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

BÁO CÁO ĐỒ ÁN HỆ ĐIỀU HÀNH

ĐỒ ÁN 2

Nhóm sinh viên :
Trương Văn Chí - 21127021
Nguyễn Gia Hưng - 21127290
Lê Hửu Trọng - 21127190
Lê Thanh Khôi Nguyên - 21127373
Phan Văn Nguyên - 21127121

GV hướng dẫn:
Lê Viết Long
Phạm Tuấn Sơn

April 3, 2023

Mục lục

Danh sách hình

1 Bảng phân công

Yêu cầu / Họ tên	**Công việc**	**Đánh giá**	**Ghi chú**
Văn Chí	Câu a,b,c	100%	
Gia Hưng	Report	100%	
Hửu Trọng	Câu d,e,f	100%	
Khôi Nguyên	Câu g,h,i	100%	
Văn Nguyên	Câu j, k ,l	100%	

2 Tìm hiểu Nachos

2.1 Nachos là gì ?

Tên thật là Not Another Completely Heuristic Operating System - Một phần mềm giả lập hệ điều hành với kiến trúc MIPS (Million Instructions Per Second) . Nó giả lập một máy tính ảo và một số thành phần cơ bản của hệ điều hành chạy trên máy tính ảo này nhằm giúp cho việc tìm hiểu và xây dựng các thành phần phức tạp hơn của hệ điều hành.

Các tập tin trong đồ án này:

• **progtest.cc** kiểm tra các thủ tục để chạy chương trình người dùng. Trong file này có hàm **StartProcess** dùng để chạy 1 chương trình và cấp phát vùng nhớ cho chương trình đó. Bên cạnh đó còn có hàm **ConsoleTest** để test màn hình console

• **syscall.h** system call interface: các thủ tục ở kernel mà chương trình người dùng có thể gọi. File này còn cung cấp 1 số define để gọi các system call dễ hơn và 1 số hàm cơ bản để tạo tương tác trên console như Write hoặc Read các loại dữ liệu khác nhau như Int, Char, String

• **exception.cc** xử lý system call và các exception khác ở mức user, ví dụ như lỗi trang, trong phần mã chúng tôi cung cấp, chỉ có 'halt' system call được viết

• **bitmap.*** các hàm xử lý cho lớp bitmap (hữu ích cho việc lưu vết các ô nhớ vật lý) filesys.h. Cụ thể như hàm **Mark()** dùng để cài đặt bit thứ n với **which** là biến được truyền vào chính là số bit cần được cài. Hàm **Clear** dùng để xóa bit thứ n với biến **which** là số bit cần được xóa

• **openfile.h** định nghĩa các hàm trong hệ thống file nachos. Trong đồ án này chúng ta sử dụng lời gọi thao tác với file trực tiếp từ Linux, trong đồ án khác chúng ta sẽ triển khai hệ thống file trên ổ đĩa giả lập. (nếu kịp thời gian)

• **translate.*** Phiên bản nachos chúng tôi gửi các bạn, chúng tôi giả sử mỗi địa chỉ ảo là cũng giống hệt như địa chỉ vật lý, điều này giới hạn chúng ta chỉ chạy 1 chương trình tại một thời điểm. Các bạn có thể viết lại phần này để cho phép nhiều chương trình chạy cùng lúc trong đồ án sau.

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

• **mipssim.cc** mô phỏng tập lệnh của MIPS R2/3000 processor

• **console.*** mô phỏng thiết bị đầu cuối sử dụng UNIX files. Một thiết bị có đặc tính (i) đơn vị dữ liệu theo byte, (ii) đọc và ghi các bytes cùng một thời điểm, (iii) các bytes đến bất đồng bộ

• **synchconsole.*** nhóm hàm cho việc quản l y' nhập xuất I/O theo dòng trong Nachos.

• **../test/*** Các chương trình C sẽ được biên dịch theo MIPS và chạy trong Nachos

3 Viết lại file exception.cc

Mô tả chi tiết : Viết lại file **exception.cc** để xử lý tất cả các exceptions được liệt kê trong **machine/machine.h**. Hầu hết các exception trong này là run-time errors, khi các exception này xảy ra thì user program không thể được phục hồi. Trường hợp đặc biệt duy nhất là no exception sẽ trả quyền điều khiển về HĐH, còn syscall exceptions sẽ được xử lý bởi các hàm chúng ta viết cho user system calls. Với tất cả exceptions khác, HĐH hiển thị ra một thông báo lỗi và Halt hệ thống.

Như đã nói ở phần giới thiệu đồ án, file này xử lí các system call và các exception, nhưng chỉ được viết sẵn system call **halt** được viết.

