

ĐẠI HỌC QUỐC GIA THÀNH PHỐ HỒ CHÍ MINH
TRƯỜNG ĐẠI HỌC KHOA HỌC TỰ NHIÊN
KHOA CÔNG NGHỆ THÔNG TIN



**Project 02:
NachOS Syscall**

Môn: Hệ điều hành

Giảng viên hướng dẫn: ThS. Lê Việt Long

Thành viên nhóm:

21120279 – Lê Trần Minh Khuê

21120289 – Diệp Quốc Hoàng Nam

21120290 – Hoàng Trung Nam

21120348 – Nguyễn Trần Trình

21120354 – Lương Thanh Tú

Thành phố Hồ Chí Minh, tháng 12 / 2023

LỜI CẢM ƠN

Chúng em xin gửi lời cảm ơn chân thành đến Khoa Công nghệ Thông tin, Trường Đại học Khoa học Tự nhiên Thành phố Hồ Chí Minh đã tạo điều kiện thuận lợi cho chúng em học tập và hoàn thành bài đồ án này. Đặc biệt, chúng em xin bày tỏ lòng biết ơn sâu sắc đến ThS. Lê Viết Long, người thầy đã dày công truyền đạt kiến thức và hướng dẫn chúng em trong quá trình làm bài.

Chúng em đã có gắng vận dụng những kiến thức đã học được để hoàn thành bài đồ án. Nhưng do kiến thức hạn chế và không có nhiều kinh nghiệm thực tiễn nên khó tránh khỏi những thiếu sót trong quá trình nghiên cứu và trình bày. Rất kính mong sự góp ý của quý Thầy để bài báo cáo của chúng em được hoàn thiện hơn.

Một lần nữa, chúng em xin trân trọng cảm ơn sự quan tâm giúp đỡ của thầy trong suốt quá trình thực hiện đồ án này.

Xin trân trọng cảm ơn!

MỤC LỤC

LỜI CẢM ƠN	2
MỤC LỤC.....	3
DANH MỤC HÌNH.....	4
DANH MỤC BẢNG.....	6
1. TỔNG QUAN	7
1.1. THÔNG TIN NHÓM	7
1.2. BẢNG PHÂN CÔNG CÔNG VIỆC.....	7
1.3. CÁC NỘI DUNG ĐÃ HOÀN THÀNH.....	8
1.4. TỔ CHỨC BÀI LÀM VÀ CÁC GHI CHÚ LẬP TRÌNH	8
1.4.1. Tổ chức bài làm	8
1.4.2. Môi trường thực nghiệm:	8
2. NỘI DUNG BÁO CÁO	9
2.1. HIỆU MÃ CHƯƠNG TRÌNH NACHOS	9
2.1.1. Giới thiệu về NachOS:	9
2.1.2. Cài đặt:	9
2.1.3. Biên dịch:	9
2.1.4. Các tệp thành phần:	11
2.2. HIỆU THIẾT KẾ NACHOS	13
2.2.1. Thiết kế NachOS:	13
2.3. CÀI ĐẶT XỬ LÝ CHO EXCEPTIONS	18
2.4. VIẾT LẠI CÁU TRÚC ĐIỀU KHIỂN	20
2.5. TÁCH HÀM TĂNG PROGRAM COUNTER:.....	21
2.6. CÀI ĐẶT TRƯỚC KHI THÊM SYSTEM CALL	22
2.6.1. Thêm lớp SynchConsole	22
2.6.2. Thêm thủ tục User2System và System2User	24

2.6.3. Các bước để cài đặt các system call mới.	25
2.7. SYSTEM CALLS	27
2.7.1. Xử lý số nguyên - Int	28
2.7.2. Xử lý ký tự - Char	30
2.7.3. Xử lý chuỗi - String	32
2.7.4. Thêm các Syscall cho các hàm trên vào syscallException	35
2.8. VIẾT CÁC CHƯƠNG TRÌNH.....	36
2.8.1. Chương trình ‘help’	36
2.8.2. Chương trình ‘ascii’	36
2.8.3. Chương trình ‘sort’	37
2.9. Sửa makefile trong folder test để biên dịch các hàm đã viết.....	38
2.10. Demo	40
2.10.1. Demo 2 hàm read-print Int:.....	40
2.10.2. Demo ReadChar() – PrintChar()	41
2.10.3. Demo ReadString và PrintString	42
2.10.4. Demo chương trình help	43
2.10.5. Demo chương trình ascii.....	43
2.10.6. Demo chương trình sort	44
3. TÀI LIỆU THAM KHẢO	45

DANH MỤC HÌNH

Hình 2.1.1. Sửa file 'Makefile'	10
Hình 2.1.2. Biên dịch NachOS bằng lệnh 'make'	10
Hình 2.1.3. Chạy thử chương trình Halt trong folder threads.....	11
Hình 2.1.4. Các tệp tin/ thư mục trong folder nachos-3.4	11
Hình 2.2.1. Minh họa quá trình biên dịch file halt.c	15

Hình 2.2.2. OP_SYSCALL trong mipssim.cc	17
Hình 2.2.3. RaiseException() trong file machine.cc	17
Hình 2.2.4. Hàm ExceptionHandler (ban đầu)	18
Hình 2.3.1. Các exceptions cần xử lý	19
Hình 2.4.1. Viết lại cấu trúc điều khiển nhận các system call	20
Hình 2.5.1. Đoạn mã tăng Program Counter có sẵn	21
Hình 2.5.2. Định nghĩa các biến PPCReg, NextPCReg, PrePCReg	21
Hình 2.5.3. Hàm IncreasePC - tăng giá trị Program Counter	21
Hình 2.6.1. Chính file 'MakeFile.common'	22
Hình 2.6.2. Chính sửa file 'system.h' và 'system.cc'	23
Hình 2.6.3. Thêm đoạn mã khởi tạo và xoá đối tượng SynchConsole	23
Hình 2.6.4. Hàm User2System	24
Hình 2.6.5. Hàm System2User	25
Hình 2.6.6. Thêm định nghĩa system call code cho các syscalls mới	25
Hình 2.6.7. Thêm prototype cho syscalls mới	26
Hình 2.6.8. Chính sửa file 'start.s' và 'start.c'	27
Hình 2.7.1. Hàm ReadInt()	28
Hình 2.7.2. Hàm PrintInt()	29
Hình 2.7.3. Chương trình int.c	30
Hình 2.7.4. Hàm ReadChar()	31
Hình 2.7.5. Hàm PrintChar()	31
Hình 2.7.6. Chương trình char.c	32
Hình 2.7.7. Hàm ReadString()	33
Hình 2.7.8. Hàm PrintString()	34
Hình 2.7.9. Chương trình string.c	34
Hình 2.7.10. Thêm các syscall vào SyscallException	35
Hình 2.8.1. Mã nguồn chương trình help	36

Hình 2.8.2. Mã nguồn chương trình ascii	37
Hình 2.9.1. Thêm tên các chương trình ở dòng "all:..."	39
Hình 2.9.2. Thêm quy tắc biên dịch cho các chương trình (1)	39
Hình 2.9.3. Thêm quy tắc biên dịch cho các chương trình (2)	40
Hình 2.10.1. Demo1 ReadInt – PrintInt.....	40
Hình 2.10.2. Demo 2 ReadInt() - PrintInt()	41
Hình 2.10.3. Demo 1 ReadChar() - PrintChar().....	41
Hình 2.10.4. Demo 2 ReadChar() - PrintChar().....	42
Hình 2.10.5. Demo ReadString() - PrintString().....	42
Hình 2.10.6. Kết quả biên dịch chương trình help.....	43
Hình 2.10.7. Kết quả biên dịch chương trình ascii	43
Hình 2.10.8. Kết quả biên dịch chương trình sort.....	44

DANH MỤC BẢNG

Bảng 1. Danh sách thành viên.....	7
Bảng 2. Bảng phân công công việc.....	7
Bảng 3. Mức độ hoàn thành.....	8

1. TỔNG QUAN

1.1. THÔNG TIN NHÓM

STT	MSSV	Họ và Tên	Email
1	21120279	Lê Trần Minh Khuê	21120279@student.hcmus.edu.vn
2	21120289	Diệp Quốc Hoàng Nam	21120289@student.hcmus.edu.vn
3	21120290	Hoàng Trung Nam	21120290@student.hcmus.edu.vn
4	21120348	Lương Thanh Tú	21120348@student.hcmus.edu.vn
5	21120354	Nguyễn Trần Trình	21120354@student.hcmus.edu.vn

Bảng 1. Danh sách thành viên

1.2. BẢNG PHÂN CÔNG CÔNG VIỆC

STT	Mô tả công việc		Người thực hiện
1	Tìm hiểu mã chương trình NachOS đã được cung cấp.		Cả nhóm tìm hiểu.
2	Tìm hiểu thiết kế và cách HĐH làm việc.		Trình bày báo cáo: Minh Khuê
3	Lập trình Exceptions và System call		
	a	Viết lại file exception.cc.	
	b	Viết lại cấu trúc điều khiển chương trình để nhận các NachOS system calls.	
	c	Viết mã tăng giá trị Program counter.	Trung Nam, Hoàng Nam
	d	Cài đặt sytem call ‘int ReadInt()’	
	e	Cài đặt sytem call ‘void PrintInt(int number)’	
	f	Cài đặt sytem call ‘char ReadChar()’	
	g	Cài đặt sytem call ‘void PrintChar(char character)’	
	h	Cài đặt sytem call ‘void ReadString(char[] buffer, int length)’	
	i	Cài đặt sytem call ‘void PrintString(char[] buffer)’	
	j	Viết chương trình ‘help’	Trần Trình, Thanh Tú
	k	Viết chương trình ‘ascii’	
	l	Viết chương trình ‘sort’	
4	Kiểm thử chương trình		
	Viết hoàn thiện báo cáo		Minh Khuê

Bảng 2. Bảng phân công công việc

1.3. CÁC NỘI DUNG ĐÃ HOÀN THÀNH

Yêu cầu	Công việc	Tỉ lệ hoàn thành
1	Hiệu mã chương trình NachOS	100%
2	Hiệu thiết kế	100%
3	Exceptions và System call	100%
4	Viết báo cáo	100%

Bảng 3. Mức độ hoàn thành

1.4. TỔ CHỨC BÀI LÀM VÀ CÁC GHI CHÚ LẬP TRÌNH

1.4.1. Tổ chức bài làm

Chúng em bố trí bài làm thành các file như sau:

- Report: Chứa file báo cáo (Report.pdf – file này).
- Source: chứa source code chương trình. Gồm folder nachos được nén lại sau khi nhóm đã thực hiện cài đặt xong các yêu cầu đồ án.

1.4.2. Môi trường thực nghiệm:

Chương trình được cài đặt và chạy thử trên máy ảo sử dụng:

- Hệ điều hành: Linux – Ubuntu 14.04
- Trình biên dịch:
 - o **gcc (Ubuntu 4.8.4-2ubuntu1~14.04.4) 4.8.4**
 - o **g++ (Ubuntu 4.8.4-2ubuntu1~14.04.4) 4.8.4**

2. NỘI DUNG BÁO CÁO

2.1. HIỆU MÃ CHƯƠNG TRÌNH NACHOS

2.1.1. Giới thiệu về NachOS:

NachOS (Not Another Complete Heuristic Operating System) là một phần mềm giả lập hệ điều hành, do Thomas Anderson đề xuất và được dùng cho việc giảng dạy và nghiên cứu.

Ban đầu NachOS được viết bằng ngôn ngữ C++ cho kiến trúc MIPS, nó được chạy như một tiến trình người dùng trên một hệ điều hành. NachOS được thiết kế các tính năng để mô phỏng cho:

- Một CPU (theo kiến trúc MIPS, gồm thanh ghi, bộ nhớ, bộ xử lý,...)
- Một ổ cứng.
- Một bộ điều khiển ngắn, các exception, bộ đếm thời gian và một số tính năng cơ bản khác.

Mục tiêu của NachOS là giới thiệu cho sinh viên các khái niệm trong thiết kế và triển khai hệ điều hành thông qua việc cài đặt các chức năng quan trọng trong hệ thống NachOS.

2.1.2. Cài đặt:

Bước 0: Tải các file cần thiết cho việc cài đặt Ubuntu và NachOS

- Nhóm tiến hành cài đặt NachOS trên hệ điều hành Ubuntu phiên bản 14.04 (32-bit) ([file cài đặt](#))
- Sau đó tải các file được thầy cung cấp trên trang thông tin môn học. ([đường dẫn đến “Tài liệu project 2”](#))

Bước 1: Cài đặt **Ubuntu 14.04** trên máy ảo **VMware Workstation Pro** bằng file ISO vừa tải ở trên.

