KHOA CÔNG NGHỆ THÔNG TIN

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

HỆ ĐIỀU HÀNH

Project 01: Exceptions và các System Call đơn giản

ĐỒ ÁN/BÀI TẬP MÔN HỌC HỆ ĐIỀU HÀNH

HỌC KỲ I – NĂM HỌC 2022 - 2023

KHOA CÔNG NGHỆ THÔNG TIN
TRƯỜNG ĐẠI HỌC KHOA HỌC TỰ NHIÊN
227 Nguyễn Văn Cừ, Phường 4, Quận 5, TP.HCM
Điện Thoại: (08) 38.354.266 - Fax:(08) 38.350.096

Danh sách thành viên nhóm:

STT	MSSV	Họ và tên	Emai
1	20120049	Nguyễn Hải Đăng	20120049@student.hcmus.edu.vn
2	20120077	Nguyễn Quang Hiển	20120077@student.hcmus.edu.vn
3	20120084	Nguyễn Văn Hiếu	20120084@student.hcmus.edu.vn

Phân công và đánh giá mức độ hoàn thành:

Phần	Câu	Ghi chú	Phân công	Điểm	Mức độ hoàn thành
1		Hiểu mã NachOS	Hiển	1	100%
2		Hiểu thiết kế	Hiếu	1	100%
3	1	Xử lý Exceptions	Hiển	0.5	100%
	2	Tăng giá trị PC	Hiển	0.5	100%
	3	ReadNum	Hiển	0.75	100%
	4	PrintNum	Hiển	0.75	100%
	5	ReadChar	Hiếu	0.5	100%
	6	PrintChar	Hiếu	0.5	100%
	7	RandomNum	Hiếu	0.5	100%
	8	ReadString	Hiếu	0.75	100%
	9	PrintString	Đăng	0.75	100%
	10	help	Đăng	0.5	100%

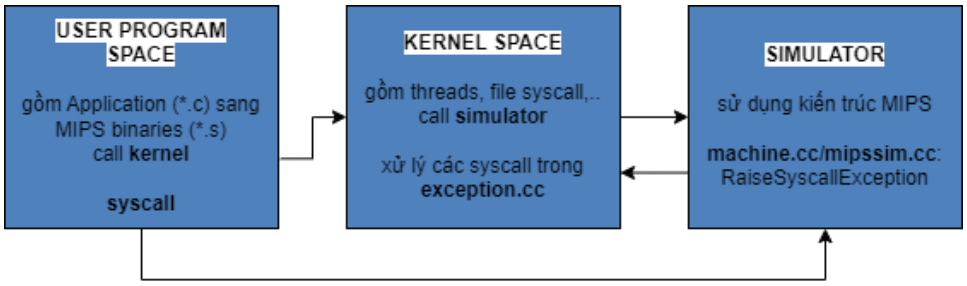
	11	ascii	Đăng	0.5	100%
	12	sort	Đăng	0.5	100%
	13	Không để user sụp HĐH	Đăng	0.5	100%
4		Báo cáo	Đăng, Hiển, Hiếu	0.5	100%

