Báo Cáo Đồ Án Hệ Điều Hành

Project 02 – System Call Nachos

19127102 – Võ Hoàng Gia Bảo

19127406 – Ngô Huy Hoàng

19127424 – Triệu Nguyễn Minh Huy

19127457 – Nguyễn Tuấn Kiệt

19127509 – Trương Thế Phú

Logo

Description automatically generated

Bộ môn Hệ Điều Hành

Khoa Công nghệ thông tin

Đại học Khoa học tự nhiên TP. HCM

I. Mục lục:

1. Bảng phân công:

2. Giới thiệu về hệ điều hành Nachos:

a. Thiết kế của Nachos:

b. Cách thức hoạt động và thực thi một chương trình:

c. Cách mà Halt.noff được thực thi :

3. Cấu hình và cài đặt Nachos:

4. Khai báo biến toàn cục gSynchConsole:

5. Các bước cài đặt System Call:

6. Hiểu các mã chương trình Nachos cho trước :

7. Ý tưởng xây dựng và cài đặt các systemcall và exception :

a. Cài đặt các exception :

b. Các chương trình ở file test:

8. Demo thử hàm thiết kế:

9. Tài liệu tham khảo:

1. Bảng phân công:

|  |  |  |  |
| --- | --- | --- | --- |
| MSSV | Họ và tên |  | Công việc |
| 19127102 | Võ Hoàng Gia Bảo |  | -Tìm hiểu Nachos  -Code phần syscall,exception.cc |
| 19127406 | Ngô Huy Hoàng | Nhóm Trưởng | -Tìm hiểu Nachos  -Code phần syscall, exception.cc |
| 19127424 | Triệu Nguyễn Minh Huy |  | -Tìm hiểu Nachos  -Tìm hiểu thiết kế của Nachos |
| 19127457 | Nguyễn Tuấn Kiệt |  | -Tìm hiểu Nachos  - Giải thích Hàm nhóm thiết kế  -Tổng hợp và viết báo cáo |
| 19127509 | Trương Thế Phú |  | -Tìm hiểu Nachos  -Tìm hiểu mã chương trình  -Giải thích Hàm nhóm thiết kế |

2. Giới thiệu về hệ điều hành Nachos :

NachOS – Not Another Completely Heuristic Operating System . là một phần mềm giả lập hệ điều hành với kiến trúc MIPS ( Million Instruction Per Second ) . Được viết bằng C++ cho MIPS , NachOS chạy như một user-process trên hệ điều hành máy chủ.

**Các thành phần của NachOS :**

* NachOS Machine : NachOS mô phỏng đại khái gần giống với kiến trúc MIPS, có thanh ghi , bộ nhớ và CPU . NachOS Machine được thực thi bởi Machine Object khi NachOS được khởi động, nó chứa các methods như Run, ReadRegister, WriteRegister ,… .
* NachOS Threads : Trong NachOS, một lớp Threads được định nghĩa và có thể có những trạng thái như là ready, running, blocked hoặc just created. Thread object cũng có chứa các methods như PutThreadToSleep, YieldCPU, ThreadFork, ThreadStackAllocate, … .
* NachOS UserPrograms : NachOS chạy chương trình người dung trong chính không gian bảo mật của chúng . NachOS có thể chạy bất kỳ tệp nhị phân MIPS nào . NachOS yêu cầu thực thi tệp phải ở định dạng “Noff” trong khi các tệp đa phần được lưu dưới dạng “a.out” hay còn gọi là định dạng “Coff”.

a. Thiết kế của Nachos:

Hiện nay , các hệ điều hành của máy tính “hiện đại” thường tách riêng bộ nhớ ảo thành 2 khu vực : không gian nhân (kernel space) và không gian người dùng (user space) . Sự tách biệt này nhằm mục đích bảo vệ bộ nhớ và phần cứng khỏi những phần mềm hoặc lỗi độc hại .