Các exception type được liệt kê trong file **machine.h** bao gồm :

No Exception : Như đã nói ở trên, đây là trường hợp đặc biệt nhất, hệ thống không có vấn đề gì cả, tiến hành trả quyền điều khiển về cho hệ điều hành.

SyscallException : exception type này xử lí dựa vào hàm được viết (chúng ta sẽ tìm hiểu kỹ hơn ở phần sau)

Và các exception khác : PageFaultException , ReadOnlyException , BusErrorException , AddressErrorException , OverflowException , IllegalInstrException . Trước khi đến với hàm hàm **void ExceptionHandler** ta tìm hiểu về công việc cấp phát và khai báo các hàm hỗ trợ gồm **User2system(), System2user()**

Hàm **User2System** và **System2User** dùng để copy vùng nhớ từ User Space vào System Space và ngược lại. Chúng nhận vào 1 địa chỉ và giới hạn của Buffer dạng số nguyên và trả về 1 chuỗi, ở đây là Buffer được copy để thao tác trong các sự kiện tới.

Để xử lí các exception nêu trên, hàm **void ExceptionHandler()** được sử dụng. Trong đó biến **which** được dùng để bắt sự kiện cho từng trường hợp.

NoException : Như đã nói ở trên, đây là trường hợp đơn giản nhất, không có vấn đề nào cần xử lí, quyền điều khiển đơn giản được trả lại cho hệ điều hành.

PageFaultException: đây là lỗi không tìm thấy bộ phiên dịch phù hợp. Trường hợp này chúng ta sẽ xuất ra màn hình "No valid translation found" để báo lỗi, sau đó gọi lại hàm **Halt** dùng để halt hệ thống.

ReadOnlyException : đây là lỗi cố gắng ghi vào 1 trang được gắn nhãn "chỉ đọc". Cũng giống như PageFaultException ở trên, trường hợp này cũng chỉ xuất ra màn hình dòng "Write attempted to page marked read-only" để báo lỗi sau đó halt hệ thống bằng hàm halt.

BusErrorException : đây là lỗi về bus truyền khi mà kết quả của quá trình phiên dịch lại dẫn đến 1 vùng nhớ không hợp lệ. Cũng giống như các lỗi trên, "Translation resulted in an invalid physical address" được in ra màn hình và halt hệ thống.

AddressErrorException : đây là lỗi khi mà tham chiếu chưa được chỉ định hoặc nằm ngoài vùng hợp lệ. Dòng "Unaligned reference or one that was beyond the end of the address space" được xuất ra màn hình để báo lỗi và halt hệ thống.

OverflowException : đây là lỗi tràn số trong việc thực hiện phép tính. Hệ thống in ra màn hình ""Integer overflow in add or sub" và halt hệ thống.

IllegalInstrException : lỗi Instr chưa được thực hiện hoặc đang dự trữ. Hệ thống in ra màn hình "Unimplemented or reserved instr" và halt hệ thống.

NumExceptionType : Số loại exception có vấn đề. Hệ thống in ra "Number exception types" và halt hệ thống.

