).

GCC 2.95.3 được gọi là cross compiler (cross compiler là compiler tạo ra những đoạn mã thực thi chạy được trên một nền tảng khác với nền tảng mà compiler đó đang sử dụng). Cross compiler **GCC 2.95.3** chỉ chịu trách nhiệm biên dịch cho thư mục /code/test, còn các thư mục còn lại thì do compiler GCC của Linux chịu trách nhiệm (được cài đặt qua command `sudo apt-get install gcc`).

- Máy ảo MIPS (thư mục /code/machine): giả lập các thành phần CPU, Register, RAM, I/O , File System...

Một số lớp quan trọng được khai báo trong file **machine.h**:

- Lớp **Instruction** mô tả mã máy cho máy ảo MIPS:

```
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 máy ảo MIPS (phần đầu của lớp):

```
class Machine {
public:
    Machine(bool debug);      // Initialize the simulation of the hardware
                        // for running user programs
    ~Machine();               // De-allocate the data structures

    // Routines callable by the Nachos kernel
    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
};
```

Trong đoạn khai báo lớp trên, ta có thể quan sát thấy một số phương thức quan trọng là **Run()** - chạy một chương trình người dùng, **ReadRegister()** - đọc nội dung của thanh ghi, **WriteRegister()** - ghi giá trị lên một thanh ghi. Ngoài ra, lớp Machine còn có nhiều phương thức và dữ liệu quan trọng khác.

- Hệ điều hành NachOS (/code/userprog, /code/thread...): phần con của hệ thống NachOS, cấu thành từ một vài file mã nguồn, có trách nhiệm xử lý các system call.

2) Quá trình biên dịch một chương trình người dùng:

Ví dụ chương trình trong file halt.c:

Quá trình biên dịch chương trình này gồm 3 giai đoạn như sau:

- Chương trình halt.c được biên dịch bởi cross compiler thành tập tin halt.s là mã hợp ngữ chạy trên kiến trúc MIPS.
- Tập tin halt.s này sẽ được liên kết với tập tin start.s để tạo thành tập tin halt.coff, là định dạng thực thi trên hệ điều hành Linux cho kiến trúc máy MIPS. Do Nachos được xây dựng đơn giản nhằm mục đích giúp tìm hiểu nên người ta sử dụng tập tin start.s thay cho toàn bộ thư viện C.

- Tập tin halt.coff được chuyển thành tập tin halt.noff, là định dạng thực thi trên hệ điều hành Nachos cho kiến trúc máy MIPS, sử dụng chương trình coff2noff được cung cấp sẵn trong thư mục /code/bin/.

File start.s giống như một thư viện mà người phát triển hệ điều hành dành cho người lập trình ứng dụng, trong đó bao gồm các system call được cài đặt phục vụ cho việc viết các chương trình người dùng.

3) Thực thi một chương trình người dùng:

Ví dụ thực thi chương trình halt (dùng để tắt NachOS):

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

Trong đó:

./userprog/nachos: mở phần mềm NachOS trong thư mục userprog đã được biên dịch.

-rs 1023 -x: các tham số cần thiết.

./test/halt: tên chương trình người dùng cần thực thi trong thư mục test.

4) User space và kernel/system space:

User space: vùng nhớ của những chương trình ứng dụng. Nó gồm mã thực thi bị giới hạn truy cập. Nó là không gian địa chỉ mà các process user thông thường chạy. Những tiến trình này không thể truy cập trực tiếp tới kernel space được. Hầu hết các chương trình mà ta sử dụng trên máy tính là thuộc user space.

Kernel space: vùng nhớ của hệ điều hành NachOS. Nó gồm các mã thực thi có quyền truy cập không hạn chế vào bất kỳ không gian địa chỉ nào của memory và tới bất kỳ phần cứng nào. Bất kỳ sự không ổn định nào bên trong mã thực thi kernel cũng có thể dẫn đến lỗi hệ thống hoàn toàn.

5) Lớp SynchConsole:

Lớp **SynchConsole** được cài đặt trong 2 file là **synchcons.h** và **synchcons.cc**. Lớp SynchConsole cung cấp 2 phương thức quan trọng là phương thức **Write** và **Read**.

Trong đó:

- **int Write(char *from, int numBytes)**: ghi vào thiết bị I/O một lượng numBytes của buffer from, trả về số byte ghi được.

- **int Read(char *into, int numBytes)**: đọc vào buffer into từ thiết bị I/O một lượng numBytes, trả về số byte đọc được.

6) Viết một system call mới và cài đặt chương trình người dùng để kiểm tra:

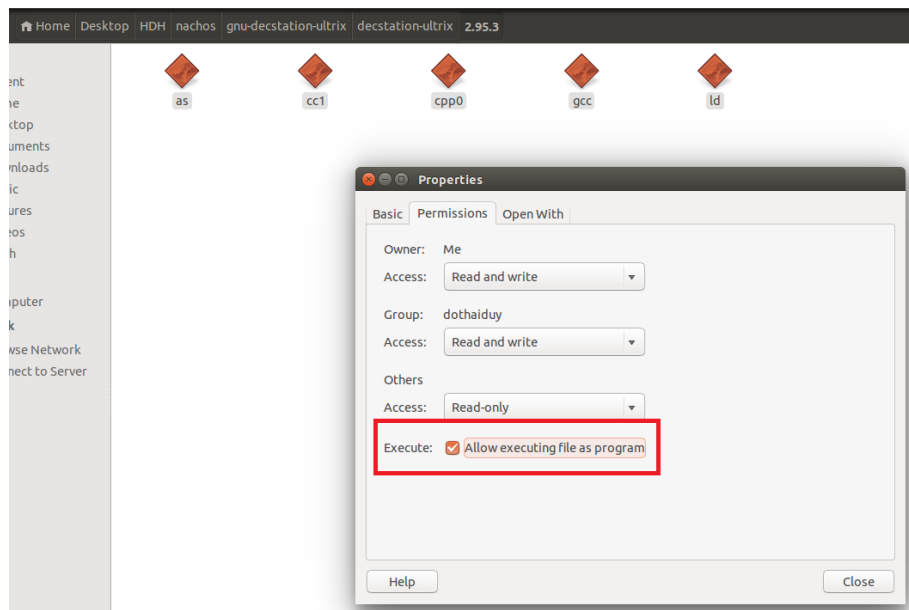
Để viết một system call mới và kiểm tra, ta thực hiện lần lượt theo các bước:

- Khai báo system call mới trong file **/code/userprog/syscall.h**, bao gồm khai báo mã code và khai báo prototype cho system call mới.
- Chỉnh sửa lại file **/code/test/start.c** và **/code/test/start.s** để system call có thể được biên dịch thành mã thực thi trên máy ảo MIPS. Điểm cần lưu ý là: thanh ghi r2 được dùng để chứa mã code của system call (khai báo trong syscall.h) và chứa giá trị trả về của system call. Các tham số khác lần lượt được chứa trong các thanh ghi từ r4 đến r7.
- Định nghĩa xử lý của system call mới trong file **/userprog/exception.cc**. Điểm cần lưu ý là: các biến cần sử dụng phải được khai báo ở đầu hàm **ExceptionHandler()**, nếu không sẽ gặp lỗi.