|  |  |
| --- | --- |
| Kernel | User |
| Chạy nhân hệ điều hành đặc quyền , phần mở rộng của nhân và đa số các drivers. | Vùng bộ nhớ chỉ để chạy ứng dụng phần mềm hoặc để thực thi một số drivers. |
| Mã thực thi có quyền truy cập vào bất kỳ không gian địa chỉ nào của memory và phần cứng. | Mã thực thi bị giới hạn truy cập và không thể truy cập trực tiếp đến kernel space được. (Truy cập gián tiếp qua một phần nhỏ của kernel gọi là the system call). |
| Bất kỳ sự “ không ổn định “ nào bên trong mã thực thi kernel đều có thể dẫn đến lỗi hệ thống hoàn toàn. | Nếu có xảy ra trục trặc hoặc vấn đề thì đều có thể khôi phục được vì truy cập bị hạn chế và giới hạn trong không gian hệ thống mà user đang hoạt động. |

**-Sơ đồ cách thức hoạt động của 2 vùng Kernel và User:**

Diagram

Description automatically generated

-Kernel space code có đặc tính để run trong kernel mode, gọi là mã thực thi Ring 0.

-Điển hình trong kiến trúc x86 có 4 Ring :

* Ring 0 : Kernel Mode. ( Tương tác trực tiếp với phần cứng vật lý như CPU hoặc bộ nhớ )
* Ring 1 : Có thể sử dụng bởi máy ảo hoặc drivers.
* Ring 2 : Sử dụng bởi drivers.
* Ring 3 : Là những ứng dụng.

Đặc quyền sắp xếp từ thấp đến cao : Ring 3 > Ring 2 > Ring 1 > Ring 0 .

Riêng Linux chỉ sử dụng 2 Ring là : Ring 0 (cho Kernel ) và Ring 3 (cho User )

b. Cách thức hoạt động và thực thi các hàm:

**-Cách hoạt động của quá trình thực thi một chương trình được viết trên hệ điều hành NachOS :**

* Một chương trình có định dạng .c ( VD : halt.c , sort.c, start.c , … ) là chương trình được biên dịch bởi cross-compiler GCC thành tập tin có định dạng .s (VD : halt.s , start.s , …) . *Định dạng .c là file input được biên dịch thành định dạng .s là mã hợp ngữ chạy trên kiến trúc MIPS .* Lưu ý : chính vì GCC không hỗ trợ biên dịch các thư viện có sẵn trong C/C++ như stdio.h, conio.h, … , nên các file input định dạng .c phải viết mà không thể sử dụng hàm trong các thư viện có sẵn . Tức là chỉ có thể dùng những toán tử (+, -, \*,/, while, if, … ) mà không thể thao tác với người dùng ( VD : muốn in ra màn hình, nhập dữ liệu từ bàn phím, tạo tập tin , … ) , để làm được điều này thì ta dùng System Call .
* Sau đó tập tin sau khi được biên dịch bởi cross-compiler này sẽ được liên kết với tập tin thư viện của hợp ngữ và tạo thành tập tin với định dạng .coff .
* Kế đến tập tin với định dạng .coff đó sẽ được chuyển thành tập tin định dạng .noff (định dạng thực thi trên hệ điều hành NachOS ) bởi tiện ích “coff2noff” .

System Call là lời gọi từ người dùng khi user không làm được gì đó mà cần đến sự giúp đỡ của hệ điều hành.

c. Cách mà Halt.noff được thực thi :

* Trong chương trình Halt chỉ có duy nhất một hàm Halt() . Đây là hàm có nhiệm vụ tắt máy ( hay ở đây là máy ảo ) và hàm Halt() được gọi là system call. Như định nghĩa về system call , chương trình Halt không thể tự tắt máy được mà phải cần đến hệ điều hành.
* Sau khi chương trình nạp vào RAM ảo , CPU MIPS bắt đầu nạp lệnh trong halt.noff để thực thi lệnh.

Diagram

Description automatically generated- Lệnh *jal main* ( đánh dấu x ) để nhảy vào hàm main() của halt.c.

Tiếp đến là lệnh *addiu $2,$0,SC\_Halt* ( được khoanh tròn ) lưu giá trị của SC\_Halt ( là 1 số nguyên ) vào thanh ghi thứ 2.

* Lệnh *syscall* ( bên dưới lệnh *addiu $2,$0,SC\_Halt* ) dùng lệnh này để gọi và xử lý system call. Khi nạp lệnh này , mã hoạt động sẽ nhận thấy lệnh *OP\_SYSCALL* và nó sẽ gọi hàm RaiseException(). Vì là system call nên hàm *syscall* sẽ chuyển qua kernel mode để thực hiện xử lý system call đó, sau đó chuyển về lại cho user mode để chạy tiếp chương trình.