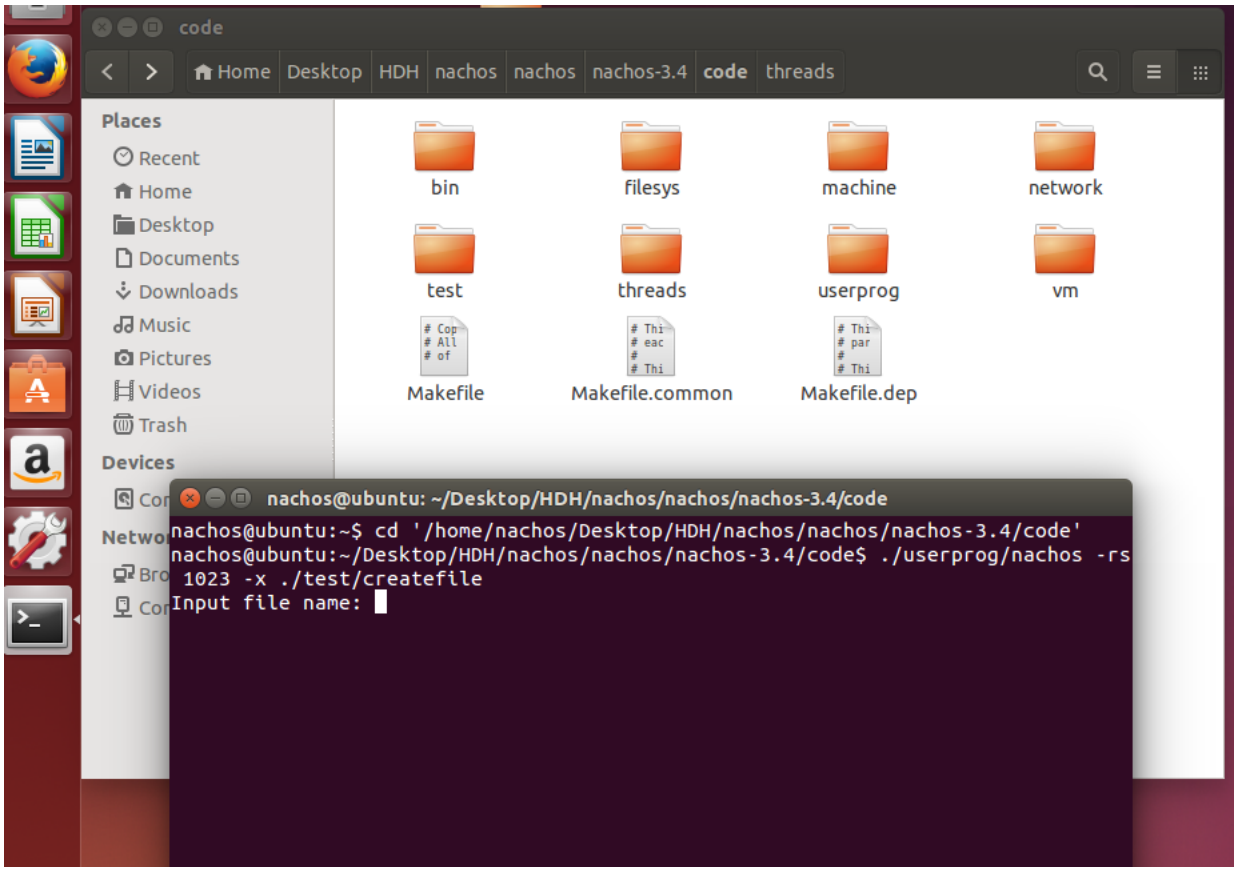
SyscallException : đây là loại Exception đặc biệt khi mà bên trong nó tiếp tục xử lí từng trường hợp để gọi các system call, chi tiết như sau :

Cũng giống như biến which ở trên, biến type được dùng để bắt sự kiện cho từng trường hợp.

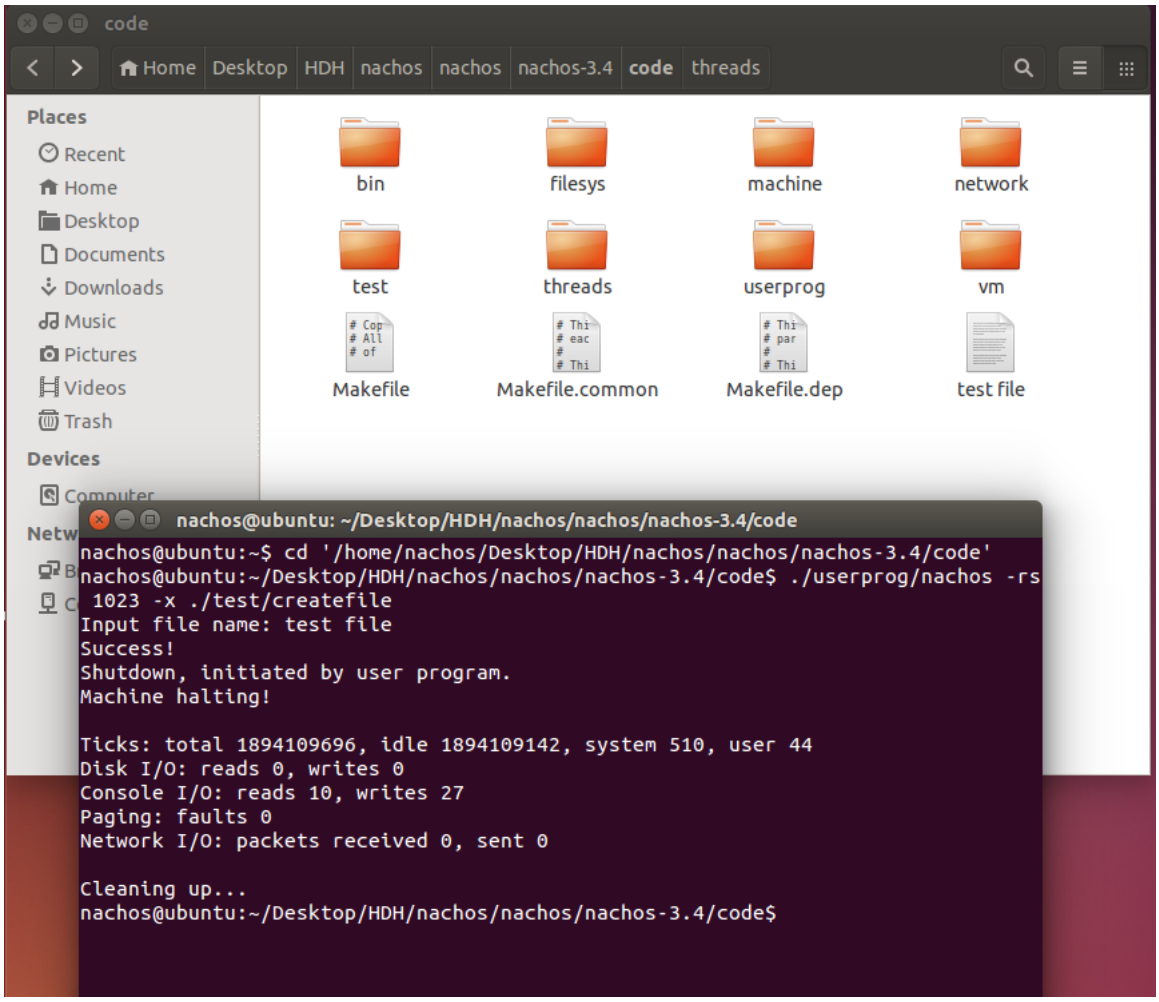
SC_Halt : đây là 1 system call khá đơn giản, được dùng để halt hệ thống. Trước khi tắt hệ thống, dòng