Bước 2: Tiến hành cài đặt gcc và g++ trên máy ảo (cài qua terminal) bằng 2 lệnh sau:

“**sudo apt-get install gcc**”

“**sudo apt-get install g++**”

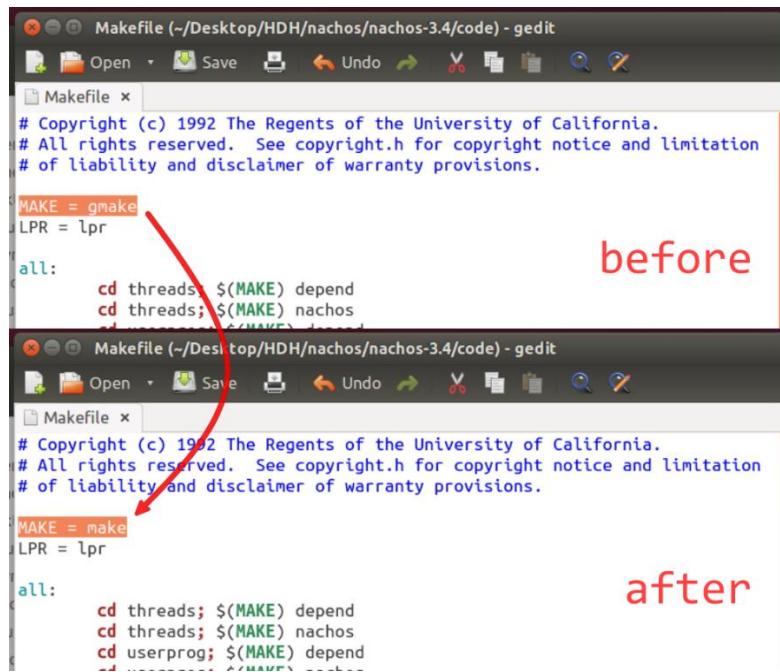
2.1.3. Biên dịch:

Biên dịch NachOS (phần chương trình đã được cung cấp)

Bước 1: Giải nén file nachos.zip đã tải về

Bước 2: Mở file “**Makefile**” trong thư mục “**./nachos/nachos/nachos-3.4/code**”

Bước 3: Sửa dòng “MAKE = gmake” thành “MAKE = make”



Hình 2.1.1. Sửa file 'Makefile'

Bước 4: Chuyển terminal đến thư mục code trong folder ‘nachos’ vừa giải nén, và tiến hành biên dịch NachOS bằng lệnh ‘make’

```
danie-nachos@ubuntu: ~/Desktop/HDH/nachos/nachos-3.4/code  
danie-nachos@ubuntu:~$ cd Desktop/HDH/nachos/nachos-3.4/code  
danie-nachos@ubuntu:~/Desktop/HDH/nachos/nachos-3.4/code$ make
```

Hình 2.1.2. Biên dịch NachOS bằng lệnh 'make'

Bước 5: Sau khi biên dịch, chuyển sang thư mục *threads* và gọi chạy thử chương trình *nachos* để xác nhận biên dịch thành công. Chương trình này sẽ gọi hàm *halt()* để tắt máy ảo NachOS.

```

danie-nachos@ubuntu: ~/Desktop/HDH/nachos/nachos-3.4/code/threads
danie-nachos@ubuntu:~/Desktop/HDH/nachos/nachos-3.4/code$ cd threads
danie-nachos@ubuntu:~/Desktop/HDH/nachos/nachos-3.4/code/threads$ ./nachos
No threads ready or runnable, and no pending interrupts.
Assuming the program completed.
Machine halting!

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

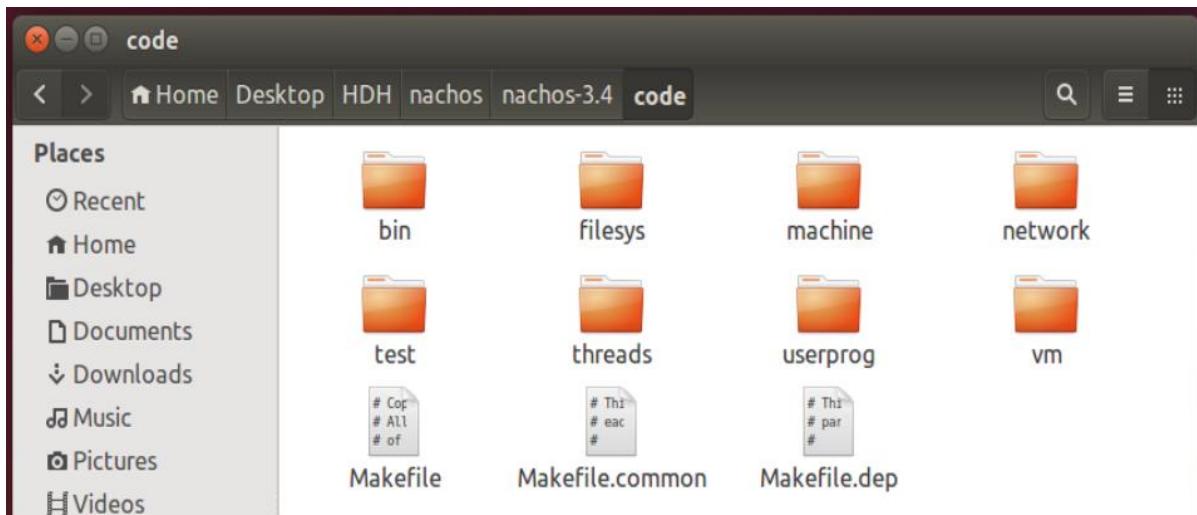
Cleaning up...
danie-nachos@ubuntu:~/Desktop/HDH/nachos/nachos-3.4/code/threads$ █

```

Hình 2.1.3. Chạy thử chương trình Halt trong folder threads

2.1.4. Các tệp thành phần:

2.1.4.1. Các tệp tin/ thư mục trong folder nachos-3.4



Hình 2.1.4. Các tệp tin/ thư mục trong folder nachos-3.4

- **bin:** chứa các đoạn mã phục vụ việc chuyển đổi định dạng và thực thi.
 - └ **coff2noff:** Trình biên dịch chuyển đổi từ định dạng COFF (Common Object File Format) sang định dạng NOFF (NachOS Object File Format). Định dạng NOFF là định dạng file đặc biệt được NachOS sử dụng.
 - └ **coff2noff.c:** Mã nguồn của chương trình coff2noff, được biên dịch thành coff2noff.
 - └ ...
- **filesystems:** Chức năng: Quản lý hệ thống tệp, bao gồm cả đọc và ghi vào đĩa.
 - └ **filesystems.cc, filesystems.h:** Mã nguồn chính của hệ thống tệp.

- └ **bitmap.***: Các thành phần liên quan đến bitmap.
 - └ **directory.***: Các thành phần liên quan đến quản lý thư mục.
 - └ **disk.***: Các thành phần liên quan đến quản lý đĩa.
 - └ **filehdr.***: Các thành phần liên quan đến header của tệp.
- **machine**: Chứa mã nguồn và tài nguyên liên quan đến **máy ảo** – giả lập các thành phần cơ bản của máy như thanh ghi, xử lý ngắn, quản lý đĩa, mạng, đồng hồ và một số thành phần khác.
 - └ **console.***: Quản lý đầu vào và đầu ra trên console.
 - └ **disk.***: Quản lý đọc và ghi dữ liệu lên đĩa.
 - └ **interrupt.***: Xử lý ngắn từ phần cứng.
 - └ **machine.***: Định nghĩa máy ảo và các thành phần liên quan.
 - └ **mipssim.***: Mô phỏng máy kiến trúc MIPS.
 - └ **network.***: Quản lý gửi và nhận dữ liệu qua mạng.
 - └ **stats.***: Thống kê và theo dõi trạng thái của hệ thống.
 - └ **timer.***: Quản lý và sử dụng timer.
- **network**: Quản lý các thành phần liên quan đến kết nối mạng, gửi nhận dữ liệu, và cài đặt giao thức mạng cơ bản, ...
 - └ **nettest.***: Chương trình kiểm thử mạng.
 - └ **post.***: Quản lý gửi và nhận dữ liệu qua mạng.
 - └ **README**: Tài liệu mô tả.
 - └ **sysdep.***: Phụ thuộc hệ thống.
- **test**: các đoạn mã cài đặt chương trình ở mức người dùng, thường để kiểm thử hệ thống
- **threads**: Quản lý luồng, hỗ trợ đa chương, đa luồng và có các thư viện hỗ trợ tương tác hệ thống.
- **userprog**: Quản lý các chương trình người dùng. Thường chứa các systemcalls để người dùng thực hiện gọi hệ thống.
- **vm**: Quản lý bộ nhớ ảo
- **Makefile, Makefile.common, Makefile.dep**: chứa các thông tin cấu hình và các quy tắc để biên dịch chương trình.

2.1.4.2. Các tập tin cần biết trong đồ án này:

- **progtest.cc**: kiểm tra các thủ tục để chạy chương trình người dùng
- **syscall.h**: Định nghĩa giao diện thủ tục hệ thống (system call interface) mà các chương trình người dùng có thể gọi để tương tác với kernel.
- **exception.cc**: Xử lý system call và các exception khác ở mức người dùng

- **bitmap.*:** các hàm xử lý cho lớp bitmap (thường dùng để theo dõi trạng thái của các bit bằng cách đánh dấu việc đang được sử dụng hay không của các tài nguyên như các ô nhớ vật lý, các block trên đĩa,...)
- **filesys.h:** định nghĩa các hàm trong hệ thống tệp NachOS. Thường chứa các hàm quản lý như mở, đóng, đọc, ghi, di chuyển con trỏ,...
- **openfile.h:** định nghĩa các hàm trong hệ thống file đang mở trong NachOS. Thường tập trung vào các file đang được mở.
- **translate.*:** Dùng để quản lý chuyển địa chỉ ảo sang địa chỉ vật lý và ngược lại. (Trong giới hạn đồ án này, giáo viên hướng dẫn giả sử mỗi địa chỉ ảo cũng giống như địa chỉ vật lý, mục đích giới hạn lại chỉ chạy 1 chương trình tại một thời điểm. Nhóm chúng em không thay đổi các file này.)
- **machine.*:** mô phỏng các thành phần của máy tính khi thực thi chương trình người dùng
- **mipssim.cc:** mô phỏng tập lệnh của MIP R2/3000 processor.
- **console.*:** chứa mã nguồn mô phỏng một thiết bị đầu cuối sử dụng các tệp tin của UNIX. Đặc điểm chung của thiết bị:
 - o Đơn vị dữ liệu theo byte: các hoạt động đọc và ghi sẽ thực hiện trên cấp độ byte.
 - o Đọc và ghi các bytes cùng một thời điểm: có khả năng đọc và ghi nhiều bytes cùng lúc, tối ưu hiệu xuất.
 - o Các bytes đến bất đồng bộ: khi chương trình yêu cầu đọc dữ liệu từ thiết bị, nó không bị chặn và có thể tiếp tục thực hiện các công việc khác trong khi đợi dữ liệu đến. Khi dữ liệu sẵn sàng, nó có thể được đọc mà không làm chậm chương trình chính.
- **synchconsole.*:** Nhóm hàm cho việc quản lý nhập xuất I/O theo dòng trong NachOS.
- **../test/*:** các chương trình C được biên dịch theo MIPS và chạy trong NachOS.

2.2. HIỆU THIẾT KẾ NACHOS

NachOS là một hệ điều hành nhỏ gọn, nhằm hỗ trợ việc tìm hiểu và xây dựng các thành phần của một hệ điều hành nên nó có một số đặc điểm riêng. Phần này nhóm sẽ trình bày về thiết kế và cách Hệ điều hành NachOS giao tiếp với các chương trình người dùng.

2.2.1. Thiết kế NachOS:

NachOS thực hiện mô hình kernel và user space để phân biệt giữa các chương trình người dùng và các tác vụ cơ bản của hệ điều hành, trong đó các tác vụ cơ bản như scheduler, quản lý bộ nhớ, quản lý tệp tin,... được cài đặt như những module độc lập. Các **system calls** được sử dụng để chuyển quyền điều khiển giữa user mode và kernel mode.

2.2.1.1. Phân biệt Kernel Space và User Space:

- **Kernel Space:**
 - + Có quyền truy cập và thực hiện hầu hết mọi tác vụ trong hệ thống.
 - + Quản lý và kiểm soát chặt chẽ tài nguyên, đảm bảo tính an toàn và ổn định của hệ thống.
 - + Cung cấp các system calls để chương trình người dùng tương tác với các tác vụ và tài nguyên hệ thống.
 - + Thường dành riêng cho các chức năng hoạt động ở cấp độ thấp nhất, giao tiếp gần như trực tiếp với phần cứng để thực hiện các nhiệm vụ quản lý cơ bản.
- **User Space:**
 - + Có quyền thấp hơn kernel và chỉ được thực thi những tác vụ được cho phép.
 - + Chỉ chứa và quản lý các mã thực thi và dữ liệu của người dùng thông thường.
 - + Cần dùng các system calls để yêu cầu xử lý từ kernel.
 - + Do việc giới hạn truy cập vào kernel space, các vấn đề hay lỗi xuất hiện trong user mode có thể được khắc phục một cách hiệu quả. Sự ổn định của hệ thống không bị ảnh hưởng nếu có sự cố xảy ra trong không gian địa chỉ của chính các tiến trình người dùng.

2.2.1.2. Giao tiếp giữa hệ điều hành và chương trình người dùng:

Trước khi các chương trình có thể thực thi được, nó phải được biên dịch. Đối với NachOS, tất cả các chương trình trong thư mục `./code/test` đều được biên dịch khi ta biên dịch NachOS. (Tuy nhiên, ta cũng có thể biên dịch các chương trình này mà không phải biên dịch lại toàn bộ NachOS.)

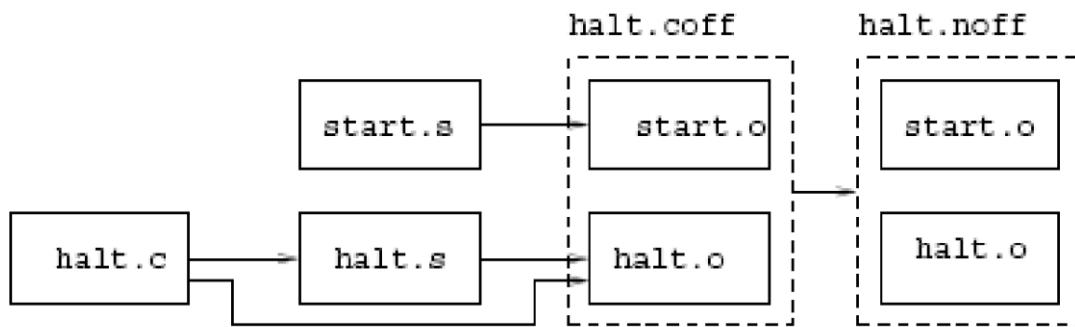
Mỗi chương trình trong hệ thống phải có các thông tin cục bộ của nó, bao gồm program counters, registers, stack pointers, và file system handler. Mặc dù user program truy cập các thông tin cục bộ của nó, nhưng HĐH điều khiển các truy cập này, HĐH đảm bảo các yêu cầu từ user program tới kernel không làm cho HĐH sụp đổ. Việc chuyển quyền điều khiển từ user mode thành system mode được thực hiện thông qua system calls, softwave interrupt/trap.