- Viết chương trình người dùng để kiểm tra. Để kiểm tra system call, ta viết một chương trình người dùng dưới dạng file .c trong thư mục **/code/test**. Điểm cần lưu ý là: vì cross compiler được sử dụng trong thư mục này đã cũ nên các biến cần phải được khai báo ở trên cùng của chương trình, nếu không sẽ gặp phải lỗi khi biên dịch.
- Sửa lại **Makefile** trong **/code/test** để biên dịch được chương trình .c mới.
- Biên dịch lại NachOS.
- Kiểm tra bằng chương trình người dùng với command được nhắc đến ở mục 3, thay **halt** bằng tên chương trình mong muốn.

Phần 3: Exceptions và System calls

1. Cấu hình sơ lược để thực hiện code nachos

- Vào thư mục **nachos/nachos-3.4/code/Makefile** và sửa “gmake” thành “make”.
- Cấp quyền (permission) cho 5 file trong thư mục **gnu-decstation-ultrix** như hình dưới:



- Thêm lớp SynchConsole vào nachos
- Copy 2 file **Synchcons.cc** và **Synchcons.h** vào **code/threads**.
 - Vào **code/threads/system.h** khai báo **#include "synchcons.h"** và khai báo 1 biến cục bộ **extern Synchconsole *gsc**
 - Vào **code/threads/system.c** khai báo một biến cục bộ (**Synchconsole *gsc**), cấp phát nó và giải phóng vùng nhớ.
 - **code/Makefile.common** khai báo 3 file **Synchcons.cc**, **Synchcons.h**, **Synchcons.o** trong **USERPROG_C**, **USERPROG_H**, **USERPROG_O**.

2. Viết các Exceptions và system calls

a. Viết lại file exception.cc

Các bước cài đặt:

- Vào **code/machine/machine.h** để xem các loại exception.
- Vào **code/userprog/exception.cc** viết thêm các exception dưới dạng switch case: Trường hợp đặc biệt duy nhất là no exception sẽ trả quyền điều khiển về HĐH, còn syscall

exceptions sẽ được xử lý bởi các hàm chúng ta viết cho user system calls. Với tất cả exceptions khác, HĐH hiển thị ra một thông báo lỗi và Halt hệ thống.

- Biên dịch lại chương trình và chạy **./userprog/nachos -rs 1023 -x ./test/halt** để xem kết quả.

b. Viết lại cấu trúc điều khiển của chương trình để nhận các Nachos system calls

- Vào **code/userprog/syscall.h** để xem các loại system calls.
- Chuyển code **if...else** về **switch...case** cho tiện trong việc kiểm tra.
- Viết các loại system calls trong case **SyscallException**.

c. Viết mã để tăng giá trị biến program counter

Mục đích của việc này là tăng program counter trước khi system call trả kết quả về.

Các bước cài đặt:

- Vào **code/machine/mipssim.cc** để xem đoạn mã.
- Vào **code/userprog/exception.cc** viết 1 hàm có tên **void IncreaseProgramCounter()**.
- Trong hàm này ta thực hiện: Vị trí trước đó gán cho vị trí hiện tại; Vị trí hiện tại gán cho vị trí tiếp theo; Vị trí tiếp theo cộng 4 byte.

d. Cài đặt system call int ReadInt()

* Các bước cài đặt:

- Trong file **code/userprog/syscall.h** thêm dòng khai báo syscall mới:
 - + **#define SC_ReadInt 11**
 - + **int ReadInt();**
- Vào **code/test/start.c, start.s** viết hàm **ReadInt** bằng MIPS.
- Vào **code/userprog/exception.cc** viết chương trình xử lý trong "**case SC_ReadInt**".
- Trong thư mục **/code/test** tạo 1 file **readint.c**
- Vào **code/test/makefile.cc** :
 - + Khai báo readint trong dòng "all:"
 - + Viết code quá trình biên dịch tạo ra file readint.

* Thiết kế:

- Trong file **readint.c** ta gọi hàm **ReadInt()** trả về một tham số nên giá trị nó được

lưu vào thanh ghi \$2.

- Trong file **start.c**, ta đã lưu **SC_ReadInt** vào thanh ghi \$2.
- Trong file **exception.c**, trong "**case SC_ReadInt**" ta thực hiện chuyển chuỗi số người dùng nhập vào nhờ hàm **Read** thành số nguyên và ghi số đó bằng cách ghi vào thanh ghi \$2 (**machine -> WriteRegister(2,sonhap)**).

e. Cài đặt system call int **PrintInt(int number)**

* Các bước:

- Trong file **code/userprog/syscall.h** thêm dòng khai báo syscall mới:
 - + **#define SC_PrintInt 12**
 - + **void PrintInt(int);**
- Vào **code/test/start.c**, **start.s** viết hàm **PrintInt** bằng MIPS.
- Vào **code/userprog/exception.cc** viết chương trình xử lý trong "**case SC_PrintInt**"
- Trong thư mục **code/test** tạo 1 file **printint.c**
- Vào **code/test/makefile.cc**:
 - + Khai báo **printint** trong dòng "all:"
 - + Viết code quá trình biên dịch tạo ra file **printint**.

* Thiết kế:

- Trong file **printint.c**, do **PrintInt(int number)** có 1 tham số truyền vào nên ta sẽ đọc giá trị ở thanh ghi \$4.
- Trong file **start.c**, ta đã lưu **SC_PrintInt** vào thanh ghi \$2.
- Trong file **exception.c**, trong "**case SC_PrintInt**" ta lấy ra được số cần xuất bằng cách đọc thanh ghi \$4 (**machine -> ReadRegister(4)**). Thực hiện chuyển số thành chuỗi ký tự số và in ra màn hình bằng hàm **Write**.

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

* Các bước:

- Trong file **code/userprog/syscall.h** thêm dòng khai báo syscall mới:
 - + **#define SC_ReadChar 13**
 - + **char ReadChar();**

- Vào **code/test/start.c, start.s** viết hàm **ReadChar** bằng MIPS.
- Vào **code/userprog/exception.cc** viết chương trình xử lý trong "**case SC_ReadChar**"
- Trong thư mục **code/test** tạo 1 file **readchar.c**
- Vào **code/test/makefile.cc** :