Mục lục

I. Tổng quan NachOS

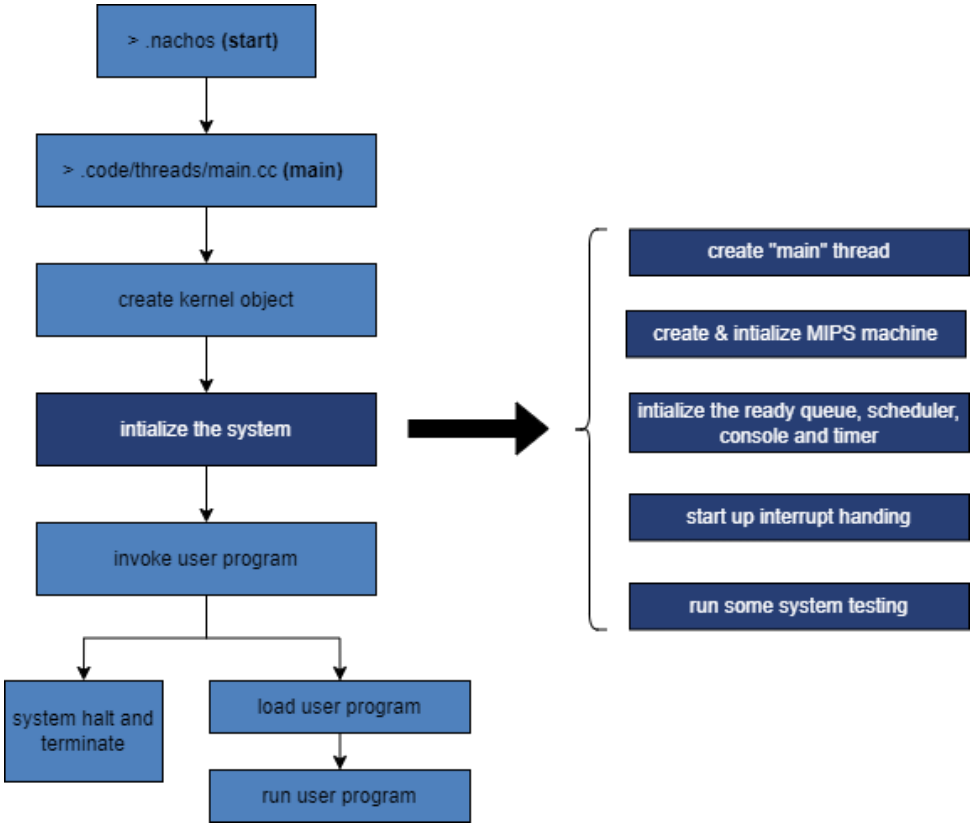
1. Thành phần NachOS

NachOS gồm 3 thành phần chính:

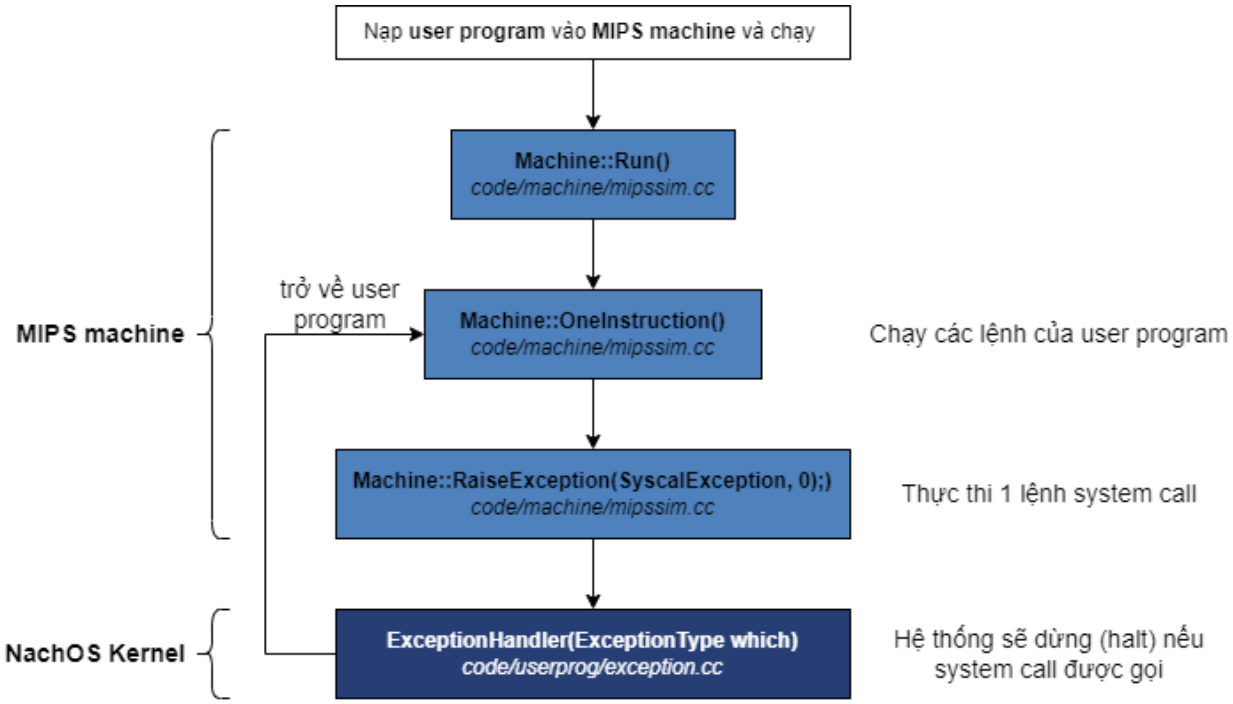


Nachos xuất hiện như **single threaded process**.

2. Khởi động NachOS

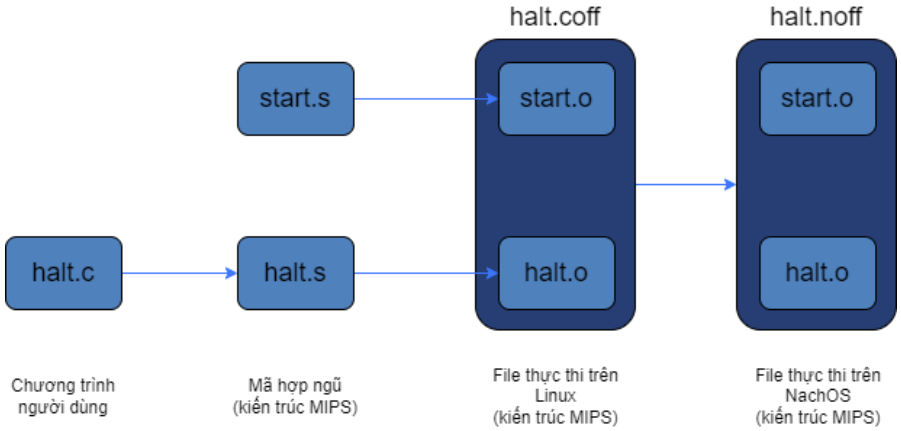


3. System Call NachOS



4. Quá trình biên dịch NachOS

Mô phỏng quá trình biên dịch bằng cách chạy chương trình "**./build.linux/nachos -x ./test/halt**"



Một chương trình (ví dụ:Halt.c) muốn được thực thi thì nó cần phải được biên dịch quá trình biên dịch trên NachOS gồm 3 bước:

- 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* được liên kết với starts.s để tạo thành tập tin halt.coff (bao gồm halt.o và start.o) là dạng file thực thi trên linux với kiến trúc MIPS
- Bước 3. Tập tin *halt.coff* được tiện ích coff2noff được chuyển thành tập tin halt.noff dạng file thực thi trên NachOS kiến trúc MIPS

II. Hiểu thiết kế

Hai chế độ ***Kernel (System Space)*** và **User Space**:

- ***Kernel Space***: 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. Nó được dành riêng cho các chức năng có độ tin cậy cao nhất bên trong hệ thống. Kernel space thường được dành riêng cho các chức năng hoạt động ở cấp độ thấp nhất, đáng tin cậy nhất của hệ điều hành. Do số lượng truy cập mà kernel có, bất kỳ sự không ổn định nào bên trong mã thực thi kernel cũng có thể dẫn đến lỗi hệ thống hoàn toàn.
- ***User Space***: Mã thực thi bị giới hạn truy cập. Nó là không gian địa chỉ mà các process user thông thường chạy. Những processes này không thể truy cập trực tiếp tới kernel space được. Khi đó lời gọi API được sử dụng tới kernel để truy vấn memory và truy cập thiết bị phần cứng. Bởi truy cập bị hạn chế, các trục trặc hay vấn đề gì xảy ra trong user mode chỉ bị giới hạn trong không gian hệ thống mà chúng đang hoạt động và luôn có thể khôi phục được. Hầu hết các đoạn mã đang chạy trên máy tính của bạn sẽ thực thi trong user mode.

Xét trên vùng nhớ RAM thật của máy tính thì ***Kernel Space*** là vùng nhớ của hệ điều hành NachOS; ***User Space*** là vùng nhớ chương trình ứng dụng chạy trên NachOS/MIPS.