Trước khi gọi một lệnh trong hệ thống thì các tham số truyền vào cần thiết phải được nạp vào các thanh ghi của CPU. Để chuyển một biến mang giá trị, tiến trình chỉ việc ghi giá trị vào thanh ghi. Để chuyển một biến tham chiếu, thì giá trị lưu trong thanh ghi đc xem như là “user space pointer”. Bởi vì user space pointer không có ý nghĩa đối với kernel, mà chúng ta cần là chuyển nội dung từ user space vào kernel sao cho ta có thể xử lý dữ liệu này. Khi trả thông tin từ system về user space, thì các giá trị phải đặt trong các thanh ghi của CPU. {Tham khảo yêu cầu đồ án}

Quá trình biên dịch trên NachOS gồm 3 bước như sau (ví dụ biên dịch chương trình **halt.c**):

- **Bước 1:** Chương trình **halt.c** được cross-compiler biên dịch thành tập tin **halt.s** (là mã hợp ngữ, chạy trên kiến trúc MIPS)
- **Bước 2:** Tập tin **halt.s** này được liên kết với tập tin **start.s** để tạo thành tập tin ‘**halt.coff**’ (bao gồm **halt.o** và **start.o**) là định dạng thực thi trên hệ điều hành Linux cho kiến trúc MIPS.
- **Bước 3:** Tập tin **halt.coff** được phần mềm **coff2noff** (được cung cấp sẵn trong `./code/bin/`) chuyển thành tập tin **halt.noff** là dạng file thực thi trên NachOS với kiến trúc MIPS.

Quá trình trên được mô tả bằng hình sau:



Hình 2.2.1. Minh họa quá trình biên dịch file *halt.c*

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

- + Nhảy đến hàm **main()** trong file **halt.c** bằng lệnh “jal main”

```

_start:      ---- start.s
jal main
move $a,$0
jal Exit    /* if we return from main, exit(0) */
.end _start

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

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

// halt.s

```

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

- + Gặp system call là **Halt** nên nhảy đến hàm **Halt()** ("jal Halt")

```

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

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

// halt.s

```

10 gcc2_compiled.:
11 __gnu_compiled_c:
12 .text
13 .align 2
14 .globl main
15 .ent main
16 main:
17 .frame $fp,24,$31
18 .mask 0xc0000000,-4
19 .fmask 0x00000000,0
20 subu $sp,$sp,24
21 sw $31,20($sp)
22 sw $fp,16($sp)
23 move $fp,$sp
24 jal __main
25 jal Halt
26 $L1:
```

- + Ta thấy “**addiu \$2,\$0,SC_Halt**” nghĩa là gán mã của system call (trường hợp này là SC_Halt) vào thanh ghi **r2**.
- + “**syscall**” : gọi xử lý system call – trao quyền cho hệ thống. Cụ thể: Khi máy ảo nạp và giải mã từng lệnh, nhận thấy opCode của lệnh là loại OP_SYSCALL trong mipssim.cc, lúc này, hàm xử lý ngoại lệ được đẩy lên:

```

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

```

Hình 2.2.2. OP_SYSCALL trong mipssim.cc

Hàm RaiseException() được cài đặt trong file **machine.cc** sẽ trao quyền cho hệ thống:

```

//-----.
// Machine::RaiseException
// Transfer control to the Nachos kernel from user mode, because
// the user program either invoked a system call, or some exception
// occurred (such as the address translation failed).
//
// "which" -- the cause of the kernel trap
// "badVaddr" -- the virtual address causing the trap, if appropriate
//-----

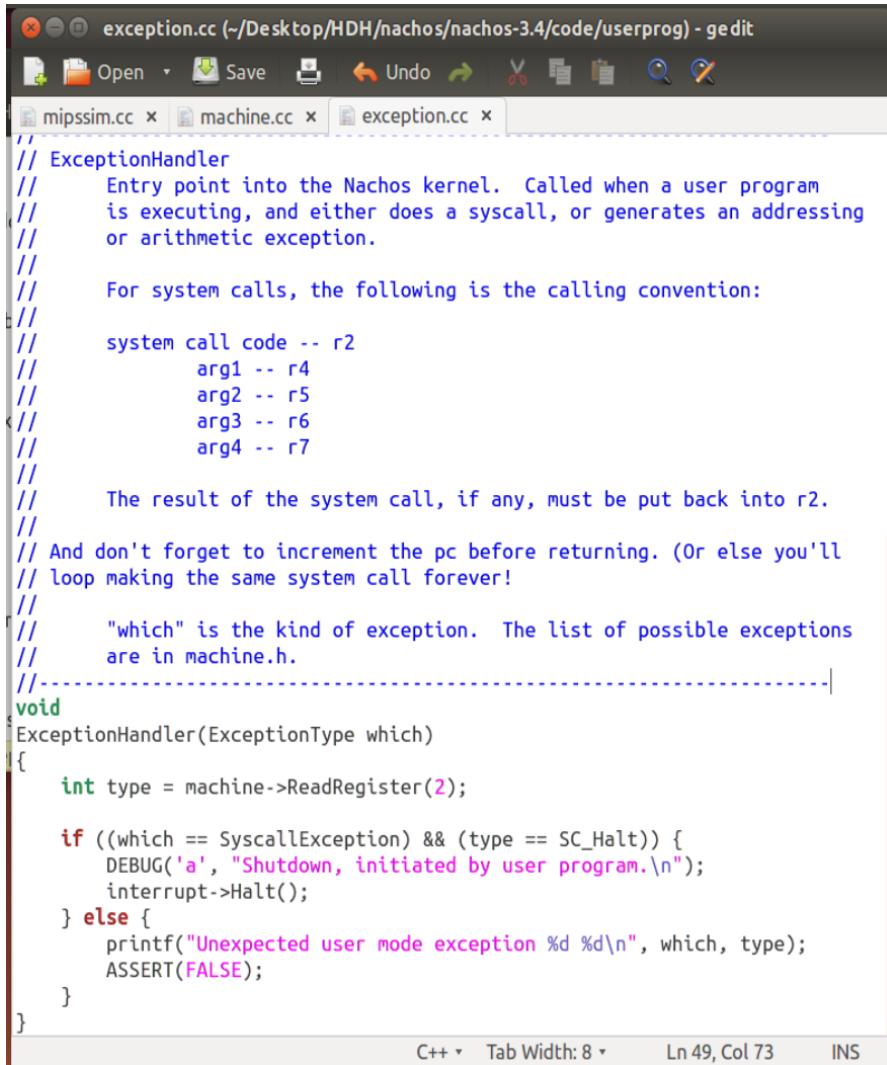
void
Machine::RaiseException(ExceptionType which, int badVAddr)
{
    DEBUG('m', "Exception: %s\n", exceptionNames[which]);

    // ASSERT(interrupt->getStatus() == UserMode);
    registers[BadVAddrReg] = badVAddr;
    DelayedLoad(0, 0);                                // finish anything in progress
    interrupt->setStatus(SystemMode);
    ExceptionHandler(which);                          // interrupts are enabled at this
    point
    interrupt->setStatus(UserMode);
}

```

Hình 2.2.3. RaiseException() trong file machine.cc

Lúc này, hệ điều hành kiểm tra phần xử lý của syscall này và thực hiện các chỉ thị:



```
// ExceptionHandler
//     Entry point into the Nachos kernel. Called when a user program
//     is executing, and either does a syscall, or generates an addressing
//     or arithmetic exception.
//
//     For system calls, the following is the calling convention:
//
//         system call code -- r2
//             arg1 -- r4
//             arg2 -- r5
//             arg3 -- r6
//             arg4 -- r7
//
//             The result of the system call, if any, must be put back into r2.
//
//             And don't forget to increment the pc before returning. (Or else you'll
//             loop making the same system call forever!
//
//             "which" is the kind of exception. The list of possible exceptions
//             are in machine.h.
//-----
void
ExceptionHandler(ExceptionType which)
{
    int type = machine->ReadRegister(2);

    if ((which == SyscallException) && (type == SC_Halt)) {
        DEBUG('a', "Shutdown, initiated by user program.\n");
        interrupt->Halt();
    } else {
        printf("Unexpected user mode exception %d %d\n", which, type);
        ASSERT(FALSE);
    }
}
```

Hình 2.2.4. Hàm ExceptionHandler (ban đầu)

- + Sau đó nhảy đến \$31, chuyển lại user mode, để tiếp tục chương trình

Lưu ý, sau khi thực hiện xong system call thì hệ điều hành sẽ trao quyền lại cho user, CPU thực hiện nạp lệnh tiếp theo. Cần phải tăng giá trị Program Counter để tránh việc CPU đọc lại chỉ thị cũ (đã thực hiện xong) làm dẫn đến vòng lặp vô tận.

2.3. CÀI ĐẶT XỬ LÝ CHO EXCEPTIONS

Các exception cần xử lý (được liệt kê trong machine/machine.h):

```

make
g++ -c machine.h
g++ -c exception.cc
g++ -c main.c

```

Hình 2.3.1. Các exceptions cần xử lý

- NoException: Mọi thứ ổn, trả quyền điều khiển về Hệ điều hành (1)
- SyscallException: Ngoại lệ phát sinh từ 1 chương trình gọi syscall. Sẽ được xử lý bởi các hàm chúng ta viết cho user system call, giá trị thanh ghi 2 là giá trị cụ thể của syscall đó. (2)
- PageFaultException: Lỗi trang.
- ReadOnlyException: Ghi vào vùng nhớ “chỉ đọc”.
- BusErrorException: Lỗi Bus, trỏ đến địa chỉ vật lý không hợp lệ.
- AddressErrorException: Lỗi địa chỉ không hợp lệ hoặc đã vượt quá không gian địa chỉ cho phép.
- OverflowException: Lỗi tràn số do phép cộng hoặc trừ.
- IllegalInstrException: Lỗi chỉ thị chưa được cài đặt hoặc chỉ thị đặc biệt dành riêng.
- NumExceptionTypes

Ngoại trừ exception (1) và (2) xử lý như đã trình bày, các exceptions còn lại chỉ in ra thông báo lỗi, và **halt** hệ thống.

⇒ Ta xử lý các exception bên trên bằng một cấu trúc switch case

2.4. VIẾT LẠI CẤU TRÚC ĐIỀU KHIỂN

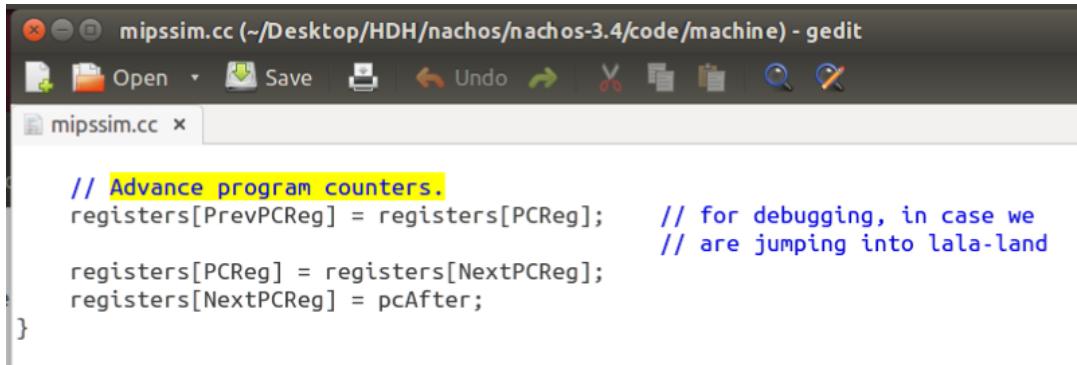
Thêm cấu trúc **switch case** bên trong hàm “ExceptionHandler()” (thuộc file **./nachos/nachos-3.4/code/userprog/exception.cc**) để xử lý các exception đã nêu ở trên. Cụ thể:

```
void ExceptionHandler(ExceptionType which)
{
    int type = machine->ReadRegister(2);
    switch(which)
    {
        case NoException:
            return;
            break;
        case PageFaultException:
            DEBUG('a', "The program will now exit due to exception: Page Fault\n");
            interrupt->Halt();
            break;
        case ReadOnlyException:
            DEBUG('a', "The program will now exit due to an attempt to write on Read-only files\n");
            interrupt->Halt();
            break;
        case BusErrorException:
            DEBUG('a', "The program will now exit due to error on bus\n");
            interrupt->Halt();
            break;
        case AddressErrorException:
            DEBUG('a', "The program will now exit due to address error\n");
            interrupt->Halt();
            break;
        case OverflowException:
            DEBUG('a', "The program will now exit due to overflow\n");
            interrupt->Halt();
            break;
        case IllegalInstrException:
            DEBUG('a', "The program will now exit due to illegal instruction\n");
            interrupt->Halt();
            break;
        case NumExceptionTypes:
            DEBUG('a', "The program will now exit due to exception: NumExceptionTypes\n");
            ASSERT(FALSE);
            interrupt->Halt();
            break;
        case SyscallException:
            switch (type)
            {
                ...
            }
            break;
        default:
            printf("Unexpected user mode exception %d %d\n",which,type);
            ASSERT(FALSE);
            break;
    }
}
```

Hình 2.4.1. Viết lại cấu trúc điều khiển nhận các system call

2.5. TÁCH HÀM TĂNG PROGRAM COUNTER:

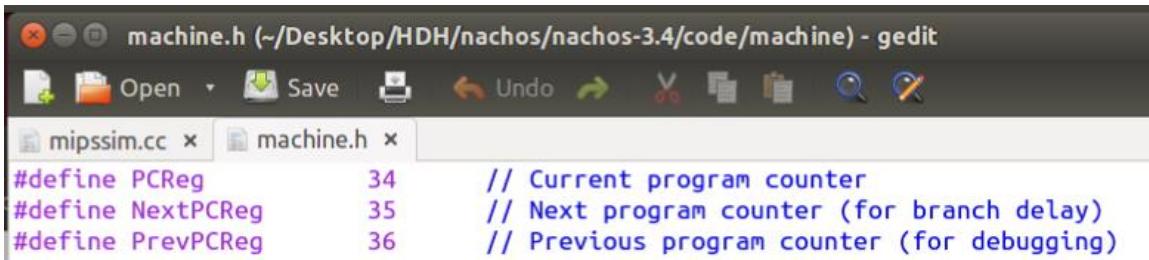
Nhóm em đã tìm thấy đoạn code tăng Program counter trong file “**mipssim.cc**” như sau:



```
// Advance program counters.
registers[PrevPCReg] = registers[PCReg];      // for debugging, in case we
                                                // are jumping into lala-land
registers[PCReg] = registers[NextPCReg];
registers[NextPCReg] = pcAfter;
}
```

Hình 2.5.1. Đoạn mã tăng Program Counter có sẵn

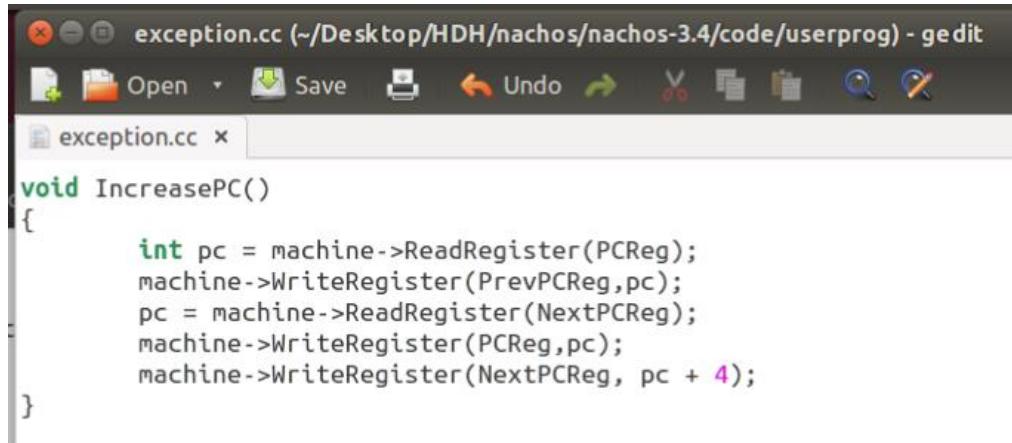
Với các biến PCReg, NextPCReg và PrePCReg được định nghĩa trong file **machine.h** tương ứng với các thanh ghi 34, 35, 36 như sau:



```
#define PCReg          34      // Current program counter
#define NextPCReg       35      // Next program counter (for branch delay)
#define PrevPCReg       36      // Previous program counter (for debugging)
```

Hình 2.5.2. Định nghĩa các biến PCReg, NextPCReg, PrePCReg

Chương trình của chúng em cần gọi khá nhiều syscall, vậy nên để thuận tiện cho việc cài đặt và sử dụng, chúng em viết một hàm để tăng program counter như sau:



```
void IncreasePC()
{
    int pc = machine->ReadRegister(PCReg);
    machine->WriteRegister(PrevPCReg,pc);
    pc = machine->ReadRegister(NextPCReg);
    machine->WriteRegister(PCReg,pc);
    machine->WriteRegister(NextPCReg, pc + 4);
}
```

Hình 2.5.3. Hàm IncreasePC - tăng giá trị Program Counter

Giải thích:

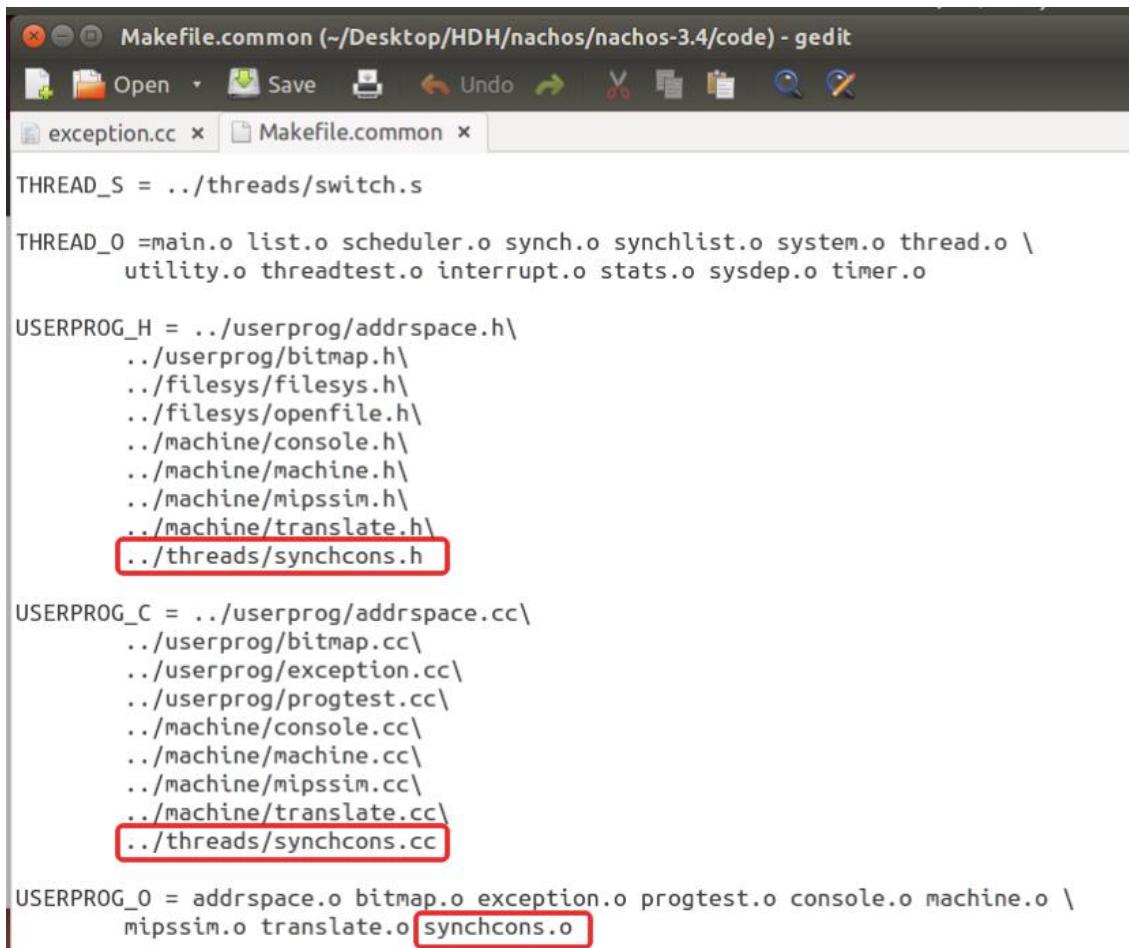
- Địa chỉ lệnh hiện tại trở thành địa chỉ cũ: Chuyển dữ liệu từ registers[PCReg] về registers[PrevPCReg]
- Địa chỉ lệnh tiếp theo trở thành địa chỉ hiện tại: Chuyển dữ liệu từ registers[NextPCReg] về registers[PCReg]
- Địa chỉ lệnh sau lệnh tiếp theo trở thành địa chỉ lệnh tiếp theo: Tăng giá trị registers[NextPCReg] lên 4 đơn vị (4 đơn vị vì NachOS sử dụng các thanh ghi lệnh có kích thước 4 bytes).

2.6. CÀI ĐẶT TRƯỚC KHI THÊM SYSTEM CALL

2.6.1. Thêm lớp SynchConsole

Bước 1: Tải file “synchcons.rar” về và giải nén. Sau đó copy 2 file synchcons.h và synchcons.cc vào thư mục “nachos/nachos-3.4/code/threads”.

Bước 2: Chỉnh file “MakeFile.common” trong folder “nachos/nachos-3.4/code”: Thêm các dòng sau và lưu lại



```

THREAD_S = ../threads/switch.s

THREAD_O = main.o list.o scheduler.o synch.o synchlist.o system.o thread.o \
           utility.o threadtest.o interrupt.o stats.o sysdep.o timer.o

USERPROG_H = ../userprog/addrspace.h\
            ../userprog/bitmap.h\
            ../filesys/filesys.h\
            ../filesys/openfile.h\
            ../machine/console.h\
            ../machine/machine.h\
            ../machine/mipssim.h\
            ../machine/translate.h\
            ../threads/synchcons.h

USERPROG_C = ../userprog/addrspace.cc\
            ../userprog/bitmap.cc\
            ../userprog/exception.cc\
            ../userprog/progtest.cc\
            ../machine/console.cc\
            ../machine/machine.cc\
            ../machine/mipssim.cc\
            ../machine/translate.cc\
            ../threads/synchcons.cc

USERPROG_O = addrspace.o bitmap.o exception.o progtest.o console.o machine.o \
            mipssim.o translate.o synchcons.o

```

Hình 2.6.1. Chỉnh file 'MakeFile.common'

Bước 3: Gọi lệnh “make” (như đã trình bày ở bước 4 – phần 2.3.1) để biên dịch lại NachOS.

Bước 4: Chỉnh sửa file system.h và system.cc (thêm các dòng được đóng khung)

```
system.h (~/Desktop/HDH/nachos/nachos-3.4/code/threads) - gedit
system.h x
extern Timer *timer; // the hardware alarm clock

#ifndef USER_PROGRAM
#include "machine.h"
#include "synchcons.h"
extern Machine* machine; // user program memory and registers
// students' applied change
extern SynchConsole *synchcons; // the console synchronizer
#endif

system.cc (~/Desktop/HDH/nachos/nachos-3.4/code/threads) - gedit
system.h x system.cc x
#endif

#ifndef USER_PROGRAM // requires either FILESYS or FILESYS_STUB
Machine *machine; // user program memory and registers
// students' applied change
SynchConsole *synchcons; // for console synchronizer
#endif
```

Hình 2.6.2. Chỉnh sửa file 'system.h' và 'system.cc'

Bước 5: Thêm đoạn khởi tạo và xoá synchConsole vào system.c

```
system.cc (~/Desktop/HDH/nachos/nachos-3.4/code/threads) - gedit
system.h x system.cc x
#endif

#ifndef USER_PROGRAM
    machine = new Machine(debugUserProg); // this must come first
    synchcons = new SynchConsole(); // STUDENT: initiate synchconsole
for syscall usage.
#endif

system.cc (~/Desktop/HDH/nachos/nachos-3.4/code/threads) - gedit
system.h x system.cc x
#endif

#ifndef NETWORK
    delete postOffice;
#endif

#ifndef USER_PROGRAM
    delete machine;
    delete synchcons; // delete synchcons instance to avoid memory leaks.
#endif
```

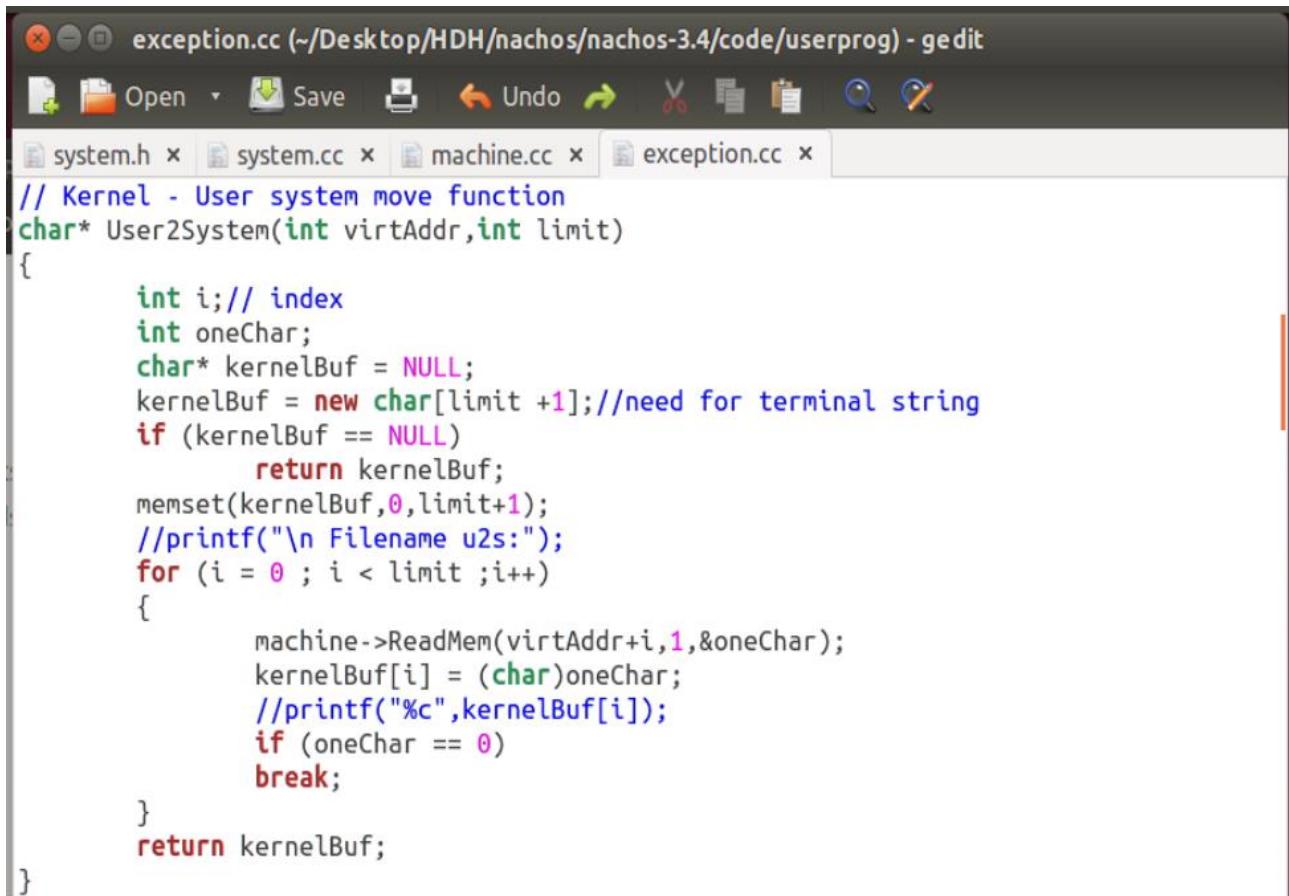
Hình 2.6.3. Thêm đoạn mã khởi tạo và xoá đối tượng SynchConsole

2.6.2. Thêm thủ tục User2System và System2User

Thêm 2 thủ tục User2System và System2User để hỗ trợ truyền dữ liệu giữa User Address Space và Kernel Address Space.

2.6.2.1. User2System:

Chuyển dữ liệu từ user space sang kernel space. Hàm này được dùng khi hệ thống cần đọc dữ liệu từ vùng nhớ của người dùng (ví dụ như đọc dữ liệu từ bộ đệm khi người dùng nhập vào).