 - + Khai báo **readchar** trong dòng "all:"
 - + Viết code quá trình biên dịch tạo ra file **readchar**.

* Thiết kế:

- Trong **exception.cc**, khi người dùng nhập trên console sẽ trả về giá trị **char*** khi dùng hàm **Read()** của biến toàn cục **gsc** trong file **synchcons.h**
- Ghi giá trị vừa tìm được vào thanh ghi \$2.
g. Cài đặt system call **void PrintChar(char character)**

* Các bước:

- Trong file **code/userprog/syscall.h** thêm dòng khai báo **syscall** mới:
 - + **#define SC_PrintChar 14**
 - + **void PrintChar(char);**
- Vào **code/test/start.c, start.s** viết hàm **PrintChar** bằng MIPS.
- Vào **code/userprog/exception.cc** viết chương trình xử lý trong "**case SC_PrintChar**"
- Trong thư mục **code/test** tạo 1 file **printchar.c**
- Vào **code/test/makefile.cc** :
 - + Khai báo **printchar** trong dòng "all:"
 - + Viết code quá trình biên dịch tạo ra file **printchar**.

* Thiết kế:

- Đọc giá trị của tham số truyền vào tại thanh ghi \$4 => Trả về **int**.
- Trong **exception.cc**, ta lấy ký tự ra từ thanh ghi \$4 sau đó ghi giá trị ra màn hình bằng hàm **Write** trong file **synchcons.h**.

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

* Các bước:

- Trong file **code/userprog/syscall.h** thêm dòng khai báo syscall mới:
+ **#define SC_ReadString 15**
+ **void ReadString(char[], int);**
- Vào **code/test/start.c, start.s** viết hàm ReadString bằng MIPS.
- Vào **code/userprog/exception.cc** viết chương trình xử lý trong “**case SC_ReadString**”
- Trong thư mục **code/test** tạo 1 file **readstring.c**.
- Vào **code/test/makefile.cc**:
+ Khai báo readstring tại dòng “all”.
+ Viết code quá trình biên dịch tạo ra file readstring.

* Thiết kế:

- Đọc địa chỉ của ký tự đầu tiên của chuỗi nhập vào tại thanh ghi \$4 => trả về int.
- Đọc giá trị của độ dài chuỗi cho phép tại thanh ghi \$5 => kiểu int.
- Cấp phát bộ nhớ động cho một mảng ký tự có độ dài cho phép vừa đọc được.
- Dựa vào hàm **Read** của biến toàn cục gsc trong file **synchcons.h**, ta sẽ biết được số byte mà chương trình thực sự đọc được. Sau đó copy chuỗi từ vùng nhớ **systemspace** sang **userspace** bằng hàm **System2User(int, int, char*)** để có được chuỗi với độ dài thực tế.
- Ghi chuỗi vừa đọc được vào thanh ghi \$2.

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

* Các bước:

- Trong file **code/userprog/syscall.h** thêm dòng khai báo syscall mới:
+ **#define SC_PrintString 16**
+ **void PrintString(char[]);**
- Vào **code/test/start.c, start.s** viết hàm **PrintString** bằng MIPS.
- Vào **code/userprog/exception.cc** viết chương trình xử lý trong “**case SC_PrintString**”

- Trong thư mục **code/test** tạo 1 file **printstring.c**.
- Vào **code/test/makefile.cc**:
 - + Khai báo **printstring** tại dòng “all”.
 - + Viết code quá trình biên dịch tạo ra file **printstring**.

* Thiết kế:

- Đọc địa chỉ của ký tự đầu tiên của chuỗi nhập vào tại thanh ghi \$4 => trả về int.
- Copy chuỗi từ vùng nhớ userspace sang systemspace nhờ hàm **User2System(char*, int)**.
- Chạy 1 vòng lặp đến khi nào gặp ký tự kết thúc của chuỗi thì dừng để xuất từng ký tự của chuỗi ra màn hình bằng hàm **Write** qua biến toàn cục **gsc** trong file **synchcons.h**.
- Ta kết thúc chuỗi có được bằng ký tự xuống dòng và cũng xuất ký tự đó ra màn hình.
- Ghi độ dài chuỗi vừa đọc vào thanh ghi \$2.

j. Cài đặt chương trình help

* Các bước:

- Trong thư mục **code/test** tạo 1 file **help.c**.
- Vào **code/test/makefile.cc**:
 - + Khai báo **help** tại dòng “all”.
 - + Viết code quá trình biên dịch tạo ra file **help**.

* Thiết kế:

- Gọi lại system call **PrintString(char[])** để giới thiệu sơ qua về thông tin nhóm và mô tả vắn tắt về chương trình **ascii** và chương trình sắp xếp mảng theo thuật toán bubble sort.

k. Cài đặt chương trình ascii

* Các bước:

- Trong thư mục **code/test** tạo 1 file **ascii.c**.
- Vào **code/test/makefile.cc**:
 - + Khai báo **ascii** tại dòng “all”.

- + Viết code quá trình biên dịch tạo ra file ascii.

* Thiết kế:

- Gọi lại system call PrintChar(char) để xuất các kí tự ascii ra màn hình (127 kí tự).

1. Cài đặt chương trình sort

* Các bước:

- Trong thư mục **code/test** tạo 1 file **sort.c**.
- Vào **code/test/makefile.cc**:
 - + Khai báo sort tại dòng “all”.
 - + Viết code quá trình biên dịch tạo ra file sort.

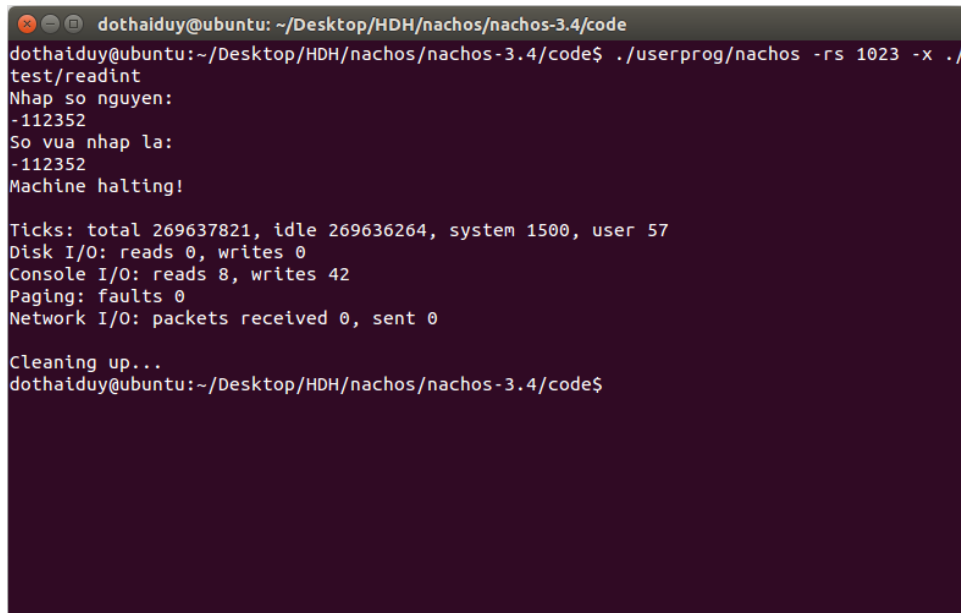
* Thiết kế:

- Cho người dùng nhập vào số phần tử của mảng số nguyên, sau đó nhập từng phần tử của mảng bằng system call ReadInt(int)
- Kế tiếp ta sẽ sắp xếp các phần tử trong mảng dựa vào thuật toán bubble sort.
- Cuối cùng xuất mảng đã được sắp xếp theo chiều tăng dần ra màn hình qua system call PrintInt(int) để người dùng biết được là chương trình đã được thực hiện và thành công.

III. Hình ảnh Demo

Trong mỗi chương trình tại em có dùng hàm PrintString, PrintChar kết hợp với nhau để cho ra output rõ ràng hơn.

1. Demo system call ReadInt

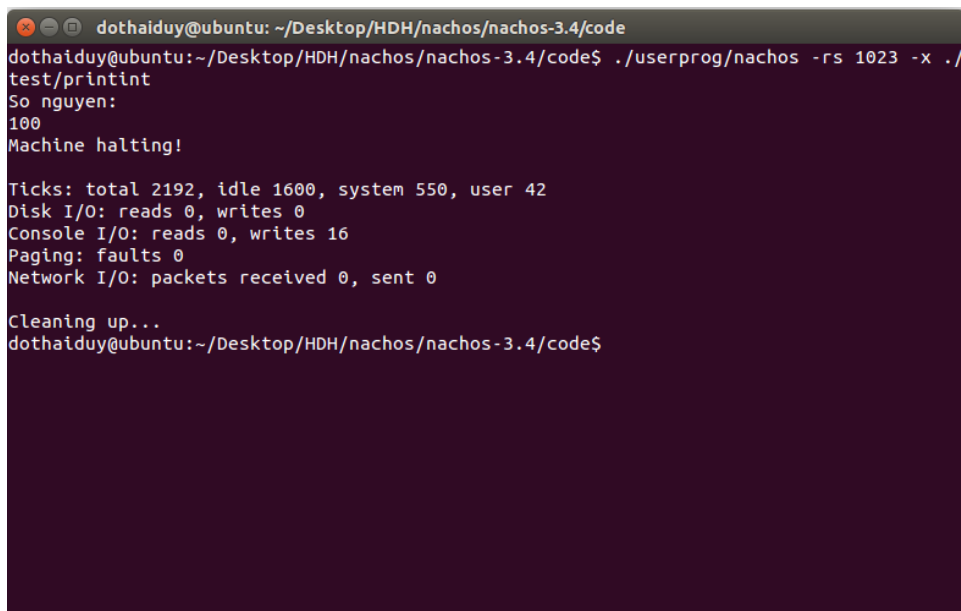