"Shutdown, initiated by user program" được in ra để báo cho người dùng.

SC_Create : đây là 1 system call được dùng để tạo file. Một biến dùng để chứa địa chỉ của file được tạo, nó được đọc từ thanh ghi số 4 (thanh ghi chứa thông tin của tham số thứ 1). Sau đó tên của file được truyền từ User space sang Kernal space bằng hàm **User2System()**. Sau khi check tên của file, nếu tên không hợp lệ hệ thống sẽ báo sau đó hủy file với việc ghi vào thanh ghi r2 (thanh ghi chứa kết quả của system call) giá trị -1,hủy filename sau đó gọi hàm tăng Program counter **IncreasePC()** (hàm này chúng ta sẽ tìm hiểu kỹ hơn trong các phần sau) sau đó return trả quyền điều khiển về cho hệ thống. Nếu không tạo được file thì các thủ tục cũng tương tự. Mặt khác nếu tạo file thành công, giá trị của biến được ghi vào thanh ghi r2 sẽ là 0 sau đó cũng là các thủ tục như trên.



Hình 1: Trước khi tạo file



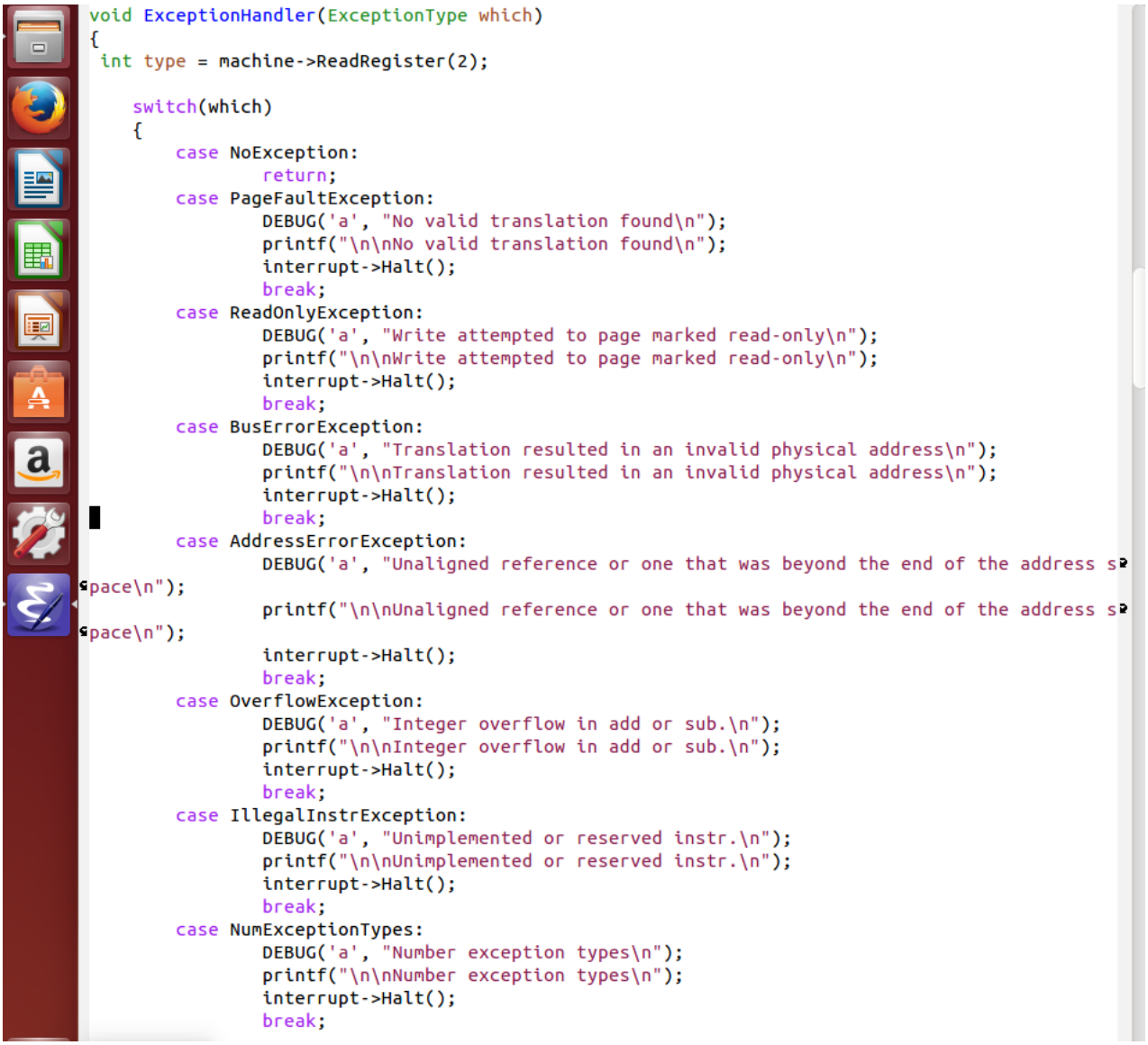
Hình 2: Sau khi tạo file

SC_ReadInt : đây là system call dùng để đọc 1 số nguyên (ta sẽ tìm hiểu kỹ hơn trong các phần sau)
SC_PrintInt : đây là system call dùng để in 1 số nguyên (ta sẽ tìm hiểu kỹ hơn trong các phần sau)
SC_ReadChar : đây là system call dùng để đọc 1 ký tự (ta sẽ tìm hiểu kỹ hơn trong các phần sau)
SC_PrintChar : đây là system call dùng để in 1 ký tự (ta sẽ tìm hiểu kỹ hơn trong các phần sau)
SC_ReadString : đây là system call dùng để đọc 1 chuỗi (ta sẽ tìm hiểu kỹ hơn trong các phần sau)
SC_PrintString : đây là system call dùng để in 1 chuỗi (ta sẽ tìm hiểu kỹ hơn trong các phần sau)

4 Viết lại cấu trúc điều khiển

Yêu cầu chi tiết : Viết lại cấu trúc điều khiển của chương trình để nhận các Nachos system calls. Kiểm tra cấu trúc mới bằng cách dùng system call Halt để kiểm tra tính đúng đắn của cấu trúc mới.

Hàm **ExceptionHandler** mà ta nói tới trong câu a là hàm để nhận và điều khiển các nachos system call. Cấu trúc điều khiển được phân phối bằng câu lệnh switch-case lồng nhau, cuối mỗi case đều được gọi hàm Halt. Các hình sau đây cho thấy sự đúng đắn của cấu trúc mới, khi mà hàm **Halt** được gọi trực tiếp đơn lẻ và gián tiếp qua các system call là như nhau.