Diagram

Description automatically generated with medium confidence***- void Machine::OneInstruction(Instruction \*instr)***để thực hiện một chỉ dẫn hoặc chương trình từ user mode với các diều kiện switch-case tương ứng. (Như là Add ( thêm vào thanh ghi ), Syscall ( System call ) , XOR , còn lại là trả về giá trị FALSE, ASSERT là syscall exit vì machine->Run ko có return )

* ***Void Machine::RaiseException(ExceptionType which, int badVAddr)*** 
  + *Interrupt->setStatus(SystemMode);* để ngắt system mode hiện tại rồi chuyển qua system mode khác (ở đây là chuyển từ User mode sang Kernel mode ).
  + *ExceptionHandler(which)* để chuyển sang dùng hàm void ExceptionHandler với ExceptionType là biến which.
  + *Interrupt->setStatus(UserMode);* cũng tương tự là chương trình về lại User mode thôi.
* ***Void ExceptionHandler(ExceptionType which)***: hàm này để thực thi system call.
  + *Int type = machine->ReadRegister(2);* để class machine đọc thanh ghi số 2 ( $2 chỗ addiu $2,$0,SC\_Halt ) .
  + Điều kiện *if((which == SyscallException) && (type == SC\_Halt ) )* để xét điều kiện với *which* và *type* sau đó nếu thỏa thì chạy hàm *interrupt->Halt()*
  + Điều kiện *else* : in ra màn hình rồi trả về giá trị FALSE.

**-Sơ đồ cách thức hoạt động của quá trình trên được mô tả như hình sau:**

Diagram

Description automatically generated

3. Cấu hình và cài đặt Nachos:

* Yêu cầu để cài đặt : Ubuntu 14.04 32-bit trên Linux(ưu tiên Redhat 9 hoặc Fedora core 3)
* Các bước cài đặt nachos:
  1. Cài đặt gói gcc/g++ bằng cách vào terminal và nhập lệnh ‘sudo apt -get install gcc’ và ‘sudo apt-get install g++’
  2. Unzip nachos.zip đã cho trên moodle vào một thư mục để xử lý trong thư mục đó
  3. Mở thư mục code và sửa file Makeup (‘gmake’ thành ‘make’) trong nachos/nachos-3.4/code
  4. Trong terminal nhập đường dẫn vào thư mục ‘code’ vào và dùng lệnh ‘make’ để biên dịch file
  5. Chạy thử chương trình :…/nachos-3.4/test/halt

\*Bỏ qua bước cài đặt cross compiler vì trong file Nachos thầy cho đã được cấu hình cross-compiler-mips

4. Khai báo biến toàn cục gSynchConsole:

Dùng synchConsole để hỗ trợ việc nhập/xuất trên console. Copy 2 file sysconsole.c và sysconsole.h vào folder threads.

Khai báo biến toàn cục gSynchConsole trong file system.h và cấp phát bộ nhớ và delete nó :

**system.h:**  
  
#ifdef USER\_PROGRAM  
#include "machine.h"  
#include "synchcons.h"  
extern Machine\* machine   
**extern SynchConsole\* gSynchConsole;**    
#endif

**system.cc**  
  
#ifdef USER\_PROGRAM   
Machine \*machine;   
**SynchConsole \*gSynchConsole;**  
#endif

#ifdef USER\_PROGRAM  
    machine = new Machine(debugUserProg);   
 **gSynchConsole = new SynchConsole();**  
#endif

#ifdef USER\_PROGRAM  
    delete machine;  
 **delete gSynchConsole;**  
#endif

5. Các bước cài đặt System Call:

**Bước 1:**

/code/userprog/syscall.h

Bắt đầu với file syscall.h (userprog folder)

Define và khai báo các system call (SC)

**Bước 2:** Ta viết lại cấu trúc điều khiển của để nhận các system call (SC) trong file exception.cc

/code/userprog/exception.cc sửa điều kiện if thành switch.. case

**Bước 3:** Ta khai báo tất cả các SC trong **start.c và start.s**

//start.c và //start.s

Chèn vào đoạn mã :

