Vietnam National University – Ho Chi Minh City
University of Science – Information Technology (High Quality Program)

EXCEPTIONS AND SYSTEM CALLS NACHOS

INTRODUCTION TO OPERATING SYSTEM

PROJECT 02

19CLC6

LECTURERS:

Mr. Nguyen Van Giang

Mr. Le Viet Long

Mục lục

I. BẢNG PHÂN CÔNG CÔNG VIỆC

Danh sách các công việc	Phân bổ công việc		
	Ngô Huy Anh **19127095**	Ngô Đăng Khoa **19127444**	Huỳnh Cao Nhật Hiếu **19127399**
Viết lại file exception.cc	✓		
Viết lại cấu trúc điều khiển chương trình để nhận Nachos system calls		✓	
Viết function tăng Program Counter			✓
System call ReadInt + PrintInt	✓		
System call ReadChar + PrintChar		✓	
System call ReadString + PrintString			✓
Viết chương trình help, ascii, sort	✓		
Viết báo cáo	✓	✓	✓
Demo	✓		
Đã hoàn thành (hiện tại)	**100%**		

II. CƠ CHẾ THỰC HIỆN CHƯƠNG TRÌNH TRÊN NACHOS

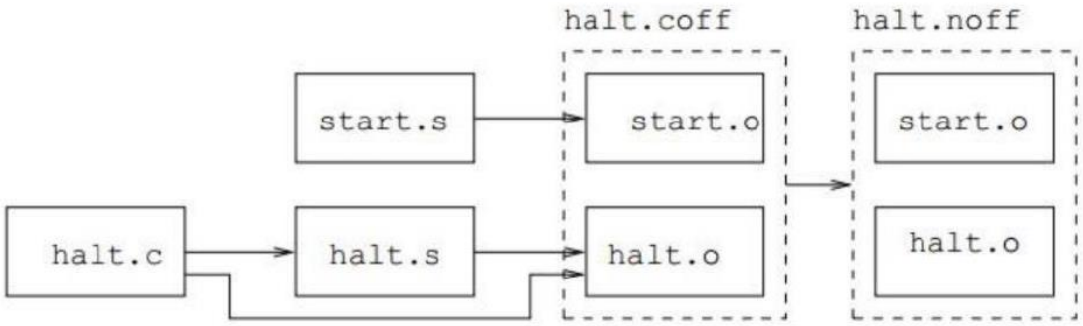
Một chương trình (halt.c) muốn được thực thi thì nó cần phải được biên dịch. Quá trình biên dịch trên NachOS gồm ba bước:

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

Quá trình thực thi được mô tả bằng hình dưới đây:



III. CÀI ĐẶT TỔNG QUAN

1) Cài đặt NachOS trên Linux

Bước 1: Cài đặt Linux trên máy ảo

Bước 2: Giải nén file nachos.zip

Bước 3: Mở file Makefile ở thư mục **test (./nachos/nachos-3.4/code.test),** sửa dòng "MAKE = gmake" thành "MAKE = make" rồi lưu lại.

Bước 4: Mở Terminal. Dùng lệnh cd dẫn vào **/nachos/nachos-3.4/code**, sau đó gõ make all để tiến hành cài đặt NachOS.

Bước 5: Chạy thử chương trình halt trên NachOS bằng lệnh : ./userprog/nachos -rs 1023 -x ./test/halt

Chương trình hiện ra như thế này thì máy ảo đã cài thành công

```
Shutdown, initiated by user program.Machine halting!

Ticks: total 42, idle 0, system 30, user 12
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
```

2) Cài đặt kiến trúc MiPS cho NachOS

Sau khi giải nén file nachos.zip, ngay trong thư mục nachos vừa giải nén có 2 file **synchcons.cc và synchcons.h**. Tiến hành copy 2 file đó vào thư mục **threads(nachos/nachos-3.4/code/threads),** cũng trong thư mục đó, mở file system.cc và khai báo thêm lớp SynchConsole, đồng thời thêm dòng tạo và delete SynchConsole

```
#ifdef USER_PROGRAM    // requires either FILESYS or
FILESYS_STUB
Machine *machine;      // user program memory and registers
SynchConsole *gSynchConsole;
#endif
```

```
#ifdef USER_PROGRAM
    machine = new Machine(debugUserProg);     // this must come first
    gSynchConsole = new SynchConsole();
#endif
```

```
#ifdef USER_PROGRAM
    delete machine;
    delete gSynchConsole;
#endif
```

3) Các bước cài đặt và tạo một System Call

Bước 1: Tìm đến file **syscall.h (./code/userprog/syscall.h)**, define system call để có thể dùng trong Switch-Case và khai báo prototype cho hàm.