```
void ExceptionHandler(ExceptionType which)
{
 int type = machine->ReadRegister(2);

    switch(which)
    {
        case NoException:
                return;
        case PageFaultException:
                DEBUG('a', "No valid translation found\n");
                printf("\n\nNo valid translation found\n");
                interrupt->Halt();
                break;
        case ReadOnlyException:
                DEBUG('a', "Write attempted to page marked read-only\n");
                printf("\n\nWrite attempted to page marked read-only\n");
                interrupt->Halt();
                break;
        case BusErrorException:
                DEBUG('a', "Translation resulted in an invalid physical address\n");
                printf("\n\nTranslation resulted in an invalid physical address\n");
                interrupt->Halt();
                break;
        case AddressErrorException:
                DEBUG('a', "Unaligned reference or one that was beyond the end of the address s
pace\n");
                printf("\n\nUnaligned reference or one that was beyond the end of the address s
pace\n");
                interrupt->Halt();
                break;
        case OverflowException:
                DEBUG('a', "Integer overflow in add or sub.\n");
                printf("\n\nInteger overflow in add or sub.\n");
                interrupt->Halt();
                break;
        case IllegalInstrException:
                DEBUG('a', "Unimplemented or reserved instr.\n");
                printf("\n\nUnimplemented or reserved instr.\n");
                interrupt->Halt();
                break;
        case NumExceptionTypes:
                DEBUG('a', "Number exception types\n");
                printf("\n\nNumber exception types\n");
                interrupt->Halt();
                break;
```