.globl <tên hàm>  
 .ent <tên hàm>  
<tên hàm>:  
 addiu $2, <tên Systemcall>  
 syscall  
 j $31  
 .end ReadInt

**Bước 4:** Viết chương trình ở mức người dùng để kiểm tra file .c ./code/test

Sử dụng hàm như đã khai báo prototype ở /code/userprog/syscall.h

**Bước 5:** Khai báo trong Makefile (test folder) để nachos biên dịch file .c này thành các file chạy trên HĐH này (file.o .coff). Mở makefile và khai báo tại:

all: halt shell matmult sort (tên file chương trình ở ./test)

Thêm đoạn sau phía sau malmult :  
<tên file>o: <tên file>.c  
$(CC) $(CFLAGS) -c .c : .o start.o  
$(LD) $(LDFLAGS) start.o .o -.coff  
../bin/coff2noff .coff

Đối với các chương trình test khác chỉ việc đổi tên.

**Bước 6:** Biên dịch lại nachos, cd tới ./nachos/code chạy lên “make all”

**Bước 7:** Chạy thử chương trình: ./userprog/nachos -rs 1023 -x ./test/<tên chương trình>

6. Hiểu các mã chương trình Nachos cho trước :

* progtest.cc: kiểm tra thủ tục để biết rằng Nachos có thể load chương trình người dùng và thực thi nó. Khi khởi động nachos và yêu cầu chạy một chương trình bằng nachos -x, nó sẽ bắt đầu một quá trình thử sử dụng hàm *StartProcess*  trong protest.cc, ngoài ra còn chứa một số hàm có ích cho việc I/O
* syscall.h: Lưu các syscall mà người dùng có thể gọi, trong đó có các syscall cho việc thao tác*(SC\_Exit, SC\_Exec, SC\_Join)*, các syscall cho việc thao tác trên tập tin *(SC\_CreateFile, SC\_Open, SC\_Read, SC\_Write, SC\_Close)* và syscall cho multithreading *(SC\_Fork, SC\_Yield)*
* exception.cc: xử lý các syscall và các exception khác như lỗi, trong phần note còn có kèm quy định sử dụng thanh ghi:
  + Thanh ghi r2: chứa system call
  + Thanh ghi r4: chứa tham số thứ nhất
  + Thanh ghi r5: chứa tham số thứ hai
  + Thanh ghi r6: chứa tham số thứ ba
  + Thanh ghi r7: chứa tham số thứ tư
  + Kết quả thực hiện của system call cũng sẽ được lưu ở thanh ghi thứ2
* bitmap.\*: gồm các thủ tục để quản lý bitmap trong đó gồm các hàm để thao tác trên bitmap(*Bitmap,Mark,Clear,Test,Find,NumClear,Print)*, bên cạnh đó còn lưu 2 hàm để đọc/ viết bitmap từ 1 file nachos*(FetchFrom,WriteBack)*
* filesys.\*:
  + gồm các thủ tục để quản lý hoạt động của hệ thống file sao cho:
    - Mỗi file gồm:
      * 1 file header(được lưu ở 1 sector trên đĩa)
      * Số lượng của khối dữ liệu
      * Một cổng vào thư mục hệ thống file.
    - Một hệ thống file bao gồm nhiều cấu trúc dữ liệu:
      * Cấu trúc dữ liệu bitmap với các disk sector trống
      * Cấu trúc dữ liệu thư mục với tên file và file headers.
  + Đối với các quá trình (Như là *Create/Remove/…*) điều chỉnh thư mục cùng hoặc không cùng với bitmap, nếu mà quá trình đó thành công, các thay đổi sẽ được ghi thẳng vào ổ đĩa trong khi các file vẫn còn mở. Nếu quá trình thay đổi thất bại do có vấn đề phát sinh, mặc dù chúng ta đã chỉnh sửa một phần của thư mục/bitmap, những thay đổi đó sẽ được thay xóa mà không lưu vào ổ đĩa
* openfile.h: gồm cấu trúc dùng để mở, đóng, đọc và viết trên một file
* translate.\*: Gồm các thủ tục để dịch địa chỉ ảo sang địa chỉ vật lý bằng cách đọc/viết từng bytes từ/vào bộ nhớ ảo bằng cách sử dụng một trong hai phương pháp ‘Linear Page Table’ hoặc là ‘Translation Lookaside buffer’, nếu có vấn đề phát sinh trong khi dịch thì sẽ RaiseException để xử lý
* machine.\*: Gồm cấu trúc dữ liệu để mô phỏng việc thi hành của các chương trình của người dùng đang chạy trên nachos, có thể hiện tình trạng của CPU và debug khi có phát sinh xảy ra
* mipssim.cc: Mô phỏng lại tập lệnh của MIPS R2/3000 processor
* console.\* gồm các cấu trúc dữ liệu dùng để mô phỏng lại hành vi của một thiết bị cuối cùng để nhập/xuất bao gồm một bàn phím và một màn hình bằng cách sử dụng UNIX files
* synchconsole.\*: Gồm cấu trúc dữ liệu chứa các hàm dùng để quản lý việc nhập/xuất trên Nachos
* …/test/\*:
  + halt.c: Dừng hệ điều hành lại
  + shell.c: Tạo shell
  + matmult.c:Tạo 3 ma trận a b c với kích thước [20][20] và nhân chúng lại với nhau
  + start.c: chạy chương trình