Bước 2: Tìm đến file **start.c và start.s (./code/test/start.c)**, thêm đoạn mã lệnh sau Yield và trước __main, với *name* là tên của hàm đã khai báo prototype

```
        .globl *name*
        .ent *name*
*name*:
        addiu $2,$0,SC_*name*
        syscall
        j       $31
        .end *name*
```

Bước 3: Tìm đến file **exception.cc (./code/userprog/exception.cc)**, chuyển điều kiện If thành Switch Case, sau đó thêm SC_*name* vào hàm **void ExceptionHandler(ExceptionType which)** và viết lệnh thực thi.

```
void ExceptionHandler(ExceptionType which)
{
        case SC_PrintInt:
        {
                int number = machine->ReadRegister(4);
                if(number == 0) {
                        gSynchConsole->Write("0", 1);
                        IncreasePC();
                        return;
                        }
        }
…
```

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: Tìm đến file Makefile ở ./code/test/Makefile để thêm tên chương trình (tên file) vào dòng all

```
all: halt shell matmult sort CheckNachos string help char test bubblesort ascii
```

Và thêm đoạn phía sau matmult

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

Bước 6: Biên dịch lại NachOS, cd tới ./nachos/code chạy lệnh "make all"

Bước 7: Chạy thử chương trình: ./usergrog/nachos -rs 1023 -x ./test/*file_name*

IV. CÀI ĐẶT CÁC SYSTEM CALL VÀ EXCEPTION

1) Viết lại cấu trúc chương trình và cài đặt lại các exception

Toàn bộ các exceptions đều được liệt kê trong file machine/machine.h.

Chương trình sẽ dùng cấu trúc điều kiện Switch Case để phân loại và xử lý các exceptions.

Khi chương trình gặp trường hợp "No Exception": Chương trình sẽ trả quyền về cho hệ điều hành bằng cách return về.

Khi gặp trường hợp "Syscall Exceptions": Chương trình sẽ tiếp tục dùng cấu trúc Switch Case để xử lý các hàm đã được viết cho user system calls.

Còn khi gặp các trường hợp còn lại được đề cập trong machine.h: Chương trình sẽ hiển thị thông báo lỗi lên trên console cho người dùng và halt hệ thống.

Cuối cùng, nếu không xác định được exception đang xảy ra: Chương trình sẽ hiển thị thông báo không xác định được lỗi lên trên console.

2) Cài đặt hàm User2System() và System2User()

- **Mục đích:** để sao chép vùng nhớ từ user sang system và ngược lại.

```
61  char* User2System(int virtAddr,int limit)
62  {
63      int i;// index
64      int oneChar;
65      char* kernelBuf = NULL;
66      kernelBuf = new char[limit +1];//need for terminal string
67      if (kernelBuf == NULL)
68         return kernelBuf;
69      memset(kernelBuf,0,limit+1);
70      for (i = 0 ; i < limit ;i++)
71      {
72         machine->ReadMem(virtAddr+i,1,&oneChar);
73         kernelBuf[i] = (char)oneChar;
74         if (oneChar == 0)
75         break;
76      }
77      return kernelBuf;
78  }
```

- Hàm User2System truyền vào địa chỉ chỉ user và giới hạn của buffer và sẽ trả về một kernel buffer là một bộ nhớ đệm buffer(char*)

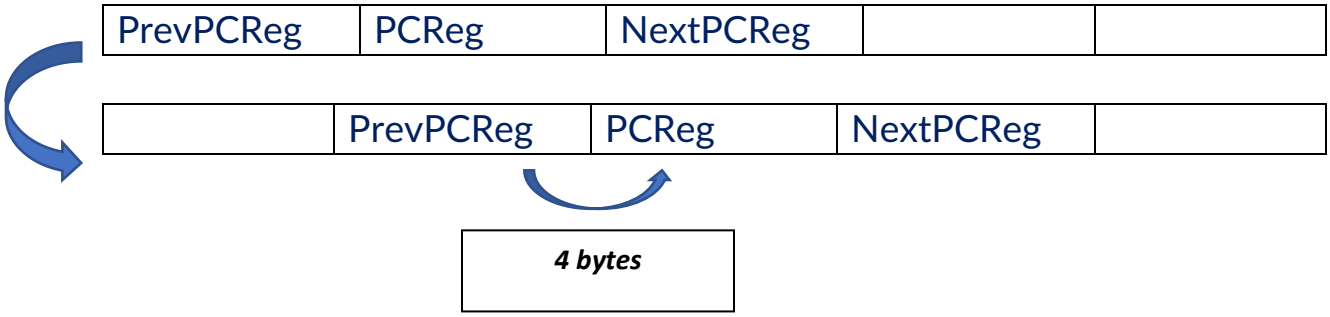
```
80  int System2User(int virtAddr,int len,char* buffer)
81  {
82      if (len < 0) return -1;
83      if (len == 0)return len;
84      int i = 0;
85      int oneChar = 0 ;
86      do{
87         oneChar= (int) buffer[i];
88         machine->WriteMem(virtAddr+i,1,oneChar);
89         i ++;
90      }while(i < len && oneChar != 0);
91      return i;
92  }
```

- Còn đối với System2User thì ngược lại, hàm truyền vào địa chỉ ảo, giới hạn buffer và bộ nhớ đệm của buffer và trả về i là số bytes đã sao chép được từ user. Từ đó ta có thể sao chép được vùng nhớ từ System sang cho User.