```
dothaiduy@ubuntu: ~/Desktop/HDH/nachos/nachos-3.4/code
dothaiduy@ubuntu:~/Desktop/HDH/nachos/nachos-3.4/code$ ./userprog/nachos -rs 1023 -x ./
test/readint
Nhap so nguyen:
-112352
So vua nhap la:
-112352
Machine halting!

Ticks: total 269637821, idle 269636264, system 1500, user 57
Disk I/O: reads 0, writes 0
Console I/O: reads 8, writes 42
Paging: faults 0
Network I/O: packets received 0, sent 0

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

2. Demo system call PrintInt



```
dothaiduy@ubuntu: ~/Desktop/HDH/nachos/nachos-3.4/code
dothaiduy@ubuntu:~/Desktop/HDH/nachos/nachos-3.4/code$ ./userprog/nachos -rs 1023 -x ./
test/printint
So nguyen:
100
Machine halting!

Ticks: total 2192, idle 1600, system 550, user 42
Disk I/O: reads 0, writes 0
Console I/O: reads 0, writes 16
Paging: faults 0
Network I/O: packets received 0, sent 0

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

3. Demo system call ReadChar

```
dothaiduy@ubuntu: ~/Desktop/HDH/nachos/nachos-3.4/code
dothaiduy@ubuntu:~/Desktop/HDH/nachos/nachos-3.4/code$ ./userprog/nachos -rs 1023 -x ./
test/readchar
Nhap 1 ky tu:
g
Ky tu vua nhap la:
g
Machine halting!