7. Ý tưởng xây dựng và cài đặt các systemcall và exception :

a. Cài đặt các exception :

**char\* User2System(int virtAddr, int limit):**Hàm thực hiện chuyển một chuỗi được lưu trong vùng nhớ của chương trình chạy trên NachOS/MIPS (USER MODE) vào vùng nhớ của hệ điều hành NachOS (KERNEL MODE)

**int System2User(int virtAddr, int len, char\* buffer):**Hàm thực hiện chuyển một chuỗi được lưu trong hệ điều hành NachOS (KERNEL MODE) vào bộ nhớ của chương trình ứng dụng chạy trên NachOS/MIPS (USER MODE).

**-Cài đặt hàm void IncreasePC():** 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.

**-Cài đặt System Call SC\_ReadInt:** đọc 1 số từ console

Tạo một mảng ký tự buffer để chứa từng chữ số trong số đã đọc được từ console. Kiểm tra số âm hay số dương (Ký tự đầu của buffer là – là số âm) thì bắt đầu chuyển vị trí xét số đầu tiên thành 1, ngược lại thì vị trí xét số đầu tiên là 0.

Kiểm tra rằng số nguyên đã nhập có hợp lệ hay không:

+ Nếu số có tồn tại chữ cái lập tức kết thúc hàm.

+ Nếu số nhập vào là số thập phân, nếu các chữ số thập phân toàn bộ là 0 thì là hợp lệ, còn ngược lại sẽ không là số nguyên hợp lệ.

Tiến hành chuyển đổi từ mảng ký tự đã kiểm tra thành số nguyên bằng cách lặp 1 số 0 nhân cho 10 và cộng cho giá trị của chữ số nguyên đó từ bảng ASCII, sau đó kiểm tra lại nếu là số âm thì nhân cho -1, và ghi số đã chuyển vào thanh ghi số 2.

**- Cài đặt System Call SC\_PrintInt:** In một số nguyên ra console

Tạo một mảng ký tự buffer để chứa từng chữ số trong số đã đọc được từ console.

Kiểm tra số đó là số âm hay dương, nếu âm thì cho ký tự buffer đầu tiên là ‘-’, đếm số lượng số trong số đó. Đếm số lượng chữ số trong số đọc được.

Chuyển dãy số đó sang một chuỗi ký tự với độ lớn là số lượng số đã đếm trước đó bằng cách xét số đó từ trái sang phải để đưa vào buffer theo vòng lặp với sự kết hợp của hàm pow (hàm lũy thừa). Sau cùng sẽ ghi buffer vào console.

Ý tưởng: Lấy số đã đọc được từ console chia cho 10 mũ số số hạng – 1 để lấy được số từ trong số đã đọc được, chia lấy dư để lấy được số cuối trong số đã chia, chuyển qua số thứ tự trong bảng ASCII và đưa vào buffer dưới dạng char rồi trừ dần số số hạng cho 1. Sau đó lặp lại cho đến khi hết số số hạng.