Hệ thống NachOS chia làm 3 phần (không tính phần mạng):

1. Chương trình ứng dụng (../code/tets): nhà lập trình ứng dụng sẽ thao tác ở đây.
2. Cỗ máy ảo MIPS
3. Hệ điều hành NachOS

III. Cài đặt tổng quan

1. Cài đặt trước

Ở *../code/threads/* trong file *system.h* và *system.cc* thực hiện khai báo, cấp phát, và xóa vùng nhớ cấp phát một biến toàn cục thuộc lớp "SynchConsole" để hỗ trợ việc nhập xuất với màn hình console.

Ở *../code/userprog* trong file *exception.cc* thực hiện cài đặt hai hàm *char* User2System(int virtAddr, int limit)* để sao chép vùng nhớ từ user sang system và hàm *int System2User(int virtAddr, int len, char* buffer)* để sao chép vùng nhớ từ system về cho user

2. Cài giá trị thanh ghi

- R2: Lưu mã syscall đồng thời lưu kết quả trả về của mỗi syscall nếu có.
- R4: Lưu tham số thứ nhất.
- R5: Lưu tham số thứ hai.
- R6: Lưu tham số thứ ba.
- R7: Lưu tham số thứ tư.

3. Các bước cài đặt System Call

- Bước 1: Tạo macro: Tránh nhầm lẫn việc gọi hàm, cần define con số cụ thể trong kernel space thành một macro.
  Vào *../code/userprog/syscall.h*
  #define <tên system call> <số cụ thể>
- Bước 2: Định nghĩa hàm trong file assembler trong 2 files*: code/test/start.c* và *code/test/start.s*

- Bước 3: Định nghĩa cụ thể cho một công việc. Mở file *usrprog/exception.cc* Chuyển đoạn mã *If…..else* sang đoạn mã *switch….case* với các *case* là các syscall cần gọi.
- Bước 4 : Viết chương trình ở mức người dùng để kiểm tra file *../code/test* Sử dụng hàm như đã khai báo prototype ở */code/userprog/syscall.h*
- Bước 5 :Thêm vào ở file Makefile trong */code/test/*
- Bước 6 : Biên dịch và chạy Nachos

IV. Cài đặt System Call và Exception

1. Viết lại exception.cc

- Bước 1: vào file code/test/machine.h để lấy danh sách các exception cần cài đặt.

```
enum ExceptionType { NoException,           // Everything ok!
                     SyscallException,      // A program executed a system call.
                     PageFaultException,    // No valid translation found
                     ReadOnlyException,     // Write attempted to page marked
                                            // "read-only"
                     BusErrorException,     // Translation resulted in an
                                            // invalid physical address
                     AddressErrorException, // Unaligned reference or one that
                                            // was beyond the end of the
                                            // address space
                     OverflowException,     // Integer overflow in add or sub.
                     IllegalInstrException, // Unimplemented or reserved instr.

                     NumExceptionTypes
};
```

Hình 1. Danh sách các exception.

- Bước 2: Viết tất cả các exception đã liệt kê vô file code/userprog/exception.cc dưới dạng switch…case trong hàm void ExceptionHandler(ExceptionType which).
- Bước 3: viết code DEBUG và gọi hàm SysHalt() để tắt hệ điều hành. Dưới đây là ví dụ:

```
case NumExceptionTypes:
    cerr << "Error " << which << " occurs\n";
    SysHalt();
    ASSERTNOTREACHED();
```

Hình 2. Ví dụ về xử lý exception.

2. Tăng giá trị Program Counter

Làm tăng Programming Counter để nạp lệnh tiếp theo để thực hiện. Ta thực hiện lưu giá trị của PC hiện tại cho PC trước, nạp giá trị kế cho PC hiện tại, nạp giá trị kế tiếp nữa cho PC kế.