Ticks: total 117806072, idle 117804734, system 1280, user 58
Disk I/O: reads 0, writes 0
Console I/O: reads 2, writes 37
Paging: faults 0
Network I/O: packets received 0, sent 0

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

4. Demo system call PrintChar

```
dothaiduy@ubuntu: ~/Desktop/HDH/nachos/nachos-3.4/code
dothaiduy@ubuntu:~/Desktop/HDH/nachos/nachos-3.4/code$ ./userprog/nachos -rs 1023 -x ./
test/printchar
Ky tu duoc xuat la:
a
Machine halting!

Ticks: total 3172, idle 2300, system 830, user 42
Disk I/O: reads 0, writes 0
Console I/O: reads 0, writes 23
Paging: faults 0
Network I/O: packets received 0, sent 0

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


5. Demo system call ReadString

```
dothaiduy@ubuntu: ~/Desktop/HDH/nachos/nachos-3.4/code
dothaiduy@ubuntu:~/Desktop/HDH/nachos/nachos-3.4/code$ ./userprog/nachos -rs 1023 -x ./
test/readstring
Nhap 1 chuoai:
do an 2 he dieu hanh 19-1
Chuoai vua nhap la:
do an 2 he dieu hanh 19-1
Machine halting!

Ticks: total 864940213, idle 864937783, system 2380, user 50
Disk I/O: reads 0, writes 0
Console I/O: reads 26, writes 61
Paging: faults 0
Network I/O: packets received 0, sent 0

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

6. Demo system call PrintString

```
dothaiduy@ubuntu: ~/Desktop/HDH/nachos/nachos-3.4/code
dothaiduy@ubuntu:~/Desktop/HDH/nachos/nachos-3.4/code$ ./userprog/nachos -rs 1023 -x ./
test/printstring
lap trinh nachos.
Machine halting!

Ticks: total 2480, idle 1800, system 650, user 30
Disk I/O: reads 0, writes 0
Console I/O: reads 0, writes 18
Paging: faults 0
Network I/O: packets received 0, sent 0

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

7. Demo chương trình help

```
dothaiduy@ubuntu: ~/Desktop/HDH/nachos/nachos-3.4/code
dothaiduy@ubuntu:~/Desktop/HDH/nachos/nachos-3.4/code$ ./userprog/nachos -rs 1023 -x ./test/help
-----
--- Thông tin nhóm: ---
+ Le Tien Dat          19120189.
+ Nguyen Van Dat       19120190.
+ Nguyen Tien Toan     19120399.
+ Do Thai Duy          19120492.
+ Nguyen Ba Ngoc       19120603.
-----
Mô tả ngắn về 2 chương trình:
+ Chương trình ascii: in ra bảng mã ascii: nachos/nachos-3.4/code/test/ascii.c
+ Chương trình sort: sắp xếp mảng theo thuật toán Bubble sort (Nhập mảng -> Sắp xếp -> Xuất mảng): nachos/nachos-3.4/code/test/sort.c
-----
Machine halting!

Ticks: total 67667, idle 50700, system 16860, user 107
Disk I/O: reads 0, writes 0
Console I/O: reads 0, writes 507
Paging: faults 0
Network I/O: packets received 0, sent 0

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

8. Demo chương trình ascii

In bảng mã ASCII từ 0 đến 127.