```
exception.cc (~/Desktop/HDH/nachos/nachos-3.4/code/userprog) - gedit
Open Save Undo Redo Cut Copy Paste Find Replace
system.h x system.cc x machine.cc x exception.cc x
// Kernel - User system move function
char* User2System(int virtAddr,int limit)
{
    int i;// index
    int oneChar;
    char* kernelBuf = NULL;
    kernelBuf = new char[limit +1];//need for terminal string
    if (kernelBuf == NULL)
        return kernelBuf;
    memset(kernelBuf,0,limit+1);
    //printf("\n Filename u2s:");
    for (i = 0 ; i < limit ;i++)
    {
        machine->ReadMem(virtAddr+i,1,&oneChar);
        kernelBuf[i] = (char)oneChar;
        //printf("%c",kernelBuf[i]);
        if (oneChar == 0)
            break;
    }
    return kernelBuf;
}
```

Hình 2.6.4. Hàm User2System

2.6.2.2. System2User

Chuyển dữ liệu từ kernel space sang user space. Hàm này được dùng khi hệ thống cần gửi dữ liệu từ hệ thống đến vùng nhớ của người dùng (ví dụ như trả kết quả cho một chương trình người dùng).

```

int System2User(int virtAddr,int len,char* buffer)
{
    if (len < 0) return -1;
    if (len == 0) return len;
    int i = 0;
    int oneChar = 0 ;
    do{
        oneChar= (int) buffer[i];
        machine->WriteMem(virtAddr+i,1,oneChar);
        i++;
    }while(i < len && oneChar != 0);
    machine->WriteRegister(2,0);
    return i;
}

```

Hình 2.6.5. Hàm System2User

2.6.3. Các bước để cài đặt các system call mới.

Trong phạm vi đồ án này, chúng em sẽ cài đặt 6 system calls tương ứng với 6 yêu cầu đã được nêu trong đề.

Bước 1: Thêm các dòng định nghĩa system call code cho các syscall mới trong file “./nachos/nachos-3.4/code/userprog/syscall.h”

```

#ifndef SYSCALLS_H
#define SYSCALLS_H

#include "copyright.h"

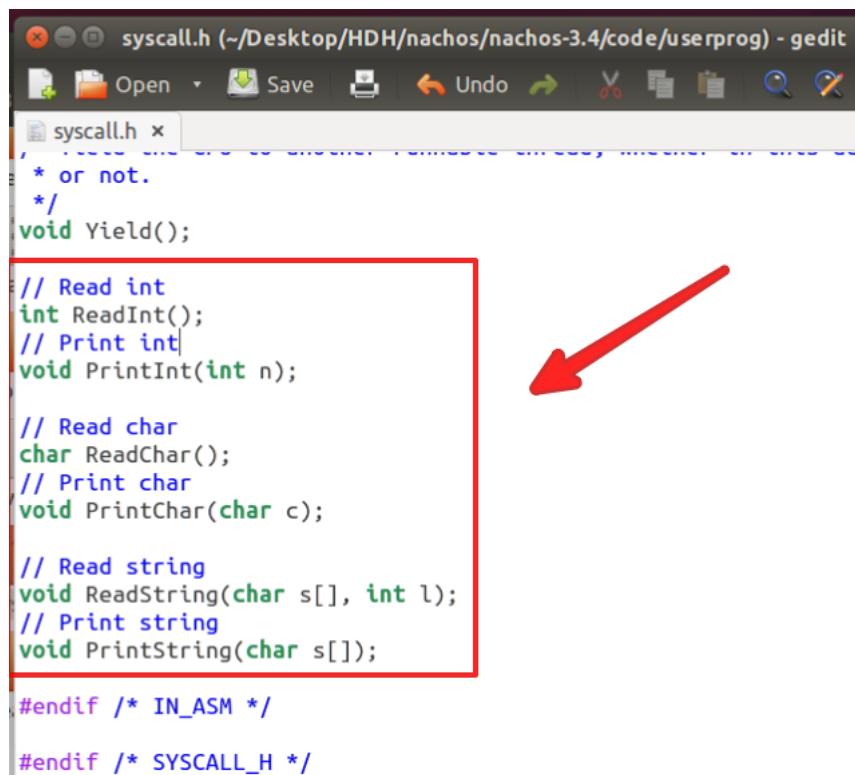
/* system call codes -- used by the stubs to tell the kernel which system call
 * is being asked for
 */
#define SC_Halt      0
#define SC_Exit      1
#define SC_Exec      2
#define SC_Join      3
#define SC_Create    4
#define SC_Open       5
#define SC_Read      6
#define SC_Write     7
#define SC_Close     8
#define SC_Fork      9
#define SC_Yield    10

#define SC_ReadInt   11
#define SC_PrintInt  12
#define SC_ReadChar  13
#define SC_PrintChar 14
#define SC_ReadString 15
#define SC_PrintString 16

```

Hình 2.6.6. Thêm định nghĩa system call code cho các syscalls mới

Bước 2: Khai báo các prototype tương ứng với các syscall ở trên:



```
/* or not.
*/
void Yield();

// Read int
int ReadInt();
// Print int
void PrintInt(int n);

// Read char
char ReadChar();
// Print char
void PrintChar(char c);

// Read string
void ReadString(char s[], int l);
// Print string
void PrintString(char s[]);

#endif /* IN_ASM */

#endif /* SYSCALL_H */
```

Hình 2.6.7. Thêm prototype cho syscalls mới

Bước 3: Thêm đoạn sau vào file “./nachos/nachos-3.4/code/test/start.s” và “./nachos/nachos-3.4/code/test/start.c” tương ứng với các hàm cần viết syscall.

Lưu ý: Viết bằng Assembly. (Một số lý do chủ yếu là vì trong file start, một số công việc quan trọng của hệ thống được khởi tạo, cấu hình, bên cạnh đó, các xử lý exception và systemcall cũng được điều khiển ở đây)

```

start.s (~/Desktop/HDH/nachos/nachos-3.4/code/test) - gedit
start.c x start.s x
    addiu $2,$0,SC_Yield
    syscall
    j      $31
    .end Yield

    .globl ReadInt
    .ent   ReadInt
ReadInt:
    addiu $2,$0,SC_ReadInt
    syscall
    j      $31
    .end ReadInt

    .globl PrintInt
    .ent   PrintInt
PrintInt:
    addiu $2,$0,SC_PrintInt
    syscall
    j      $31
    .end PrintInt

    .globl ReadChar
    .ent   ReadChar
ReadChar:
    addiu $2,$0,SC_ReadChar
    syscall
    j      $31
    .end ReadChar

    .globl PrintChar
    .ent   PrintChar
PrintChar:
    addiu $2,$0,SC_PrintChar
    syscall
    j      $31
    .end PrintChar

    .globl ReadString
    .ent   ReadString
ReadString:
    addiu $2,$0,SC_ReadString
    syscall
    j      $31
    .end ReadString

    .globl PrintString
    .ent   PrintString
PrintString:
    addiu $2,$0,SC_PrintString
    syscall
    j      $31
    .end PrintString

```

ReadInt
PrintInt

ReadChar
PrintChar

ReadString
PrintString

Hình 2.6.8. Chỉnh sửa file 'start.s' và 'start.c'

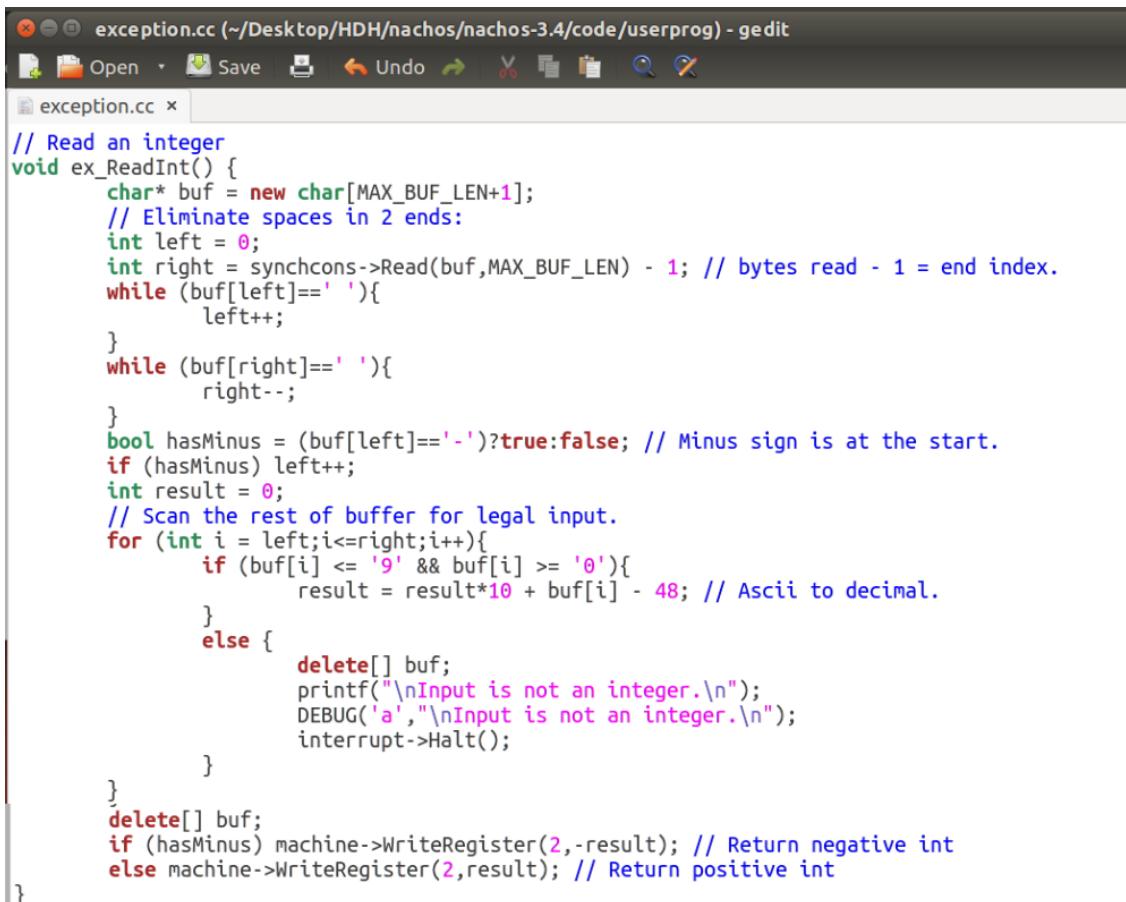
2.7. SYSTEM CALLS

Sau khi hoàn thành các bước trên, nhóm em bắt đầu viết các hàm được yêu cầu vào file “./nachos/nachos-3.4/code/userprog/exception.cc”

2.7.1. Xử lý số nguyên - Int

2.7.1.1. Nhập vào số nguyên - ‘int ReadInt()’

- Chức năng: **đọc** vào một số nguyên từ console.
- Mô tả: Vì khi người dùng nhập vào và chương trình nhận thông tin đều dưới dạng ký tự (char), nên sau khi nhận giá trị, nhóm tiến hành kiểm tra có phải số âm hay không, sau đó ép kiểu về dạng số (bằng cách chuyển đổi từ mã ascii sang số hệ thập phân).
- Cụ thể:
 - + Tạo mảng char* buf có kích thước MAX_BUF_LEN (255).
 - + Đọc mảng char vừa được nhập và gán vào mảng buf.
 - + Loại bỏ các khoảng trắng thừa ở đầu và cuối chuỗi.
 - + Kiểm tra số âm (bắt đầu bằng ‘-’)
 - + Chuyển từ char sang int (dựa vào bảng mã ascii). Nếu không thể chuyển, thông báo ra màn hình và **halt chương trình**.
 - + Trả về giá trị đã chuyển đổi được (bao gồm trường hợp số âm).



```
// Read an integer
void ex_ReadInt() {
    char* buf = new char[MAX_BUF_LEN+1];
    // Eliminate spaces in 2 ends:
    int left = 0;
    int right = synchcons->Read(buf,MAX_BUF_LEN) - 1; // bytes read - 1 = end index.
    while (buf[left]==' '){
        left++;
    }
    while (buf[right]==' '){
        right--;
    }
    bool hasMinus = (buf[left]=='-')?true:false; // Minus sign is at the start.
    if (hasMinus) left++;
    int result = 0;
    // Scan the rest of buffer for legal input.
    for (int i = left;i<=right;i++){
        if (buf[i] <= '9' && buf[i] >= '0'){
            result = result*10 + buf[i] - 48; // Ascii to decimal.
        }
        else {
            delete[] buf;
            printf("\nInput is not an integer.\n");
            DEBUG('a','\nInput is not an integer.\n');
            interrupt->Halt();
        }
    }
    delete[] buf;
    if (hasMinus) machine->WriteRegister(2,-result); // Return negative int
    else machine->WriteRegister(2,result); // Return positive int
}
```

Hình 2.7.1. Hàm ReadInt()

2.7.1.2. Xuất một số nguyên - ‘void PrintInt(int number)’

- Chức năng: **in** một số nguyên ra console.
- Giải thích: Vì hàm synchConsole->Write chỉ nhận tham số kiểu char, nên trước khi in cần chuyển về kiểu char.
- Cụ thể:
 - + Nhận vào biến n có kiểu dữ liệu integer đọc từ thanh ghi 4.
 - + Tạo mảng buf để lưu giá trị chuyển đổi
 - + Kiểm tra là số âm hay dương
 - + Tạo biến len => lưu trữ số lượng phần tử trong mảng char buf
 - + Thực hiện vòng lặp để chuyển n từ số nguyên thành một mảng char (buf)
 - + In ra màn hình (xét cả trường hợp là số dương hay số âm)

```
exception.cc (~/Desktop/HDH/nachos/nachos-3.4/code/userprog) - gedit
exception.cc x

void ex_PrintInt() {
    int n = machine->ReadRegister(4);
    char* buf = new char[MAX_BUF_LEN+1];
    // Int to string converter.