3) Cài đặt hàm IncreasePC()

Mục đích của hàm IncreasePC là dùng để tăng Program Counter(PC) để tiến tới lệnh tiếp theo và nạp lệnh đó vào.

Hàm sẽ xử lí việc đọc và lưu giá trị của PC hiện tại cho PC trước, nạp giá trị kế tiếp cho PC hiện tại và nạp giá trị kế tiếp nữa cho PC kế. Trong đó, khoảng cách giữa 2 PC là 4 bytes.



4) Cài đặt system call Read Int – int ReadInt()

System call này sẽ sử dụng lớp SynchConsole để đọc một số nguyên do người dùng nhập vào. Nếu giá trị người dùng nhập không phải là số nguyên thì sẽ return về 0.

Do cần phải kiểm tra input của user nên phải tạo mảng char để chứa input. Sau đó sử dụng hàm Read của lớp SynchConsole để lưu input vào mảng cho đồng thời lấy được độ dài của mảng.

Chạy vòng lặp để ghi nhớ vị trí bắt đầu và kết thúc của input (Trường hợp 123.0000 – input này vẫn xem là số nguyên, nên cần phải ghi nhớ vị trí bắt đầu và kết thúc) đồng thời kiểm tra xem input đầu vào có kí tự bất không thỏa hay không.

Trường hợp input không phải là số nguyên:

- Chương trình sẽ hiển thị thông báo lỗi lên trên console, sau đó dùng lệnh WriteRegister ở lớp Machine để ghi vào thanh r2 giá trị 0.
- Làm tăng Program Counter để nạp lệnh tiếp theo.

Trường hợp input là số nguyên:

- Tính toán lại số nguyên input từ mảng char đã được kiểm tra, sau đó dùng lệnh WriteRegister ở lớp Machine để ghi vào thanh r2 giá trị đã tính toán.
- Làm tăng Program Counter để nạp lệnh tiếp theo.

5) Cài đặt system call Print Int – void PrintInt(int number)

System call này sẽ sử dụng lớp SynchConsole để xuất một số nguyên được truyền vào tham số ra màn hình.

Đầu tiên, dùng lệnh ReadRegister ở lớp Machine để đọc địa chỉ của thanh ghi r4.

Trường hợp số nhận được là số 0:

- Sử dụng lớp SynchConsole để Write "0" ra console cho user.
- Làm tăng Program Counter để nạp lệnh tiếp theo.

Trường hợp số nhận được không phải là số 0:

- Kiểm tra xem số nhận được là dương hay là âm
- Sử dụng mảng char để lưu từng chữ số của tham số number vào
- Sử dụng lớp SynchConsole để Write mảng char ra console cho user.
- Làm tăng Program Counter để nạp lệnh tiếp theo.

6) Cài đặt system call Read Char – char ReadChar()

System call này sẽ sử dụng lớp SynchConsole để đọc một kí tự(char) do người dùng nhập vào.

Sau đó, `int numBytes = gSynchConsole->Read(buffer, maxBytes)` sẽ làm nhiệm vụ đếm số kí tự mà người dùng nhập vào từ việc sử dụng lớp SynchConsole để đọc.

Ở đây, ta phân ra thành 3 trường hợp mà người dùng có thể nhập vào:

- Người dùng nhập vào số kí tự nhiều hơn 1 (`numBytes > 1`) thì thông báo cho người dùng là lỗi và chỉ được nhập 1 kí tự duy nhất.

- Nếu người dùng không nhập gì (*numBytes == 0*) thì thông báo là kí tự rỗng và thông báo lỗi.
- Cuối cùng là người dùng nhập chính xác 1 kí tự thì ta lấy kí tự đó và ghi kí tự đó vào thanh ghi R2(Lưu mã syscall đồng thời lưu kết quả trả về của mỗi syscall nếu có).

Sau đó ta tăng PC lên và kết thúc.

7) Cài đặt system call Print Char – void PrintChar(char character)

System call này sẽ sử dụng lớp SynchConsole để xuất một số kí tự được truyền vào tham số ra màn hình.

Đầu tiên, dùng lệnh ReadRegister ở lớp Machine để đọc địa chỉ của thanh ghi r4.