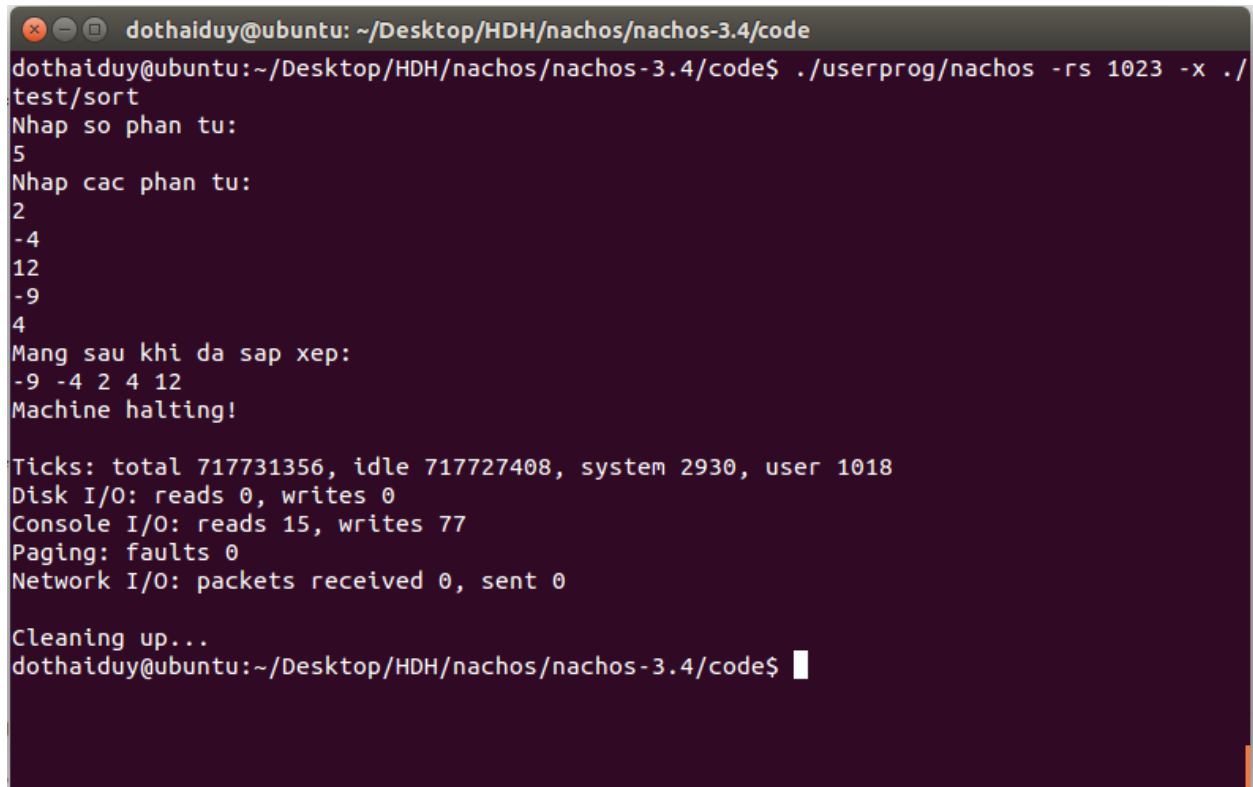
```
dothaiduy@ubuntu: ~/Desktop/HDH/nachos/nachos-3.4/code
dothaiduy@ubuntu:~/Desktop/HDH/nachos/nachos-3.4/code$ ./userprog/nachos -rs 1023 -x ./test/ascii
Bảng mã ASCII là:
 01 02 03 04 05

 06 07 08 09 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50 51 52 53 54 55 56 57 58 59 60 61 62 63 64 65 66 67 68 69 70 71 72 73 74 75 76 77 78 79 80 81 82 83 84 85 86 87 88 89 90 91 92 93 94 95 96 97 98 99 100 101 102 103 104 105 106 107 108 109 110 111 112 113 114 115 116 117 118 119 120 121 122 123 124 125 126 127 128 129 130 131 132 133 134 135 136 137 138 139 140 141 142 143 144 145 146 147 148 149 150 151 152 153 154 155 156 157 158 159 160 161 162 163 164 165 166 167 168 169 170 171 172 173 174 175 176 177 178 179 180 181 182 183 184 185 186 187 188 189 190 191 192 193 194 195 196 197 198 199 200 201 202 203 204 205 206 207 208 209 210 211 212 213 214 215 216 217 218 219 220 221 222 223 224 225 226 227 228 229 230 231 232 233 234 235 236 237 238 239 240 241 242 243 244 245 246 247 248 249 250 251 252 253 254 255 256 257 258 259 260 261 262 263 264 265 266 267 268 269 270 271 272 273 274 275 276 277 278 279 280 281 282 283 284 285 286 287 288 289 290 291 292 293 294 295 296 297 298 299 300 301 302 303 304 305 306 307 308 309 310 311 312 313 314 315 316 317 318 319 320 321 322 323 324 325 326 327 328 329 330 331 332 333 334 335 336 337 338 339 340 341 342 343 344 345 346 347 348 349 350 351 352 353 354 355 356 357 358 359 360 361 362 363 364 365 366 367 368 369 370 371 372 373 374 375 376 377 378 379 380 381 382 383 384 385 386 387 388 389 390 391 392 393 394 395 396 397 398 399 400 401 402 403 404 405 406 407 408 409 410 411 412 413 414 415 416 417 418 419 420 421 422 423 424 425 426 427 428 429 430 431 432 433 434 435 436 437 438 439 440 441 442 443 444 445 446 447 448 449 450 451 452 453 454 455 456 457 458 459 460 461 462 463 464 465 466 467 468 469 470 471 472 473 474 475 476 477 478 479 480 481 482 483 484 485 486 487 488 489 490 491 492 493 494 495 496 497 498 499 500 501 502 503 504 505 506 507 508 509 510 511 512 513 514 515 516 517 518 519 520 521 522 523 524 525 526 527 528 529 530 531 532 533 534 535 536 537 538 539 540 541 542 543 544 545 546 547 548 549 550 551 552 553 554 555 556 557 558 559 560 561 562 563 564 565 566 567 568 569 570 571 572 573 574 575 576 577 578 579 580 581 582 583 584 585 586 587 588 589 590 591 592 593 594 595 596 597 598 599 600 601 602 603 604 605 606 607 608 609 610 611 612 613 614 615 616 617 618 619 620 621 622 623 624 625 626 627 628 629 630 631 632 633 634 635 636 637 638 639 640 641 642 643 644 645 646 647 648 649 650 651 652 653 654 655 656 657 658 659 660 661 662 663 664 665 666 667 668 669 670 671 672 673 674 675 676 677 678 679 680 681 682 683 684 685 686 687 688 689 690 691 692 693 694 695 696 697 698 699 700 701 702 703 704 705 706 707 708 709 710 711 712 713 714 715 716 717 718 719 720 721 722 723 724 725 726 727 728 729 730 731 732 733 734 735 736 737 738 739 740 741 742 743 744 745 746 747 748 749 750 751 752 753 754 755 756 757 758 759 760 761 762 763 764 765 766 767 768 769 770 771 772 773 774 775 776 777 778 779 780 781 782 783 784 785 786 787 788 789 790 791 792 793 794 795 796 797 798 799 800 801 802 803 804 805 806 807 808 809 810 811 812 813 814 815 816 817 818 819 820 821 822 823 824 825 826 827 828 829 830 831 832 833 834 835 836 837 838 839 840 841 842 843 844 845 846 847 848 849 850 851 852 853 854 855 856 857 858 859 860 861 862 863 864 865 866 867 868 869 870 871 872 873 874 875 876 877 878 879 880 881 882 883 884 885 886 887 888 889 890 891 892 893 894 895 896 897 898 899 900 901 902 903 904 905 906 907 908 909 910 911 912 913 914 915 916 917 918 919 920 921 922 923 924 925 926 927 928 929 930 931 932 933 934 935 936 937 938 939 940 941 942 943 944 945 946 947 948 949 950 951 952 953 954 955 956 957 958 959 960 961 962 963 964 965 966 967 968 969 970 971 972 973 974 975 976 977 978 979 980 981 982 983 984 985 986 987 988 989 990 991 992 993 994 995 996 997 998 999 1000 1001 1002 1003 1004 1005 1006 1007 1008 1009 1010 1011 1012 1013 1014 1015 1016 1017 1018 1019 1020 1021 1022 1023 1024 1025 1026 1027 1028 1029 1030 1031 1032 1033 1034 1035 1036 1037 1038 1039 1040 1041 1042 1043 1044 1045 1046 1047 1048 1049 1050 1051 1052 1053 1054 1055 1056 1057 1058 1059 1060 1061 1062 1063 1064 1065 1066 1067 1068 1069 1070 1071 1072 1073 1074 1075 1076 1077 1078 1079 1080 1081 1082 1083 1084 1085 1086 1087 1088 1089 1090 1091 1092 1093 1094 1095 1096 1097 1098 1099 1100 1101 1102 1103 1104 1105 1106 1107 1108 1109 1110 1111 1112 1113 1114 1115 1116 1117 1118 1119 1120 1121 1122 1123 1124 1125 1126 1127 1128 1129 1130 1131 1132 1133 1134 1135 1136 1137 1138 1139 1140 1141 1142 1143 1144 1145 1146 1147 1148 1149 1150 1151 1152 1153 1154 1155 1156 1157 1158 1159 1160 1161 1162 1163 1164 1165 1166 1167 1168 1169 1170 1171 1172 1173 1174 1175 1176 1177 1178 1179 1180 1181 1182 1183 1184 1185 1186 1187 1188 1189 1190 1191 1192 1193 1194 1195 1196 1197 1198 1199 1200 1201 1202 1203 1204 1205 1206 1207 1208 1209 1210 1211 1212 1213 1214 1215 1216 1217 1218 1219 1220 1221 1222 1223 1224 1225 1226 1227 1228 1229 1230 1231 1232 1233 1234 1235 1236 1237 1238 1239 1240 1241 1242 1243 1244 1245 1246 1247 1248 1249 1250 1251 1252 1253 1254 1255 1256 1257 1258 1259 1260 1261 1262 1263 1264 1265 1266 1267 1268 1269 1270 1271 1272 1273 1274 1275 1276 1277 1278 1279 1280 1281 1282 1283 1284 1285 1286 1287 1288 1289 1290 1291 1292 1293 1294 1295 1296 1297 1298 1299 1300 1301 1302 1303 1304 1305 1306 1307 1308 1309 1310 1311 1312 1313 1314 1315 1316 1317 1318 1319 1320 1321 1322 1323 1324 1325 1326 1327 1328 1329 1330 1331 1332 1333 1334 1335 1336 1337 1338 1339 1340 1341 1342 1343 1344 1345 1346 1347 1348 1349 1350 1351 1352 1353 1354 1355 1356 1357 1358 1359 1360 1361 1362 1363 1364 1365 1366 1367 1368 1369 1370 1371 1372 1373 1374 1375 1376 