    bool isNeg = false;
    if (n<0) {
        isNeg = true;
        n=-n;
    }

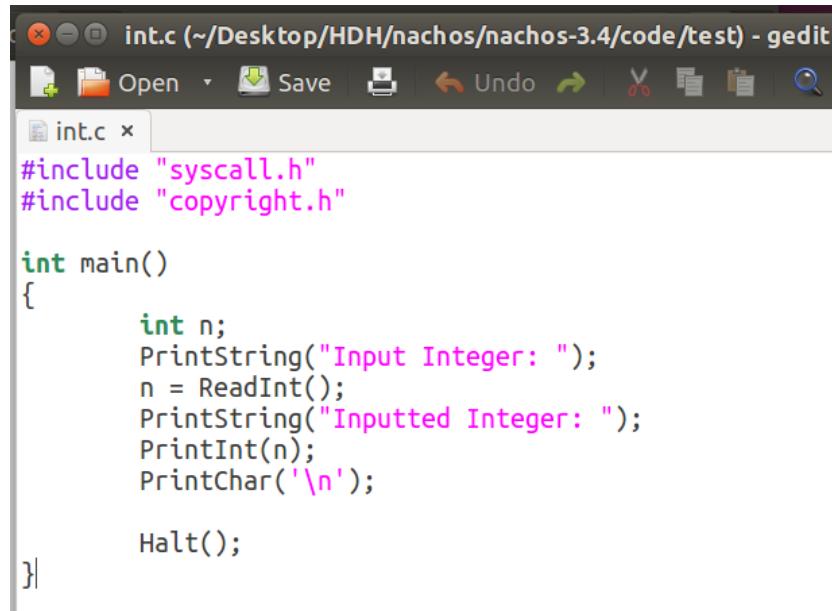
    int temp = n, len = 0;
    // find int length
    do {
        temp/=10;
        len++;
    } while (temp != 0);
    // convert to char

    int start = 0;
    if (isNeg) {
        buf[0] = '-';
        len++;
        start++;
    }
    temp = len;
    //buf[len] = 0;
    while (start < temp--){
        buf[temp] = (n%10)+48;
        n/=10;
    }

    //synchcons writes to console
    synchcons->Write(buf,len);
    delete[]buf;
}
```

Hình 2.7.2. Hàm PrintInt()

2.7.1.3. Chương trình kiểm thử ReadInt() – PrintInt()



```
#include "syscall.h"
#include "copyright.h"

int main()
{
    int n;
    PrintString("Input Integer: ");
    n = ReadInt();
    PrintString("Inputted Integer: ");
    PrintInt(n);
    PrintChar('\n');

    Halt();
}
```

Hình 2.7.3. Chương trình int.c

2.7.2. Xử lý ký tự - Char

2.7.2.1. Đọc một ký tự - ‘char ReadChar()’

- Chức năng: **đọc** một chuỗi từ console.
- Mô tả: Gán chuỗi được nhập trên màn hình thành một mảng kiểu char
- Cụ thể:
 - + Nhận vào mảng char từ console (through qua hàm synchcons)
 - + Dùng biến numBytes để kiểm tra số ký tự nhập vào.
 - + Xử lý: nếu nhập vào >1 hoặc không nhập ký tự nào => in thông báo ra màn hình và gán 0.
 - + Ngược lại, gán ký tự cho biến char c
 - + Ghi biến c vào thanh ghi 2
 - + Giải phóng mảng buf.

```
// Read a character.
void ex_ReadChar() {
    char* buf = new char[MAX_BUF_LEN+1];
    int numBytes = synchcons->Read(buf,MAX_BUF_LEN); //Read input into buffer;

    if (numBytes>1) // buf has more than 1 character
    {
        printf("Only 1 character can be entered.\n");
        DEBUG('a',"Only 1 character can be entered.\n");
        machine->WriteRegister(2,0);
    }
    else if (numBytes==0) // buf has less than 1 character
    {
        printf("Empty.\n");
        DEBUG('a',"Empty.\n");
        machine->WriteRegister(2,0);
    }
    else
    {
        char c = buf[0];
        machine->WriteRegister(2,c); //Successful input -> register syscall code
    }
    delete[] buf;
}
```

Hình 2.7.4. Hàm ReadChar()

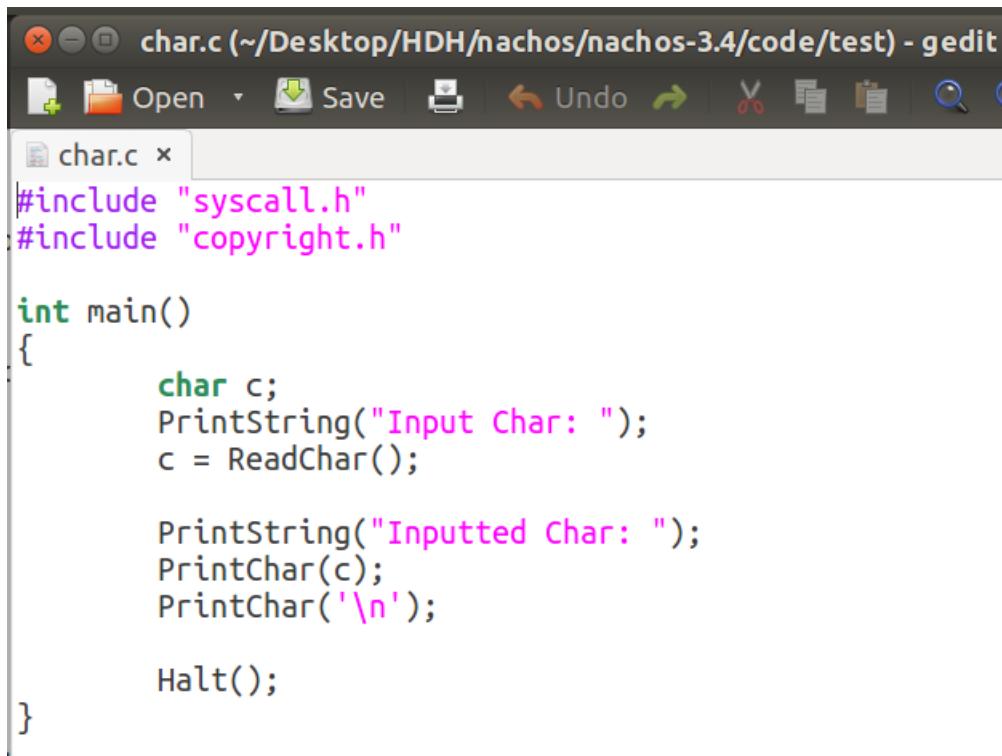
2.7.2.2. Xuất một ký tự - ‘void PrintChar()’

- Chức năng: **in** ký tự ra màn hình console
- Mô tả: Sử dụng synchcons->Write() của lopspw SynchConsole để in ra một ký tự và cũng là output của system call.

```
// Print a character.
void ex_PrintChar() {
    // just print.
    char c = machine->ReadRegister(4);
    synchcons->Write(&c,1);
}
```

Hình 2.7.5. Hàm PrintChar()

2.7.2.3. Chương trình char.c kiểm thử ReadChar() – PrintChar()



```
#include "syscall.h"
#include "copyright.h"

int main()
{
    char c;
    PrintString("Input Char: ");
    c = ReadChar();

    PrintString("Inputted Char: ");
    PrintChar(c);
    PrintChar('\n');

    Halt();
}
```

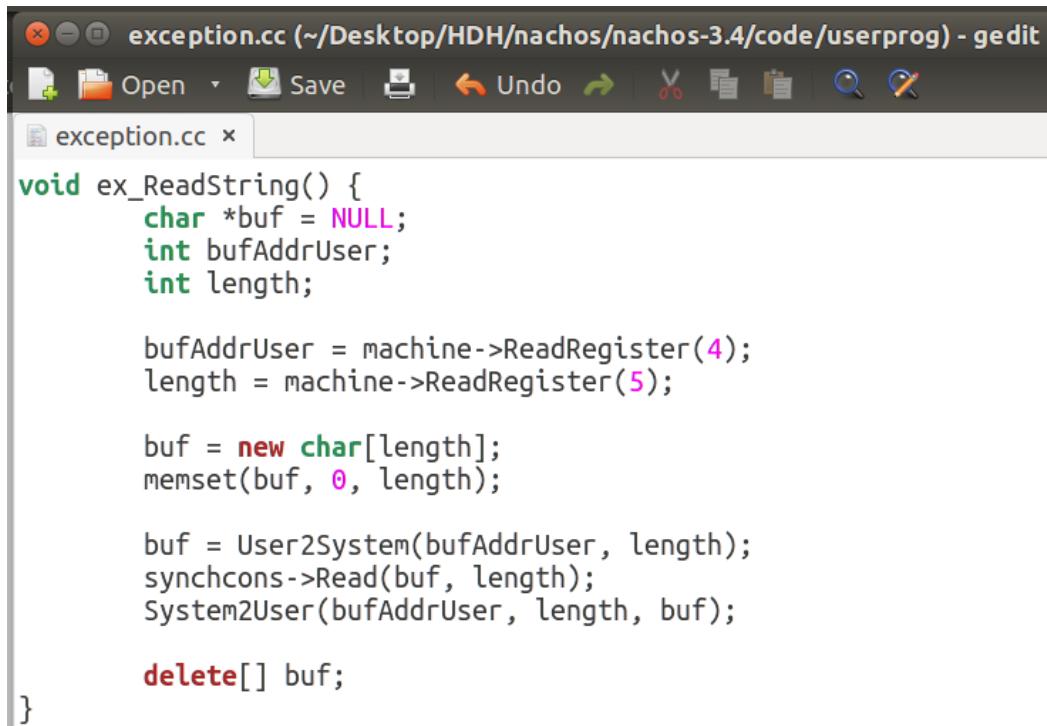
Hình 2.7.6. Chương trình char.c

2.7.3. Xử lý chuỗi - String

2.7.3.1. Đọc một chuỗi ký tự - ‘void ReadString()’

- Chức năng: **đọc** vào một string.
- Mô tả:
 - + char *buf = NULL;: Khai báo con trỏ buf kiểu char và khởi tạo giá trị ban đầu là NULL. Con trỏ này sẽ được sử dụng để chứa chuỗi ký tự.
 - + int bufAddrUser;: Khai báo biến bufAddrUser kiểu int để lưu địa chỉ bắt đầu của chuỗi ký tự trong UserAddressSpace .
 - + int length;: Khai báo biến length kiểu int để lưu độ dài của chuỗi ký tự cần đọc.
 - + bufAddrUser = machine->ReadRegister(4): Đọc địa chỉ của chuỗi ký tự từ thanh ghi số 4
 - + length = machine->ReadRegister(5);: Đọc độ dài của chuỗi ký tự từ thanh ghi số 5
 - + buf = new char[length];: Cấp phát bộ nhớ đủ để chứa chuỗi ký tự với độ dài là length. Bộ nhớ này được sử dụng để chứa dữ liệu trước khi được chuyển từ UserAddressSpace sang syystem/kernel space.

- + `memset(buf, 0, length);`: Sử dụng hàm `memset` để khởi tạo toàn bộ bộ nhớ mới cấp phát (`buf`) thành giá trị 0, đảm bảo rằng chuỗi ký tự sẽ được xử lý đúng, đặc biệt là khi kết thúc bằng `\0`.
- + `buf = User2System(bufAddrUser, length);`: Gọi hàm `User2System` để sao chép dữ liệu từ user space (tại địa chỉ `bufAddrUser`) sang system space (`buf`).
- + `synchcons->Read(buf, length);`: Sử dụng đối tượng `synchcons` để đọc `length` ký tự từ console và lưu vào bộ nhớ đã cấp phát (`buf`).
- + `System2User(bufAddrUser, length, buf);`: Gọi hàm `System2User` để sao chép dữ liệu từ system space (`buf`) sang user space (tại địa chỉ `bufAddrUser`).
- + Giải Phóng `buf`;



```

exception.cc (~/Desktop/HDH/nachos/nachos-3.4/code/userprog) - gedit
exception.cc x

void ex_ReadString() {
    char *buf = NULL;
    int bufAddrUser;
    int length;

    bufAddrUser = machine->ReadRegister(4);
    length = machine->ReadRegister(5);

    buf = new char[length];
    memset(buf, 0, length);

    buf = User2System(bufAddrUser, length);
    synchcons->Read(buf, length);
    System2User(bufAddrUser, length, buf);

    delete[] buf;
}

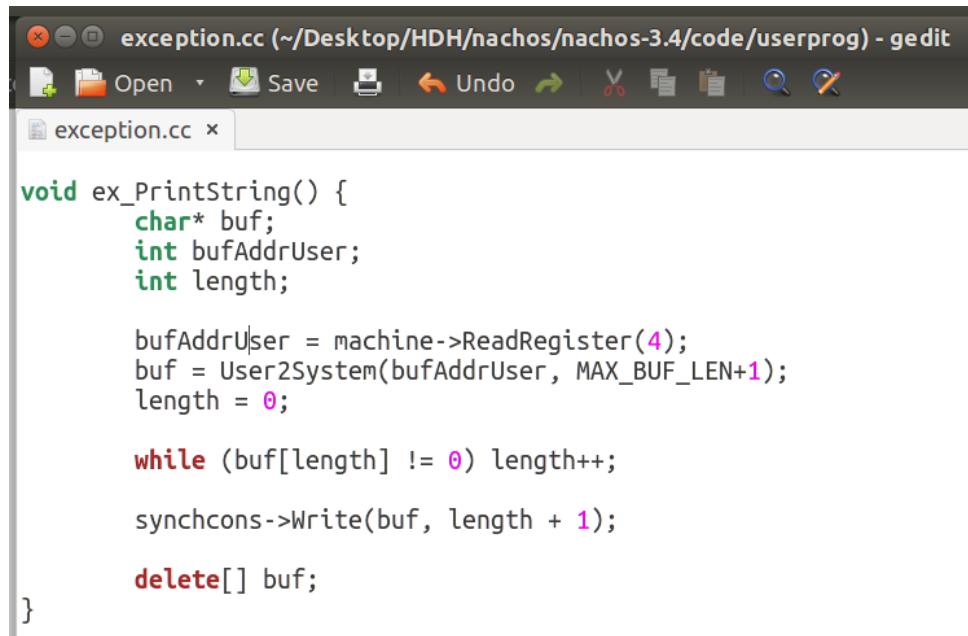
```

Hình 2.7.7. Hàm `ReadString()`

2.7.3.2. Xuất một chuỗi ký tự - ‘void PrintString()’

- Chức năng: **in** một string ra console.
- Mô tả:
 - + `buf; bufAddrUser; length` có ý nghĩa tương tự như trong hàm `ReadString`.
 - + `bufAddrUser = machine->ReadRegister(4);`: Đọc địa chỉ của chuỗi ký tự từ thanh ghi số 4
 - + `buf = User2System(bufAddrUser, MAX_BUF_LEN+1);`: Gọi hàm `User2System` để sao chép dữ liệu từ user space (tại địa chỉ `bufAddrUser`) sang system space (`buf`).

- + Tính độ dài thật sự của string (lưu vào length).
- + In ra màn hình bằng hàm synchcons->Write(buf,length+1)
- + Giải phóng buf.