**- Cài đặt System Call SC\_ReadChar:** Đọc 1 ký tự từ console

Kiểm tra xem người dùng có nhập sai cách hay ko (nhập quá 1 ký tự hoặc không nhập gì). Nếu có sẽ tiến hành ghi gái trị 0 vào thanh ghi số 2, ngược lại sẽ tiến hành ghi ký tự đọc được vào thanh ghi số 2.

**- Cài đặt System Call SC\_PrintChar:** In ra kí tự ra console

Đọc ký tự từ thanh ghi số 4 và ghi lại console ký tự đó.

**- Cài đặt System Call SC\_ReadString:** Đọc chuỗi ký tự từ console

Đọc từ thanh ghi và lưu vào buffer chuỗi bằng cách sử dụng địa chỉ và độ dài trong thanh ghi số 4 và số 5, copy chuỗi từ vùng nhớ user space sang system space, trả lại độ dài bằng hàm Read rồi copy chuỗi từ vùng nhớ User space sang lại user space, xóa buffer rồi IncreasePC.

**- Cài đặt System Call SC\_PrintString:** In ra chuỗi ra console

Đọc từ thanh ghi và lưu vào buffer chuỗi bằng cách copy từ vùng nhớ User Space sang System Space với bộ đệm buffer dài 200 ký tự sử dụng bộ nhớ ảo đọc từ thanh ghi số 4, đọc độ dài và in ra chuỗi đó, sau đó xóa buffer và IncreasePC.

b. Các chương trình ở file test:

- Viết các file có mã nguồn C của các chương trình  trong thư mục nachos/nachos-3.4/code/test:

- **ascii.c:**

- In ra bảng ASCII từ 32 tới 127 ,vì không thể đọc giá trị dưới 31 và 128

- **help.c:**

- In ra thông tin nhóm và chương trình

- **readchar.c, readint.c, readstring.c:**

- Đọc kí tự/số/chuỗi vừa nhập từ thanh ghi

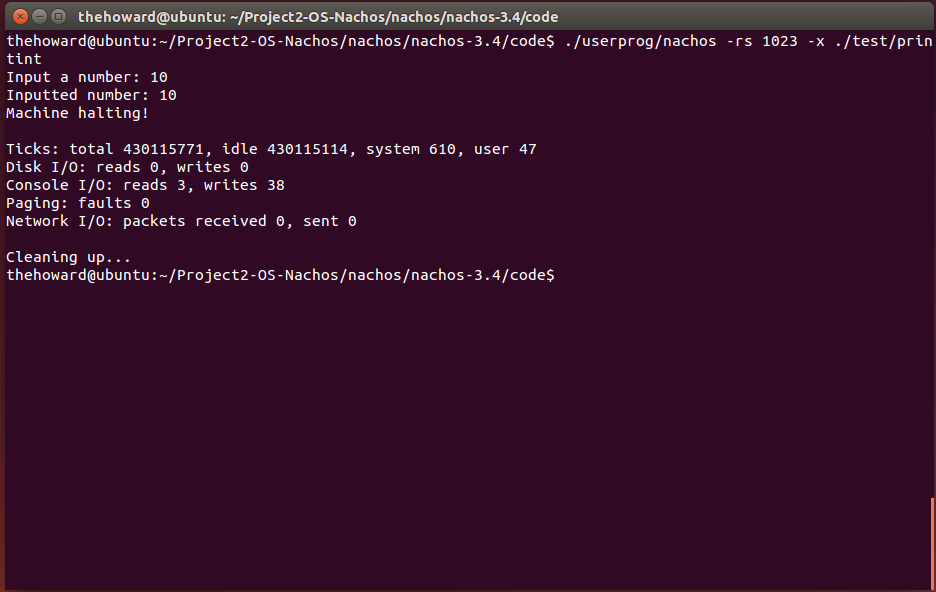
- **printchar.c, printint.c, printstring.c:**