1377 1378 1379 1380 1381 1382 1383 1384 1385 1386 1387 1388 1389 1390 1391 1392 1393 1394 1395 1396 1397 1398 1399 1400 1401 1402 1403 1404 1405 1406 1407 1408 1409 1410 1411 1412 1413 1414 1415 1416 1417 1418 1419 1420 1421 1422 1423 1424 1425 1426 1427 1428 1429 1430 1431 1432 1433 1434 1435 1436 1437 1438 1439 1440 1441 1442 1443 1444 1445 1446 1447 1448 1449 1450 1451 1452 1453 1454 1455 1456 1457 1458 1459 1460 1461 1462 1463 1464 1465 1466 1467 1468 1469 1470 1471 1472 1473 1474 1475 1476 1477 1478 1479 1480 1481 1482 1483 1484 1485 1486 1487 1488 1489 1490 1491 1492 1493 1494 1495 1496 1497 1498 1499 1500 1501 1502 1503 1504 1505 1506 1507 1508 1509 1510 1511 1512 1513 1514 1515 1516 1517 1518 1519 1520 1521 1522 1523 1524 1525 1526 1527 1528 1529 1530 1531 1532 1533 1534 1535 1536 1537 1538 1539 1540 1541 1542 1543 1544 1545 1546 1547 1548 1549 1550 1551 1552 1553 1554 1555 1556 1557 1558 1559 1560 1561 1562 1563 1564 1565 1566 1567 1568 1569 1570 1571 1572 1573 1574 1575 1576 1577 1578 1579 1580 1581 1582 1583 1584 1585 1586 1587 1588 1589 1590 1591 1592 1593 1594 1595 1596 1597 1598 1599 1600 1601 1602 1603 1604 1605 1606 1607 1608 1609 1610 1611 1612 1613 1614 1615 1616 1617 1618 1619 1620 1621 1622 1623 1624 1625 1626 1627 1628 1629 1630 1631 1632 1633 1634 1635 1636 1637 1638 1639 1640 1641 1642 1643 1644 1645 1646 1647 1648 1649 1650 1651 1652 1653 1654 1655 1656 1657 1658 1659 1660 1661 1662 1663 1664 1665 1666 1667 1668 1669 1670 1671 1672 1673 1674 1675 1676 1677 1678 1679 1680 1681 1682 1683 1684 1685 1686 1687 1688 1689 1690 1691 1692 1693 1694 1695 1696 1697 1698 1699 1700 1701 1702 1703 1704 1705 1706 1707 1708 1709 1710 1711 1712 1713 1714 1715 1716 1717 1718 1719 1720 1721 1722 1723 1724 1725 1726 1727 1728 1729 1730 1731 1732 1733 1734 1735 1736 1737 1738 1739 1740 1741 1742 1743 1744 1745 1746 1747 1748 1749 1750 1751 1752 1753 1754 1755 1756 1757 1758 1759 1760 1761 1762 1763 1764 1765 1766 1767 1768 1769 1770 1771 1772 1773 1774 1775 1776 1777 1778 1779 1780 1781 1782 1783 1784 1785 1786 1787 1788 1789 1790 1791 1792 1793 1794 1795 1796 1797 1798 1799 1800 1801 1802 1803 1804 1805 1806 1807 1808 1809 1810 1811 1812 1813 1814 1815 1816 1817 1818 1819 1820 1821 1822 1823 1824 1825 1826 1827 1828 1829 1830 1831 1832 1833 1834 1835 1836 1837 1838 1839 1840 1841 1842 1843 1844 1845 1846 1847 1848 1849 1850 1851 1852 1853 1854 1855 1856 1857 1858 1859 1860 1861 1862 1863 1864 1865 1866 1867 1868 1869 1870 1871 1872 1873 1874 1875 1876 1877 1878 1879 1880 1881 1882 1883 1884 1885 1886 1887 1888 1889 1890 1891 1892 1893 1894 1895 1896 1897 1898 1899 1900 1901 1902 1903 1904 1905 1906 1907 1908 1909 1910 1911 1912 1913 1914 1915 1916 1917 1918 1919 1920 1921 1922 1923 1924 1925 1926 1927 1928 1929 1930 1931 1932 1933 1934 1935 1936 1937 1938 1939 1940 1941 1942 1943 1944 1945 1946 1947 1948 1949 1950 1951 1952 1953 1954 1955 1956 1957 1958 1959 1960 1961 1962 1963 1964 1965 1966 1967 1968 1969 1970 1971 1972 1973 1974 1975 1976 1977 1978 1979 1980 1981 1982 1983 1984 1985 1986 1987 1988 1989 1990 1991 1992 1993 1994 1995 1996 1997 1998 1999 2000 2001 2002 2003 2004 2005 2006 2007 2008 2009 2010 2011 2012 2013 2014 2015 2016 2017 2018 2019 2020 2021 2022 2023 2024 2025 2026 2027 2028 2029 2030 2031 2032 2033 2034 2035 2036 2037 2038 2039 2040 2041 2042 2043 2044 2045 2046 2047 2048 2049 2050 2051 2052 2053 2054 2055 2056 2057 2058 2059 2060 2061 2062 2063 2064 2065 2066 2067 2068 2069 2070 2071 2072 2073 2074 2075 2076 2077 2078 2079 2080 2081 2082 2083 2084 2085 2086 2087 2088 2089 2090 2091 2092 2093 2094 2095 2096 2097 2098 2099 2100 2101 2102 2103 2104 2105 2106 2107 2108 2109 2110 2111 2112 2113 2114 2115 2116 2117 2118 2119 2120 2121 2122 2123 2124 2125 2126 2127 2128 2129 2130 2131 2132 2133 2134 2135 2136 2137 2138 2139 2140 2141 2142 2143 2144 2145 2146 2147 2148 2149 2150 2151 2152 2153 2154 2155 2156 2157 2158 2159 2160 2161 2162 2163 2164 2165 2166 2167 2168 2169 2170 2171 2172 2173 2174 2175 2176 2177 2178 2179 2180 2181 2182 2183 2184 2185 2186 2187 2188 2189 2190 2191 2192 2193 2194 2195 2196 2197 2198 2199 2200 2201 2202 2203 2204 2205 2206 2207 2208 2209 2210 2211 2212 2213 2214 2215 2216 2217 2218 2219 2220 2221 2222 2223 2224 2225 2226 2227 2228 2229 2230 2231 2232 2233 2234 2235 2236 2237 2238 2239 2240 2241 2242 2243 2244 2245 2246 2247 2248 2249 2250 2251 2252 2253 2254 2255 2256 2257 2258 2259 2260 2261 2262 2263 2264 2265 2266 2267 2268 2269 2270 2271 2272 2273 2274 2275 2276 2277 2278 2279 2280 2281 2282 2283 2284 2285 2286 2287 2288 2289 2290 2291 2292 2293 2294 2295 2296 2297 2298 2299 2300 2301 2302 2303 2304 2305 2306 2307 2308 2309 2310 2311 2312 2313 2314 2315 2316 2317 2318 2319 2320 2321 2322 2323 2324 2325 2326 2327 2328 2329 2330 2331 2332 2333 2334 2335 2336 2337 2338 2339 2340 2341 2342 2343 2344 2345 2346 2347 2348 2349 2350 2351 2352 2353 2354 2355 2356 2357 2358 2359 2360 2361 2362 2363 2364 2365 2366 2367 2368 2369 2370 2371 2372 2373 2374 2375 2376 2377 2378 2379 2380 2381 2382 2383 2384 2385 2386 2387 2388 2389 2390 2391 2392 2393 2394 2395 2396 2397 2398 2399 2400 2401 2402 2403 2404 2405 2406 2407 2408 2409 2410 2411 2412 2413 2414 2415 2416 2417 2418 2419 2420 2421 2422 2423 2424 2425 2426 2427 2428 2429 2430 2431 2432 2433 2434 2435 2436 2437 2438 2439 2440 2441 2442 2443 2444 2445 2446 2447 2448 2449 2450 2451 2452 2453 2454 2455 2456 2457 2458 2459 2460 2461 2462 2463 2464 2465 2466 2467 2468 2469 2470 2471 2472 2473 2474 2475 2476 2477 2478 2479 2480 2481 2482 2483 2484 2485 2486 2487 2488 2489 2490 2491 2492 2493 2494 2495 2496 2497 2498 2499 2500 2501 2502 2503 2504 2505 2506 2507 2508 2509 2510 2511 2512 2513 2514 2515 2516 2517 2518 2519 2520 2521 2522 2523 2524 2525 2526 2527 2528 2529 2530 2531 2532 2533 2534 2535 2536 2537 2538 2539 2540 2541 2542 2543 2544 2545 2546 2547 2548 2549 2550 2551 2552 2553 2554 2555 2556 2557 2558 2559 2560 2561
```