```

void ex_PrintString() {
    char* buf;
    int bufAddrUser;
    int length;

    bufAddrUser = machine->ReadRegister(4);
    buf = User2System(bufAddrUser, MAX_BUF_LEN+1);
    length = 0;

    while (buf[length] != 0) length++;

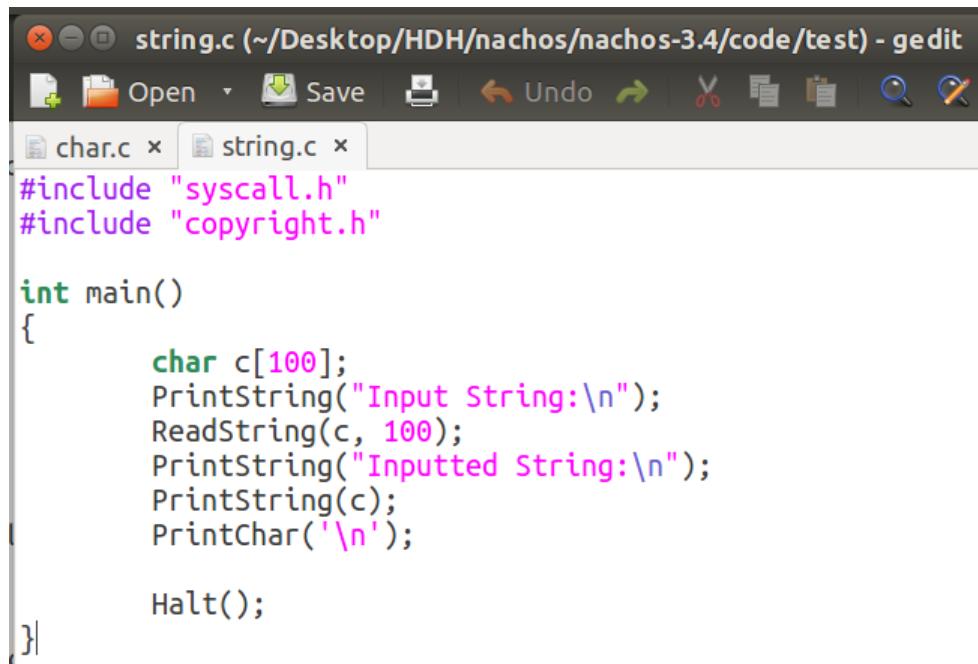
    synchcons->Write(buf, length + 1);

    delete[] buf;
}

```

Hình 2.7.8. Hàm PrintString()

2.7.3.3. Chương trình kiểm thử ReadString() – PrintString()



```

#include "syscall.h"
#include "copyright.h"

int main()
{
    char c[100];
    PrintString("Input String:\n");
    ReadString(c, 100);
    PrintString("Inputted String:\n");
    PrintString(c);
    PrintChar('\n');

    Halt();
}

```

Hình 2.7.9. Chương trình string.c

2.7.4. Thêm các Syscall cho các hàm trên vào syscallException

```
exception.cc (~/Desktop/HDH/nachos/nachos-3.4/code/userprog) - gedit
File Open Save Undo Redo Cut Copy Paste Find Replace
char.c x string.c x exception.cc x

case SyscallException:
    switch (type)
    {
        case SC_Halt:
            DEBUG('a', "Shutdown, initiated by user program.\n");
            interrupt->Halt();
            break;

        case SC_ReadInt:
            ex_ReadInt();
            IncreasePC();
            break;

        case SC_PrintInt:
            ex_PrintInt();
            IncreasePC();
            break;

        case SC_ReadChar:
            ex_ReadChar();
            IncreasePC();
            break;

        case SC_PrintChar:
            ex_PrintChar();
            IncreasePC();
            break;

        case SC_ReadString:
            ex_ReadString();
            IncreasePC();
            break;

        case SC_PrintString:
            ex_PrintString();
            IncreasePC();
            break;

        default:
            ASSERT(FALSE);
            interrupt->Halt();
            break;
    }
}
```

Hình 2.7.10. Thêm các syscall vào SyscallException

2.8. VIẾT CÁC CHƯƠNG TRÌNH

2.8.1. Chương trình ‘help’

- Chức năng: In ra các dòng giới thiệu cơ bản về nhóm và mô tả ngắn tắt về chương trình **sort** và **ascii**.
- Mô tả: Chương trình đơn thuần ứng dụng gọi hàm PrintString() đã cài đặt trước đó để in ra các thông tin cần thiết.

```
#include "syscall.h"
#include "copyright.h"

int main()
{
    PrintString("Nhóm gồm 5 thành viên:\n");
    PrintString("21120279 - Lê Trần Minh Khue\n");
    PrintString("21120289 - Diep Quốc Hoàng Nam\n");
    PrintString("21120290 - Hoàng Trung Nam\n");
    PrintString("21120348 - Nguyễn Trần Trinh\n");
    PrintString("21120354 - Lương Thành Tu\n");

    PrintString("Giới thiệu số chương trình:\n");
    PrintString("1. ascii: In ra bảng mã ascii\n");
    PrintString("2. sort: Nhập n và số n phần tử của mảng,\n");
    PrintString("sắp xếp sau đó in ra mảng đã sắp xếp\n");

    Halt();
}
```

Hình 2.8.1. Mã nguồn chương trình help

2.8.2. Chương trình ‘ascii’

- Chức năng: in ra các ký tự trong bảng mã ascii có giá trị thập phân từ 32 – 126 (do các giá trị khác là các ký tự điều khiển)
- Mô tả: Sử dụng 2 hàm đã cài đặt là PrintInt và PrintChar để in. Mỗi ký tự được trình bày theo mẫu <mã>: <ký tự>

The screenshot shows a Gedit window titled "ascii.c (~/Desktop/HDH/nachos/nachos-3.4/code/test) - gedit". The file contains C code that prints ASCII codes from 32 to 127. The code includes header files "syscall.h" and "copyright.h", and defines a main function that prints each character followed by its ASCII value and a colon. It also handles control codes and ends with a newline and a halt.

```
#include "syscall.h"
#include "copyright.h"

int main()
{
    int i, j;
    PrintString("\n\ttASCII Codes\n");
    PrintString("First 32 and last(DEL) are control code,\nso no need to print them.\n");
    for (i = 32; i < 127; i++) {
        if ((i - 32) % 6 == 0) PrintChar('\n');
        if (i < 100) PrintChar(' ');
        if (i < 10) PrintString(" ");
        PrintInt(i);
        PrintChar(':');
        PrintChar((char)i);
        PrintString(" ");
    }
    PrintChar('\n');
    Halt();
}
```

Hình 2.8.2. Mã nguồn chương trình ascii

2.8.3. Chương trình ‘sort’

- Chức năng: cho phép người dùng nhập vào một mảng n ($n \leq 100$) số nguyên. Sử dụng thuật toán bubble sort để sắp xếp mảng
- Mô tả:
 - + Nhập Kích Thước Mảng: Sử dụng một vòng lặp while để yêu cầu người dùng nhập kích thước của mảng (n).
 - + In thông báo "Input size of Array: " và đọc giá trị được nhập bằng hàm ReadInt().
 - + Nhập Giá Trị cho Mảng: Sử dụng vòng lặp for để nhập giá trị cho mỗi phần tử của mảng. In thông báo và sử dụng ReadInt() để đọc giá trị từ người dùng.
 - + In Mảng Chưa Sắp Xếp: Mỗi lần in 10 giá trị và xuống dòng.
 - + Sắp Xếp Mảng: Dùng Bubble Sort để sắp xếp mảng theo thứ tự tăng dần.
 - + In Mảng Đã Sắp Xếp
 - + Gọi hàm Halt() để kết thúc chương trình.

```

sort.c (~/Desktop/HDH/nachos/nachos-3.4/code/test) - gedit
sort.c x
#include "syscall.h"
#include "copyright.h"

main()
{
    int n = -1;
    int i, j;
    int tmp;
    int Arr[101];

    while (n < 0 || n > 100) {
        PrintString("Input size of Array: ");
        n = ReadInt();
    }

    for (i = 0; i < n; i++) {
        PrintString("Arr[");
        PrintInt(i);
        PrintString("] = ");
        Arr[i] = ReadInt();
        i = i + 1;
    }

    PrintString("Input Array:");
    for (i = 0; i < n; i++) {
        if (i % 10 == 0) {
            PrintChar('\n');
        }
        PrintInt(Arr[i]);
        PrintChar(' ');
    }
}

sort.c (~/Desktop/HDH/nachos/nachos-3.4/code/test) - gedit
sort.c x
    PrintChar('\n');

    for (i = 0; i < n - 1; i++) {
        for (j = 0; j < n - i - 1; j++) {
            if (Arr[j] > Arr[j + 1]) {
                tmp = Arr[j];
                Arr[j] = Arr[j + 1];
                Arr[j + 1] = tmp;
            }
        }
    }

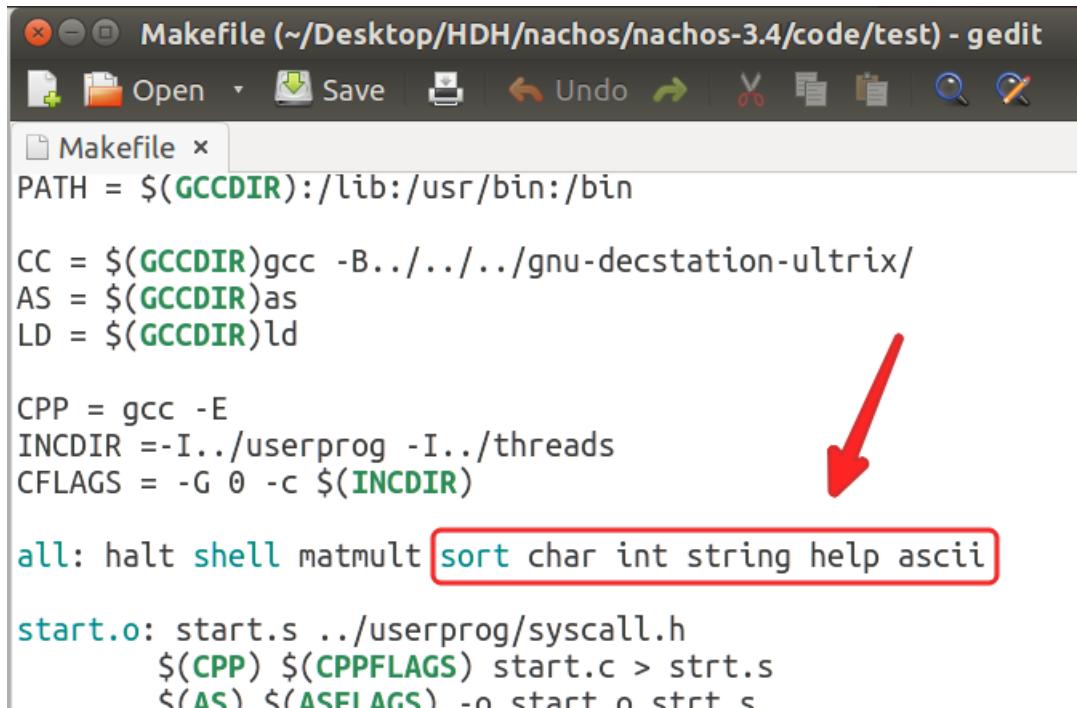
    PrintString("Sorted Array:");
    for (i = 0; i < n; i++) {
        if (i % 10 == 0) {
            PrintChar('\n');
        }
        PrintInt(Arr[i]);
        PrintChar(' ');
    }
    PrintChar('\n');

    Halt();
}

```

2.9. Sửa makefile trong folder test để biên dịch các hàm đã viết

Bổ sung các hàm vừa tạo vào trong file Makefile (thuộc “./nachos/nachos-3.4/code/test”) để có thể biên dịch và chạy thử



```

Makefile (~/Desktop/HDH/nachos/nachos-3.4/code/test) - gedit
File Open Save Print Undo Redo Find Replace Selection Cut Copy Paste Select All Find Next Find Previous Selection Cut Copy Paste Select All Find Next Find Previous
Makefile x
PATH = $(GCCDIR):/lib:/usr/bin:/bin

CC = $(GCCDIR)gcc -B../../../../../gnu-decstation-ultrix/
AS = $(GCCDIR)as
LD = $(GCCDIR)ld

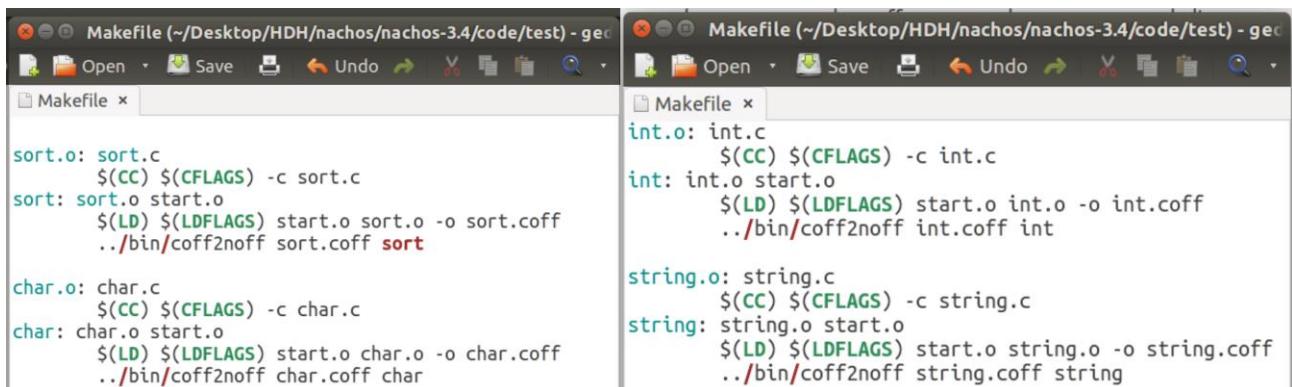
CPP = gcc -E
INCDIR =-I../userprog -I../threads
CFLAGS = -G 0 -c $(INCDIR)

all: halt shell matmult sort char int string help ascii

start.o: start.s ../userprog/syscall.h
$(CPP) $(CPPFLAGS) start.c > strt.s
$(AS) $(ASFLAGS) -o start.o strt.s

```

Hình 2.9.1. Thêm tên các chương trình ở dòng "all:..."



<pre> sort.o: sort.c \$(CC) \$(CFLAGS) -c sort.c sort: sort.o start.o \$(LD) \$(LDFLAGS) start.o sort.o -o sort.coff/bin/coff2noff sort.coff sort char.o: char.c \$(CC) \$(CFLAGS) -c char.c char: char.o start.o \$(LD) \$(LDFLAGS) start.o char.o -o char.coff/bin/coff2noff char.coff char </pre>	<pre> int.o: int.c \$(CC) \$(CFLAGS) -c int.c int: int.o start.o \$(LD) \$(LDFLAGS) start.o int.o -o int.coff/bin/coff2noff int.coff int string.o: string.c \$(CC) \$(CFLAGS) -c string.c string: string.o start.o \$(LD) \$(LDFLAGS) start.o string.o -o string.coff/bin/coff2noff string.coff string </pre>
--	---

Hình 2.9.2. Thêm quy tắc biên dịch cho các chương trình (1)

```

Makefile (~/Desktop/HDH/nachos/nachos-3.4/code/test) -ged
File Open Save Print Undo Redo Find Replace Copy Paste Find Next Find Previous Find All Find and Replace
Makefile x
help.o: help.c
    $(CC) $(CFLAGS) -c help.c
help: help.o start.o
    $(LD) $(LDFLAGS) start.o help.o -o help.coff
    ..../bin/coff2noff help.coff help

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

```

Hình 2.9.3. Thêm quy tắc biên dịch cho các chương trình (2)

Sau đó biên dịch lại NachOS bằng lệnh **make**

2.10. Demo

Cách chạy chương trình:

Bước 1: Chuyển terminal đến thư mục code trong folder ‘nachos-3.4’

Bước 2: Gõ lệnh: “./userprog/nachos -rs 1023 -x ./test/<tên chương trình>” để chạy chương trình mong muốn.

Ví dụ: để chạy chương trình help, ta gõ: “./userprog/nachos -rs 1023 -x ./test/help”

2.10.1. Demo 2 hàm read-print Int:

```

root@krammus-VirtualBox:/home/krammus/Downloads/HDH_Final/nachos/nachos-3.4/code#
./userprog/nachos -rs 1023 -x ./test/int
Input Integer: 109
Inputted Integer: 109
Machine halting!