Hình 3: Một phần cấu trúc điều khiển system call

5 Tăng program counter

Yêu cầu chi tiết : Tất cả các system calls (ko phải Halt) sẽ yêu cầu Nachos tăng program counter trước khi system call trả kết quả về. Nếu không lập trình đúng phần này thì Nachos sẽ bị vòng lặp gọi thực hiện system call này mãi mãi. Cũng như các hệ thống khác, MIPS xử lý dựa trên giá trị của program counter, vì vậy bạn phải viết mã để tăng gia trị biến program counter, tìm đoạn mã này trong thư mục machine. Bạn phải copy mã này vào vị trí thích hợp trong phần xử lý các system call của bạn. Hiện tại, bạn phải dùng Halt system call tại cuối mỗi user program.

Hàm **IncreasePC()** sẽ đảm nhận nhiệm vụ yêu cầu Nachos tăng program counter.
Đầu tiên, hàm **ReadRegister()** được sử dụng để đọc thông tin trên thanh ghi pc, biến truyền vào ở đây là **PCReg** (nên nhớ răng hàm **ReadRegister** chỉ nhận tham số truyền vào là 1 số nguyên), lí do cho việc này chính là **PCReg** đã được define là 34 - là địa chỉ của thanh ghi pc.

Sau đó, giá trị lấy được sẽ cộng thêm 4 (tức là giá trị của thanh ghi pc đã được tăng) và truyền vào biến **pcAfter**.

Tiếp đó, giá trị hiện thời của thanh ghi pc được giữ lại trong thanh ghi chứa giá trị trước đó của thanh ghi - 36 (giá trị của thanh được ghi lại này được lưu lại nhằm mục đích debug lỗi)

Sau đó giá trị của thanh ghi chứa giá trị hiện thời được truyền vào giá trị của thanh ghi chứa giá trị của thanh ghi tiếp theo và cuối cùng là cập nhật giá trị của thanh ghi chứa giá trị của thanh ghi tiếp theo bằng biến **pcAfter** đã cập nhật ở đầu. Nói chung hàm này cập nhật giá trị của các program counter bao gồm **PrevReg**, **PCReg** và **NextPCReg** thêm một lượng mà đối với thanh ghi chứa giá trị kế tiếp là 4.

```
void IncreasePC()
{
        //int temp=machine->ReadRegister(PCReg);
        //machine->WriteRegister(PrevPCReg, temp);
        //temp=machine->ReadRegister(NextPCReg);
        //machine->WriteRegister(PCReg, temp);
        //machine->WriteRegister(NextPCReg, temp+4);
        int pcAfter=machine->ReadRegister(PCReg)+4;
        machine->registers[PrevPCReg] = machine->registers[PCReg];      // for debugging,
se we
                                                // are jumping into lala-land
        machine->registers[PCReg] = machine->registers[NextPCReg];
        machine->registers[NextPCReg] = pcAfter;

}
```

Hình 4: Hàm tăng program counter IncreasePC()

Hàm tăng program counter sau khi được cài đặt và được đặt vào đúng vị trí :

```
case SC_Create:
        int virtAddr;
        char* filename;
        DEBUG('a',"\n SC_Create call ...");
        DEBUG('a',"\n Reading virtual address of filename");
        virtAddr = machine->ReadRegister(4);
        DEBUG ('a',"\n Reading filename.");
        filename = User2System(virtAddr,MaxFileLength+1);
        if (filename == NULL)
        {
                printf("\n Not enough memory in system");
                DEBUG('a',"\n Not enough memory in system");
                machine->WriteRegister(2,-1);
                delete filename;
                IncreasePC();
                return;
        }
        DEBUG('a',"\n Finish reading filename.");
        if (!fileSystem->Create(filename,0))
        {
                printf("\n Error create file '%s'",filename);
                machine->WriteRegister(2,-1);
                delete filename;
                IncreasePC();
                return;
        }
        machine->WriteRegister(2,0);
        delete filename;
        IncreasePC();
        break;
        //return;

case SC_ReadInt:
        char* buffer;
```

Hình 5: Một ví dụ về việc sử dụng hàm IncreasePC()