9. Demo chương trình sort

Tại em dùng hàm ReadInt() để đọc số nên phải xuống hàng (chứ không thể cách ra) để không xảy ra trường hợp không phải số nguyên thì trả về số 0 và bị lỗi.

Mảng dưới được sắp xếp tăng dần.



```
dothaiduy@ubuntu: ~/Desktop/HDH/nachos/nachos-3.4/code
dothaiduy@ubuntu:~/Desktop/HDH/nachos/nachos-3.4/code$ ./userprog/nachos -rs 1023 -x ./
test/sort
Nhap so phan tu:
5
Nhap cac phan tu:
2
-4
12
-9
4
Mang sau khi da sap xep:
-9 -4 2 4 12
Machine halting!

Ticks: total 717731356, idle 717727408, system 2930, user 1018
Disk I/O: reads 0, writes 0
Console I/O: reads 15, writes 77
Paging: faults 0
Network I/O: packets received 0, sent 0

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

IV. Tài liệu tham khảo

1. Nguồn tham khảo từ tài liệu của thầy Long:

- [1] Bien dich va cai dat Nachos.pdf
- [2] Giao tiep giua HDH Nachos va chương trình người dùng.pdf
- [3] Cach Viet Mot SystemCall.pdf
- [4] Cach Them 1 Lop Vao Nachos.pdf

2. Nguồn tham khảo trên Internet:

- [5] <https://github.com/thoaihuynh2509/HeDieuHanh/tree/master/DoAn1>
- [6] https://www.youtube.com/watch?v=t0jtY1C129s&list=PLRgTVtca98hUgCN2_2vzsAAXPiTFbvHpO&ab_channel=Th%C3%A0nhChungNguy%E1%BB%85n
- [7] <https://drive.google.com/drive/folders/1aQcLxNqebf2DrMBBMD6ArnAvYk0iqUto>