- Yêu cầu người dùng nhập và xuất ra console kí tự/số /chuỗi đã nhập

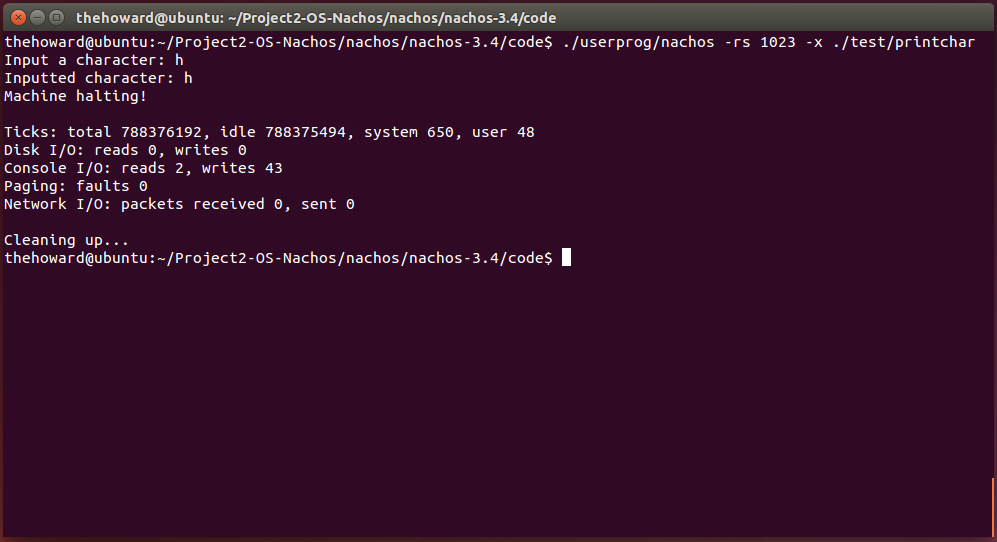
- **sort.c:** tạo 1 mảng 1 chiều a[101] với các giá trị bị xếp ngược, ta xắp xếp lại sử dụng biến tạm thời tmp

8. Demo thử hàm thiết kế:

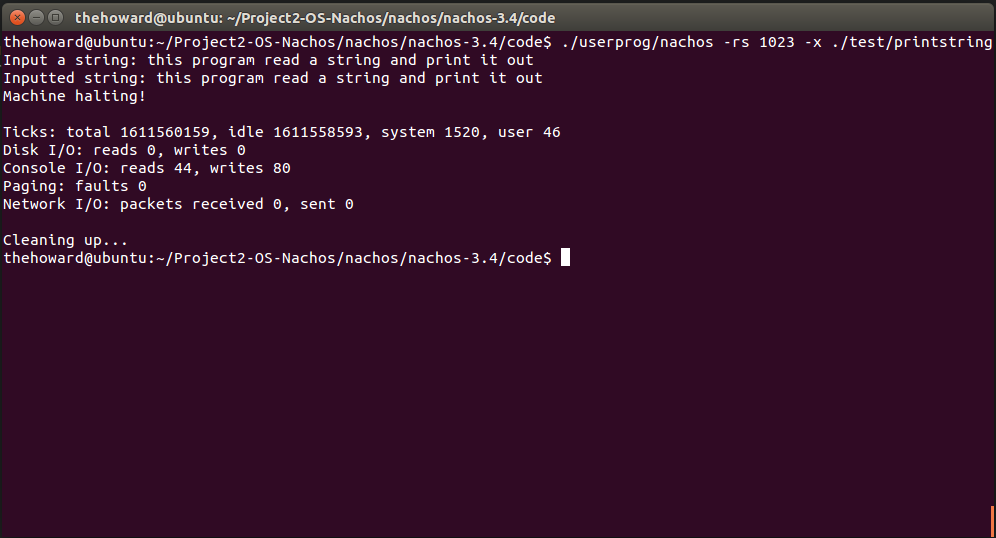
-Hàm : **PrintInt** In một số nguyên do user nhập



-Hàm : **PrintChar** In ra kí tự do user nhập



-Hàm : **PrintString** In ra chuỗi do user nhập



9. Tài liệu tham khảo:

- 4 file PDF của thầy

- Các link tham khảo khác:

<https://dotrungduchd.blogspot.com/2013/05/system-call-of-nachos.html>

<https://programmersought.com/article/43283381421/>

+ Lập trình Syscall trên Nachos

<https://dangkhoahome.blogspot.com/2013/03/trong-file-huong-dan-cua-thay-khong.html>

<https://www.youtube.com/playlist?list=PLRgTVtca98hUgCN2_2vzsAAXPiTFbvHpO>