3. System Call int Readnum()

- Mô tả cài đặt: *handleSC_ ReadNum()*
- Chức năng: đọc một số nguyên do người dùng nhập vào.
- Cách viết:
    - Chúng ta cài đặt hàm *int SysReadNum()* ở *ksyscall.h* để lấy số result được đọc.
    - Sau khi nhận được ký tự nhập vào chúng ta lưu giá trị đó vào thanh ghi (sử dụng hàm *WriteRegister* được cài đặt ở file *machine.h*).
    - Cuối cùng là tăng giá trị Program Counter trước khi System Call trả về kết quả, tránh vòng lặp vô hạn.
- Các hàm hỗ trợ:
    - int SysReadNum(): hàm này sẽ xử lý các trường có thể xảy ra khi người dùng nhập một kí tự bất kì và luôn trả về kết quả là một số nguyên.
        - Nếu người dùng nhập kí tự khoảng cách, ghi một số nằm ngoài data range của kiểu int, nhập một kí tự hay nhiều kí tự không phải là số: hàm trả về số 0 và trên bash khi viết câu lệnh thực thi sẽ xuất hiện số 0 trên màn hình.

```
dangcpr@ubuntu:~/nachos/NachOS-4.0/code/test$ ../build.linux/nachos -x numio

0Machine halting!

Ticks: total 238936562, idle 238936458, system 80, user 24
Disk I/O: reads 0, writes 0
Console I/O: reads 2, writes 1
Paging: faults 0
Network I/O: packets received 0, sent 0
```

Hình 3. Nếu nhập khoảng trắng.

```
dangcpr@ubuntu:~/nachos/NachOS-4.0/code/test$ ../build.linux/nachos -x numio
-
0Machine halting!

Ticks: total 77403172, idle 77403038, system 110, user 24
Disk I/O: reads 0, writes 0
Console I/O: reads 2, writes 1
Paging: faults 0
Network I/O: packets received 0, sent 0
dangcpr@ubuntu:~/nachos/NachOS-4.0/code/test$
```

Hình 4. Nếu nhập kí tự không phải số nguyên.

```
dangcpr@ubuntu:~/nachos/NachOS-4.0/code/test$ ../build.linux/nachos -x numio
10000000000000
0Machine halting!

Ticks: total 275152882, idle 275152418, system 440, user 24
Disk I/O: reads 0, writes 0
Console I/O: reads 13, writes 1
Paging: faults 0
Network I/O: packets received 0, sent 0
```

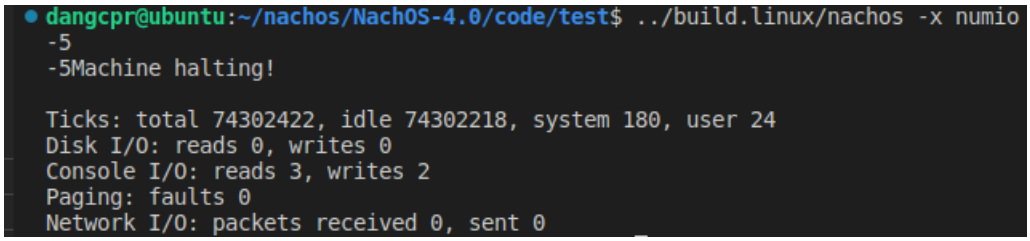
Hình 5. Nếu nhập số nguyên nằm ngoài data range của kiểu int.

- Nếu người dùng nhập một số (dương hay âm) thì hàm sẽ trả về số mà người dùng nhập.

```
dangcpr@ubuntu:~/nachos/NachOS-4.0/code/test$ ../build.linux/nachos -x numio
3
3Machine halting!

Ticks: total 38845972, idle 38845838, system 110, user 24
Disk I/O: reads 0, writes 0
Console I/O: reads 2, writes 1
Paging: faults 0
Network I/O: packets received 0, sent 0
```

Hình 6. Nhập số nguyên dương.



Hình 7. Nhập số nguyên âm.