Ticks: total 377315590, idle 377314914, system 630, user 46
Disk I/O: reads 0, writes 0
Console I/O: reads 4, writes 39
Paging: faults 0
Network I/O: packets received 0, sent 0

Cleaning up...
root@krammus-VirtualBox:/home/krammus/Downloads/HDH_Final/nachos/nachos-3.4/code#

```

Hình 2.10.1. Demo1 ReadInt – PrintInt

```
root@krammus-VirtualBox:/home/krammus/Downloads/HDH_Final/nachos/nachos-3.4/code# ./userprog/nachos -rs 1023 -x ./test/int
Input Integer: r

Input is not an integer.
Machine halting!

Ticks: total 190072073, idle 190071734, system 320, user 19
Disk I/O: reads 0, writes 0
Console I/O: reads 2, writes 16
Paging: faults 0
Network I/O: packets received 0, sent 0

Cleaning up...
root@krammus-VirtualBox:/home/krammus/Downloads/HDH_Final/nachos/nachos-3.4/code#
```

Hình 2.10.2. Demo 2 ReadInt() - PrintInt()

2.10.2. Demo ReadChar() – PrintChar()

```
root@krammus-VirtualBox:/home/krammus/Downloads/HDH_Final/nachos/nachos-3.4/code# ./userprog/nachos -rs 1023 -x ./test/char
Input Char: r
Inputted Char: r
Machine halting!

Ticks: total 415239331, idle 415238764, system 520, user 47
Disk I/O: reads 0, writes 0
Console I/O: reads 2, writes 31
Paging: faults 0
Network I/O: packets received 0, sent 0

Cleaning up...
root@krammus-VirtualBox:/home/krammus/Downloads/HDH_Final/nachos/nachos-3.4/code#
```

Hình 2.10.3. Demo 1 ReadChar() - PrintChar()

```

root@krammus-VirtualBox:/home/krammus/Downloads/HDH_Final/nachos/nachos-3.4/code#
./userprog/nachos -rs 1023 -x ./test/char
Input Char: 23
Only 1 character can be entered.
Inputted Char:
Machine halting!

Ticks: total 275777331, idle 275776764, system 520, user 47
Disk I/O: reads 0, writes 0
Console I/O: reads 3, writes 31
Paging: faults 0
Network I/O: packets received 0, sent 0

Cleaning up...
root@krammus-VirtualBox:/home/krammus/Downloads/HDH_Final/nachos/nachos-3.4/code#
./userprog/nachos -rs 1023 -x ./test/char
Input Char:
Empty.
Inputted Char:
Machine halting!

Ticks: total 97875451, idle 97874884, system 520, user 47
Disk I/O: reads 0, writes 0
Console I/O: reads 1, writes 31
Paging: faults 0
Network I/O: packets received 0, sent 0

Cleaning up...
root@krammus-VirtualBox:/home/krammus/Downloads/HDH_Final/nachos/nachos-3.4/code#

```

Hình 2.10.4. Demo 2 ReadChar() - PrintChar()

2.10.3. Demo ReadString và PrintString

```

root@krammus-VirtualBox:/home/krammus/Downloads/HDH_Final/nachos/nachos-3.4/code#
Search your computer and online sources test/string
Input String:
this is a test
Inputted String:
this is a test
Machine halting!

Ticks: total 564432708, idle 564431803, system 860, user 45
Disk I/O: reads 0, writes 0
Console I/O: reads 15, writes 49
Paging: faults 0
Network I/O: packets received 0, sent 0

Cleaning up...
root@krammus-VirtualBox:/home/krammus/Downloads/HDH_Final/nachos/nachos-3.4/code#

```

Hình 2.10.5. Demo ReadString() - PrintString()

2.10.4. Demo chương trình help

```
root@krammus-VirtualBox:/home/krammus/Downloads/HDH_Final/nachos/nachos-3.4/code# ./userprog/nachos -rs 1023 -x ./test/help
Nhóm gồm 5 thành viên:
21120279 - Lê Trần Minh Khue
21120289 - Diep Quoc Hoang Nam
21120290 - Hoang Trung Nam
21120348 - Nguyen Tran Trinh
21120354 - Luong Thanh Tu
Giới thiệu số chương trình:
1. ascii: In ra bảng mã ascii
2. sort: Nhập n và so sánh phần tử của mảng,
sắp xếp sau đó in ra mảng đã sắp xếp
Machine halting!

Ticks: total 35112, idle 31200, system 3830, user 82
Disk I/O: reads 0, writes 0
Console I/O: reads 0, writes 312
Paging: faults 0
Network I/O: packets received 0, sent 0

Cleaning up...
root@krammus-VirtualBox:/home/krammus/Downloads/HDH_Final/nachos/nachos-3.4/code#
```

Hình 2.10.6. Kết quả biên dịch chương trình help

2.10.5. Demo chương trình ascii

```
root@krammus-VirtualBox:/home/krammus/Downloads/HDH_Final/nachos/nachos-3.4/code# ./userprog/nachos -rs 1023 -x ./test/ascii
ASCII Codes
First 32 and last(DEL) are control code,
so no need to print them.

 32:   33:!  34:":  35:#  36:$  37:%
 38:&  39:'  40:(  41:)  42:*  43:+
 44:,  45:-  46:.  47:/  48:@  49:-
 50:2  51:3  52:4  53:5  54:6  55:7
 56:8  57:9  58:::  59:;  60:<  61:=
 62:>  63:?
 64:@  65:A  66:B  67:C
 68:D  69:E  70:F  71:G  72:H  73:I
 74:J  75:K  76:L  77:M  78:N  79:O
 80:P  81:Q  82:R  83:S  84:T  85:U
 86:V  87:W  88:X  89:Y  90:Z  91:[
 92:\  93:]  94:^  95:_  96:`  97:a
 98:b  99:c  100:d  101:e  102:f  103:g
 104:h  105:i  106:j  107:k  108:l  109:m
 110:n  111:o  112:p  113:q  114:r  115:s
 116:t  117:u  118:v  119:w  120:x  121:y
 122:z  123:{  124:|  125:}  126:~

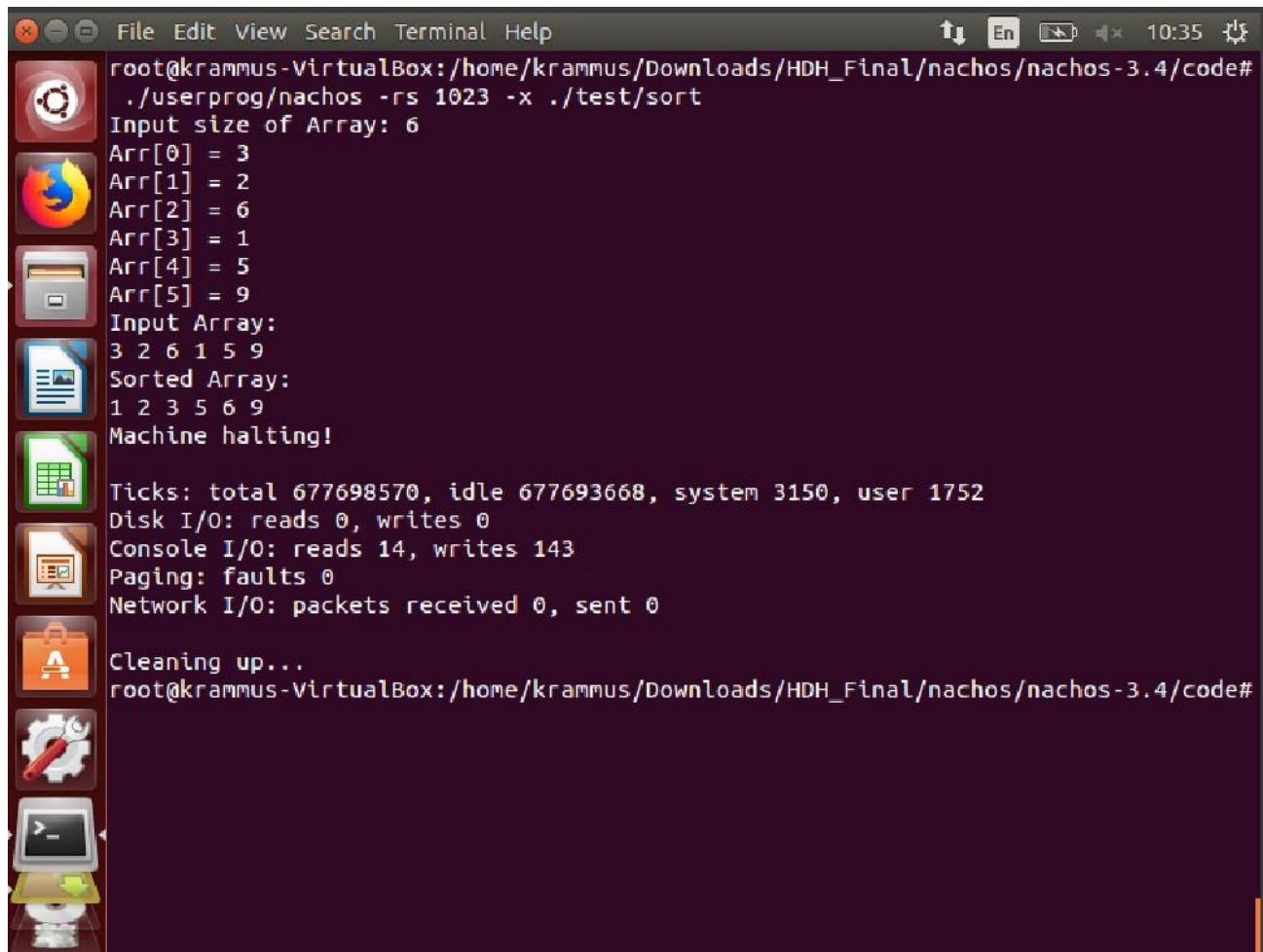
Machine halting!

Ticks: total 113265, idle 86100, system 20540, user 6625
Disk I/O: reads 0, writes 0
Console I/O: reads 0, writes 861
Paging: faults 0
Network I/O: packets received 0, sent 0

Cleaning up...
```

Hình 2.10.7. Kết quả biên dịch chương trình ascii

2.10.6. Demo chương trình sort



The screenshot shows a terminal window with a dark background and light-colored text. At the top, there's a menu bar with "File", "Edit", "View", "Search", "Terminal", and "Help". On the right side of the window, there are several icons representing different applications, such as a terminal, file manager, browser, and others. The main area of the terminal contains the following text:

```
root@krammus-VirtualBox:/home/krammus/Downloads/HDH_Final/nachos/nachos-3.4/code# ./userprog/nachos -rs 1023 -x ./test/sort
Input size of Array: 6
Arr[0] = 3
Arr[1] = 2
Arr[2] = 6
Arr[3] = 1
Arr[4] = 5
Arr[5] = 9
Input Array:
3 2 6 1 5 9
Sorted Array:
1 2 3 5 6 9
Machine halting!
Ticks: total 677698570, idle 677693668, system 3150, user 1752
Disk I/O: reads 0, writes 0
Console I/O: reads 14, writes 143
Paging: faults 0
Network I/O: packets received 0, sent 0
Cleaning up...
root@krammus-VirtualBox:/home/krammus/Downloads/HDH_Final/nachos/nachos-3.4/code#
```

Hình 2.10.8. Kết quả biên dịch chương trình sort

3. TÀI LIỆU THAM KHẢO

[1] Trần Trung Dũng, Phạm Tuán Sơn (2022). Giáo trình Hệ điều hành. NXB Khoa học và Kỹ thuật, Hồ Chí Minh.

[2] Tài liệu giảng viên cung cấp trong quá trình thực hiện đồ án. Bao gồm:

- [1] Bien dich va Cai dat nachos.
- [2] Giao tiep giua HDH Nachos va chuong trinh nguoi dung.
- [3] Cach Viet Mot SystemCall.
- [4] Cach Them 1 Lop Vao Nachos.
- Huong dan cai Nachos tren ubuntu.

[3] General Nachos Documentation, <https://homes.cs.washington.edu/~tom/nachos/>, truy cập ngày 30/11/2023.

[4] Overview of the Nachos Operating System ,
https://student.cs.uwaterloo.ca/~cs350/common/os_overview.html, truy cập ngày 30/11/2023.