Sau đó, hàm Write trong lớp SynchConsole sẽ dùng để in kí tự được truyền vào ra màn hình.

Việc cuối cùng là tăng PC lên và kết thúc.

8) Cài đặt system call Read String – void ReadString(char buff[], int length)

System call này sẽ sử dụng lớp SynchConsole để đọc một chuỗi kí tự do người dùng nhập vào. Đầu tiên system call này sẽ lấy giá trị địa chỉ ở thanh ghi số 4 và độ dài của chuỗi ở thanh ghi số 5. Sau đó thực hiện chép giá trị địa chỉ đã đọc ở thanh ghi số 4 từ vùng nhớ User sang System bằng hàm **User2System()**. Giá trị chép được là buffer chuỗi kí tự. Ta gọi phương thức Read của lớp **SynchConsole** đọc buffer với độ dài length, trả về số byte đã đọc thực sự cho thanh ghi số 2 và chép buffer từ phía System sang User bằng hàm **System2User().**

9) Cài đặt system call Print String – void PrintString(char buff[])

System call này sẽ sử dụng lớp SynchConsole để xuất một chuỗi kí tự được truyền vào tham số ra màn hình. Đầu tiên, dùng lệnh ReadRegister ở lớp Machine để đọc địa chỉ của thanh ghi r4. Sau đó thực hiện chép giá trị địa chỉ đã đọc ở thanh ghi số 4 từ vùng nhớ User sang System bằng hàm **User2System()**. Tiếp theo chạy một vòng lặp while để lấy được kích thước của chuỗi. Cuối cùng sử dụng hàm Write của lớp SynchConsole in giá trị ra màn hình.

V. CÀI ĐẶT CÁC CHƯƠNG TRÌNH YÊU CẦU

1) Chương trình Help

Chương trình sẽ gọi lại system call PrintString(char buff[]) đã được đề cập ở phần V để in ra dòng giới thiệu cơ bản về nhóm và mô tả vắn tắt về hai chương trình ascii và bubble sort.

Cuối chương trình sẽ gọi system call Halt().

2) Chương trình Ascii

Chương trình sẽ dùng vòng lặp để chạy từ 0 đến 127.

Sử dụng lại system call PrintInt(int number) để in số ở hệ thập phân của ký tự.

Sử dụng lại system call PrintChar(char character) để in ký tự.

Cuối chương trình sẽ gọi system call Halt().

3) Chương trình Bubble Sort

Chương trình sẽ sử dụng lại system call ReadInt() cho người dùng nhập vào độ dài của mảng đồng thời kiểm tra tính đúng đắn.

Sử dụng lại system call ReadInt() để đọc từng phần tử được user nhập vào của mảng.

Sử dụng lại system call PrintInt(int number) để in ra từng phần tử của mảng.

Cuối chương trình sẽ gọi system call Halt().

VI. DEMO CÁC SYSTEM CALL VÀ CHƯƠNG TRÌNH

1) System Call Read Int và Print Int – Test tất cả trường hợp

https://youtu.be/PhC3zdGJnIQ

2) System Call Read Char và Print Char – Test tất cả trường hợp

https://youtu.be/dv9ojca4-sk

3) System Call Read String và Print String – Test tất cả trường hợp

https://youtu.be/dyHsP9ivybc

4) Tổng thể đồ án

https://youtu.be/6ijmuv3o8e4

VII. THAM KHẢO

1) **Giáo trình Hệ Điều Hành** – Tác giả: Thầy Trần Trung Dũng, Thầy Phạm Tuấn Sơn – Nhà xuất bản Khoa học và Kỹ thuật – Chỉnh sửa và bổ sung năm 2019.
2) **Bộ Video hướng dẫn Lập trình NachOS HCMUS của Thầy Lê Viết Long** – Giảng viên bộ môn Hệ Điều Hành lớp 19CLC6 – Tài liệu được upload trên moodle môn học
3) **Bộ Video hướng dẫn Lập trình NachOS HCMUS** – Tác giả: Thành Chung Nguyễn – Link đính kèm: https://youtube.com/playlist?list=PLRgTVtca98hUgCN2_2vzsAAXPiTFbvHpO
4) **Tài liệu tham khảo hướng dẫn lập trình NachOS HCMUS** – Tác giả: Nguyễn Thành Chung – Link đính kèm: https://github.com/nguyenthanhchungfit/Nachos-Programing-HCMUS
5) **Tài liệu tham khảo hướng dẫn lập trình NachOS HCMUS** – Tác giả: ngankhanh98 – Link đính kèm: https://github.com/ngankhanh98/nachos
6) **Tài liệu tham khảo hướng dẫn lập trình NachOS HCMUS** – Tác giả: Dang Khoa – Link đính kèm: http://dangkhoahome.blogspot.com/p/nachos.html