4. System Call void PrintNum(int number)

- Mô tả cài đặt: *handleSC_PrintNum(int number)*
- Chức năng: xuất ra số nguyên được người dùng truyền qua tham số number.
- Cách viết:
  - Đầu tiên chúng ta cần lấy số nguyên được lưu ở thanh ghi r4.
  - Sau khi nhận được ký tự nhập vào, tiến hành cài đặt hàm *SysPrintNum()* để in số nguyên ra màn hình.
  - Cuối cùng là tăng giá trị Program Counter trước khi System Call trả về kết quả, tránh vòng lặp vô hạn.
- Các hàm hỗ trợ:
  - void SysPrintNum(int num): hàm hỗ trợ in số nguyên.
    - Nếu người dùng nhập số nguyên dương thì chương trình sẽ in nguyên số đó ra màn hình (nếu như số đó nằm ngoài data range thì màn hình in ra số 0).
    - Nếu người dùng nhập số nguyên âm thì chương trình sẽ tách dấu "-" thành 1 kí tự riêng, kiểm tra phần nguyên còn lại như trường hợp trên, rồi ghép dấu "-" vào trước số và in số ra màn hình.

5. System Call char ReadChar()

- Mô tả cài đặt: *handleSC_ReadChar()*
- Chức năng: đọc một ký tự do người dùng nhập vào.
- Cách viết:

- Chúng ta cài đặt hàm *char SysReadChar()* ở *ksyscall.h*, sử dụng phương thức *GetChar()* của lớp *Synch Console* để nhận ký tự nhập vào từ Console IO.
- Sau khi nhận được ký tự nhập vào chúng ta lưu giá trị đó vào thanh ghi (sử dụng hàm *WriteRegister* được cài đặt ở file *machine.h*).
- Cuối cùng là tăng giá trị Program Counter trước khi System Call trả về kết quả, tránh vòng lặp vô hạn.

6. System Call void PrintChar(char character)

- Mô tả cài đặt: *handleSC_PrintChar(char character)*
- Chức năng: xuất ra ký tự được người dùng truyền qua character
- Cách viết:
  - Đầu tiên chúng ta cần lấy ký tự nhập vào được lưu ở thanh ghi.
  - Sau khi nhận được ký tự nhập vào, tiến hành cài đặt hàm *SysPrintChar()* sử dụng hàm *PutChar()* được xây dựng sẵn trong lớp Synch Console để in giá trị đó ra màn hình Console.
  - Cuối cùng là tăng giá trị Program Counter trước khi System Call trả về kết quả, tránh vòng lặp vô hạn.

7. System Call int RamdomNum()

- Mô tả cài đặt: *handleSC_RandomNum()*
- Chức năng: trả về một số nguyên dương ngẫu nhiên
- Cách viết:
  - Chúng ta sử dụng hàm *random()* có sẵn ở thư viện stdlib.h để viết hàm *SysRandomNum()* ở file *ksyscall.h*
  - Sau khi random được giá trị thì lưu kết quả vào thanh ghi thông qua hàm *WriteRegister()*
  - Cuối cùng là tăng giá trị Program Counter trước khi System Call trả về kết quả, tránh vòng lặp vô hạn.

8. System Call void ReadString(char[] buffer, int length)

- Mô tả cài đặt: *handleSC_ReadString()*
- Chức năng: đọc một chuỗi do người dùng nhập vào.
- Cách viết:
    - Đầu tiên ta lấy địa chỉ của chuỗi và độ dài của chuỗi lưu vào 2 biến memPtr và length.
    - Kiểm tra độ chiều của chuỗi length có lớn hơn 255 ký tự hay không. Nếu có thì thông báo cho người dùng biết và thoát Syscall.
    - Xây dựng hàm SysReadString() để đọc từng ký tự nhập vào của chuỗi thông qua duyệt từng ký tự và sử dụng hàm ReadChar() đã xây dựng.
    - Sử dụng hàm stringSys2User() thực hiện chuyển chuỗi vào bộ nhớ của chương trình ứng dụng.
    - Cuối cùng là tăng giá trị Program Counter trước khi System Call trả về kết quả, tránh vòng lặp vô hạn.