6 Cài đặt system call ReadInt()

Yêu cầu chi tiết : Cài đặt system call int **ReadInt()**. **ReadInt** system call 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 trả về zero (0)

Sau khi khai báo và cấp phát, ta gọi hàm **Read** trong class **synchConsole** để đọc buffer và trả ra độ dài của chuỗi số vừa đọc. Sau đó là các hàm xử lí chuỗi để check xem chuỗi vừa được đọc có hợp lệ không, số vừa đọc ra là số âm hay số dương, là số nguyên hay số thập phân. Sau khi chạy xong, buffer xin cấp phát được giải phóng, tăng program counter trước khi break system call.

```
//negative or positive number
bool isNegative;
isNegative = false;//set number is negative
int indexFirst;
indexFirst=0;
int indexLast;
indexLast=0;
if(buffer[0] == '-')
{
        isNegative = true;
        indexFirst=1;

}
```

Hình 6: Đoạn code check âm dương

```
//check buffer must be integer
for(int i= indexFirst; i<numbytes; i++)
{
        //update indexLast
        indexLast = i;
        if(buffer[i] == '.')//9.0000
        {
                for(int j = i + 1; j < numbytes; j++)
                {
                        // invalid number
                        if(buffer[j] != '0')//9.00100
                                {
                                        printf("\n\n The integer number is not valid");
                                        DEBUG('a', "\n The integer number is not valid");
                                        machine->WriteRegister(2, 0);
                                        IncreasePC();
                                        delete buffer;
                                        return;
                                }
                }
                // update indexLast
                indexLast = i - 1;
                delete buffer;
                return;

        }
        else if(buffer[i] < '0' && buffer[i] > '9')//90238320jslks
        {
                printf("\n\n The integer number is not valid");
                DEBUG('a', "\n The integer number is not valid");
                machine->WriteRegister(2, 0);
                IncreasePC();
                delete buffer;
                return;

        }

}
//convert string buffer to integer
int n;
n=0;//result
for(int i = indexFirst; i<= indexLast; i++)
```

Hình 7: Một phần đoạn code check số có hợp lệ hay không

7 Cài đặt system call PrintInt()

Yêu cầu chi tiết : Cài đặt system call void **PrintInt(int number)**. **PrintInt** system call sẽ sử dụng lớp SynchConsole để xuất một số nguyên ra màn hình.

Thông tin từ việc đọc thông tin từ thanh ghi số 4 sẽ được truyền vào biến **number**. Do trong class **synchConsole** đã có sẵn hàm Write nên phần còn lại của system call này chỉ là xử lí chuỗi vừa đọc được để xuất ra màn hình.

Về xuất ra số âm, chúng ta sẽ đếm số phần tử của số. Sau đó tạo 1 mảng kiểu char để lưu các số của số đó với phần tử đầu tiên là dấu trừ (-). Cuối cùng là dùng hàm **Write** trong class **synchConsole** để xuất ra màn hình 1 chuỗi với độ dài tăng 1 so với số ban đầu (do có thêm dấu trừ).

*số dương chúng ta cũng thực hiện tương tự.

```
case SC_PrintInt:
        int number;
        number = machine->ReadRegister(4);
        int count;
        count=0;
        if(number == 0)
        {
                synchConsole->Write("0", 1);
                IncreasePC();
                return;
        }
        if(number < 0)// number is negative
        {
                number = number * -1;
                int temp;
                temp = number;
                while(temp)// count how many numbers
                {
                        count++;
                        temp /= 10;
                }
                char* buffer;
                int MAX_BUFFER;
                MAX_BUFFER= 255;
                buffer = new char[MAX_BUFFER + 1];
                for(int i = count; i >= 1; i--)//press type int to char
                {
                        buffer[i] = (char)((number % 10) + 48);
                        number /= 10;
                }
                buffer[0] = '-';
                buffer[count + 1] = 0;
                synchConsole->Write(buffer, count + 1);
                delete buffer;
                IncreasePC();
                //return;
        }
        else// number is positive
        {
                int temp;
                temp = number;
```

Xử lí số âm

Hình 8: Đoạn đầu của code system call bao gồm xử lí số âm

8 Cài đặt system call ReadChar()

Yêu cầu chi tiết : Cài đặt system call char **ReadChar()**. **ReadChar** system call sẽ sử dụng lớp **synchConsole** để đọc một ký tự do người dùng nhập vào. Nếu người dùng nhập quá một kí tự thì in ra "Chi duoc nhap duy nhat 1 ky tu!".