9. System Call void PrintString(char[] buffer)

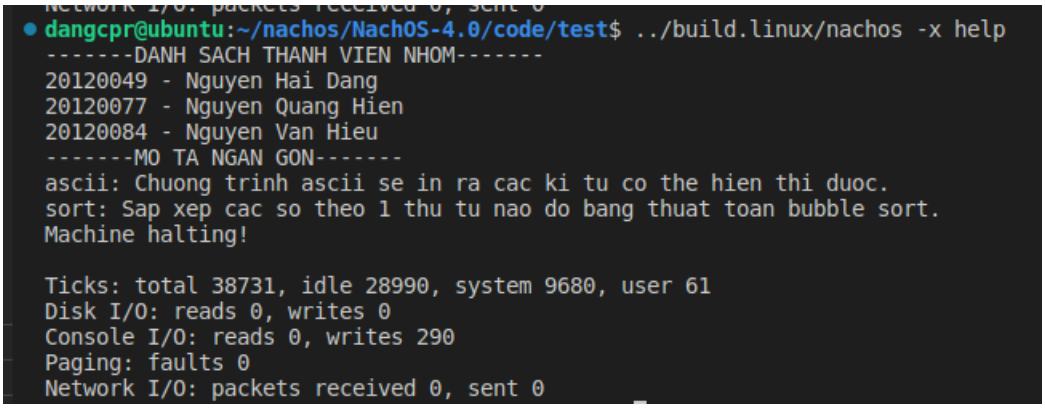
- Mô tả cài đặt: *handleSC_PrintString()*
- Xuất ra chuỗi đã được truyền vào tham số buffer trong nguyên mẫu system call.
- Input: char[] buffer
- Output: chuỗi được in ra màn hình console.
- Cách viết:
    - Đọc địa chỉ của buffer và lưu vào thanh ghi r4 (ReadRegister(4)), gán vào biến memPtr.
    - Hàm stringUser2System(memPtr) sẽ đọc chuỗi buffer từ địa chỉ memPtr.
    - Hàm SysPrintString(buffer, strlen(buffer)) sẽ hỗ trợ in chuỗi buffer ra màn hình.

- Cuối cùng là tăng giá trị Program Counter trước khi System Call trả về kết quả, tránh vòng lặp vô hạn.

- Các hàm hỗ trợ:
  - char* stringUser2System(int addr, int convert_length = -1): Hàm hỗ trợ đọc chuỗi từ địa chỉ addr.
    - Đầu tiên, sẽ có 1 vòng lặp để tìm độ dài của chuỗi, nếu như kí tự đang đọc là \0, mặt khác nếu \0 không xuất hiện thì tham số convert_length sẽ làm kết thúc process hoặc chuỗi rỗng.
    - Sau đó, sẽ có một vòng lặp thứ hai để gắn từng kí tự vô chuỗi.
  - void SysPrintString(char* buffer, int length): Hàm này nằm trong file ksyscall.h của thư mục userprog. Hàm hỗ trợ in chuỗi ra màn hình bằng cách sử dụng phương thức synchConsoleout và PutChar của kernel.

10. Chương trình help

- Chương trình để in ra thông tin về về nhóm và các thành viên, cách sử dụng chương trình ascii và sort.
- Chương trình sử dụng system call PrintString() để in chuỗi ra màn hình.
- Sau đó dùng system call Halt() để trả quyền về hệ điều hành.

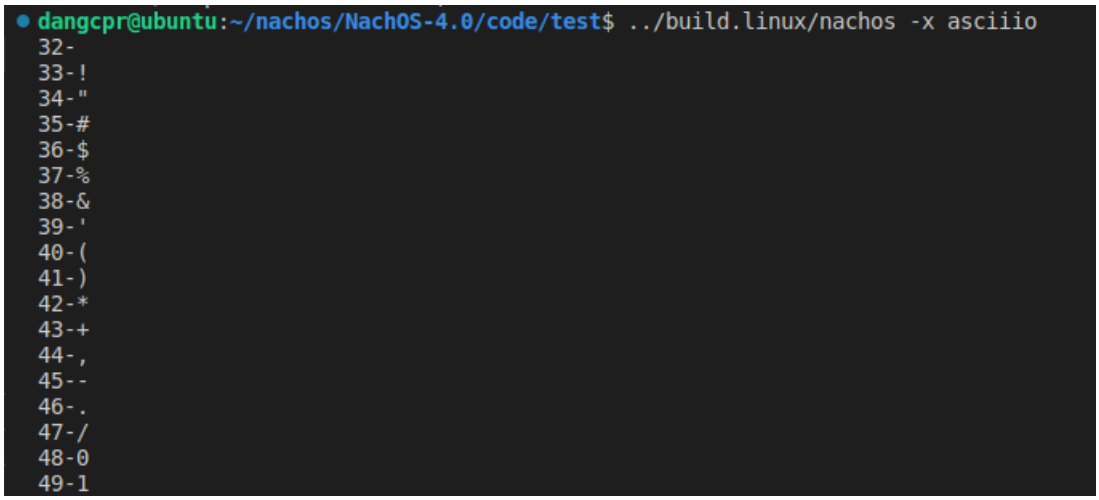


Hình 8. Chương trình help.

11. Chương trình ascii

- Mục đích chương trình này là in ra các kí tự có thể in ra trong bảng mã ascii.

- Chương trình tên là asciiio.c ở trong thư mục test.
- Các kí tự ascii có số thứ tự từ 32 đến 126 là những kí tự có thể in được. Vì vậy chúng em sử dụng vòng lặp với biến đếm i từ 32 đến 126 để in hết các kí tự có thể in được.
- Chúng em sử dụng system call PrintNum và PrintChar cho chương trình này.



Hình 9. Các kí tự có thể in được.

12. Chương trình Sort

- Sử dụng thuật toán bubble sort.
- Input: số lượng số nguyên và giá trị từng số.
- Output: dãy số sau khi sắp xếp.
- Chúng em sử dụng system call ReadNum để đọc số lượng phần tử.

Hình 10. Nhập số lượng số nguyên.

- Sau đó, màn hình console sẽ xuất hiện thông báo nhập từng phần tử.

```
dangcpr@ubuntu:~/nachos/NachOS-4.0/code/test$ ../build.linux/nachos -x sort
Enter n: 4
a[0]: 3
a[1]: -2
a[2]: 5
a[3]: 0
```

Hình 11. Nhập từng phần tử.

- Người dùng tiếp tục chọn cách sắp xếp (ấn 1 để sắp xếp tăng, 2 để sắp xếp giảm).

```
Your sort order (1: increasing, 2: decreasing) is: 2
```

Hình 12. Chọn thứ tự sắp xếp.

- Sau khi nhập đủ số phần tử, màn hình console sẽ xuất ra dãy theo thứ tự mà người dùng đã chọn.

```
Your sort order (1: increasing, 2: decreasing) is: 2
Day so sau khi sap xep la: 5 3 0 -2
Machine halting!

Ticks: total 240291316, idle 240285604, system 4570, user 1142
Disk I/O: reads 0, writes 0
Console I/O: reads 13, writes 121
Paging: faults 0
Network I/O: packets received 0, sent 0
```

Hình 13. Dãy đã sắp xếp.

- Chúng em đã viết chương trình sort.c và viết hay dòng sort.c và sort vào makefile trong thư mục test.

13. Không cho user làm sụp Hệ điều hành

- Sẽ có một số tình huống mà người dùng sẽ làm sụp hệ điều hành, và chúng em đã xử lý một số tình huống sau (các tình huống được code trong file ksyscall.h của thư mục userprog)
    - **Người dùng nhập chuỗi rỗng - hàm isBlank(char c):** nếu người dùng nhập chuỗi rộng hoặc nhập các kí tự không in được, màn hình console sẽ hiện ký tự "0". Hàm này sau đó được sử dụng trong hàm readNumberUntilBlank().

- **Chương trình yêu cầu nhập số nhưng người dùng lại nhập chữ cái hoặc nhập các kí tự khác không phải là chữ số:** màn hình sẽ in ra ký tự "0".
- **Người dùng nhập số nguyên vượt quá giá trị kiểu dữ liệu int:** màn hình sẽ in ra ký tự "0".

IV. Tài liệu tham khảo

[1]: Tài liệu Project 1 Tutorial, Hệ điều hành 20_22

[2]: Hướng dẫn NachOS, Dang Khoa