Sau khi khai báo và cấp phát, ta gọi hàm **Read** trong class **synchConsole** để đọc buffer và trả ra độ dài của chuỗi ký tự vừa đọc. Sau đó các hàm xử lí chuỗi sẽ kiểm tra xem chuỗi kí tự có hợp lệ hay không, quá 1 kí tự, 1 kí tự hoặc là kí tự rỗng. Sau khi kiểm tra xong thì giải phóng buffer đã được cấp phát và tăng program counter và cuối cùng là break system call.

```
if(byte > 1)
{
    printf("Chi duoc nhap duy nhat 1 ky tu!");
    DEBUG('a', "\nERROR: Chi duoc nhap duy nhat 1 ky tu!");
    machine->WriteRegister(2, 0);
}
else if(byte == 0)
{
    printf("Ky tu rong!");
    DEBUG('a', "\nERROR: Ky tu rong!");
    machine->WriteRegister(2, 0);
}
else
{
    char c = buffer1[0];
    machine->WriteRegister(2, c);
}
```

Hình 9: Đoạn code kiểm tra chuỗi kí tự hợp lệ

9 Cài đặt system call PrintChar

Yêu cầu chi tiết : Cài đặt system call void **PrintChar(char character)**. **PrintChar** system call sẽ sử dụng lớp **synchConsole** để xuất một ký tự ra màn hình

Thông tin từ việc đọc thông tin từ thanh ghi số 4 sẽ được truyền vào biến **character**. Do trong class **synchConsole** đã có sẵn hàm **Write** nên phần còn lại của system call này chỉ là xử lí chuỗi vừa đọc được để xuất ra màn hình và tăng program counter.

```
case SC_PrintChar:
    char character;
    character = (char)machine->ReadRegister(4); //read register r4
    synchConsole->Write(&character, 1); // print character 1 byte
    IncreasePC();
    break;
    //return;
```

Hình 10: Đoạn code của system call PrintChar

10 Cài đặt system call ReadString

Yêu cầu chi tiết :Cài đặt đặt system call void **ReadString (char[] buffer, int length)**, **ReadString** system call sử dụng lớp **synchConsole** dùng để đọc một chuỗi ký tự vào trong buffer (chuỗi sẽ kết thúc khi người dùng nhấn enter, hoặc có chiều dài lớn hơn hoặc bằng giá trị length (>= length)).

Khai báo 2 biến **virtAddress2** và **length2** dùng để lưu lần lượt địa chỉ và độ dài của chuỗi. Lấy 2 thông tin đó bằng cách đọc thanh ghi số 4 và số 5. Tiếp theo sao chép nội dung chuỗi từ User Memory Space sang System Memory Space bằng cách gán biến **buffer2** vào biến hàm **User2System(virtAddress2,length2)**.

Sử dụng phương thức **synchConsole->Read(buffer2, length2)** để đọc chuỗi nhập vào.

Gọi hàm **System2User(virtAddress2, length2, buffer2)** để trả giá trị về cho User.

```
case SC_ReadString:
    int virtAddress2;
    int length2;
    char* buffer2;
    virtAddress2= machine->ReadRegister(4);//address from register r4
    length2 = machine->ReadRegister(5); // length from register r5
    buffer2 = User2System(virtAddress2, length2); // coppy User Space to System Space
    synchConsole->Read(buffer2, length2); //  use point SynchConsole->read for read string
    System2User(virtAddress2, length2, buffer2); // Copy string from System Space to User Space
    delete buffer2;
    IncreasePC();
    break;
    //return;
```

Hình 11: Đoạn code của system call ReadString

11 Cài đặt system call PrintString

Yêu cầu chi tiết :Cài đặt đặt system call **void PrintString (char[] buffer)**, PrintString system call dùng để in chuỗi ký tự trong buffer ra màn hình .

Khai báo biến **virtAddress3** để lưu địa chỉ từ thanh ghi số 4. Và khai báo biến buffer3 để copy từ User sang System với độ dài 255 kí tự.

Tiếp tục khai báo biến **length3** kiểu int để đếm độ dài chuỗi bằng hàm while. Sau đó gọi phương thức **synchConsole->Write(buffer3, length3 + 1)** để in chuỗi ra màn hình.

```
case SC_PrintString:
    int virtAddress3;
    char* buffer3;
    virtAddress3 = machine->ReadRegister(4); //address from register r4
    buffer3 = User2System(virtAddress3, 255); // coppy User Space to System Space with length 255 character
    int length3;
    length3 = 0;
    while (buffer3[length3] != 0)
        length3++; // count length string
    synchConsole->Write(buffer3, length3 + 1); // use point SynchConsole->write for print string
    delete buffer3;
    IncreasePC();
    break;
    //return;
```

Hình 12: Đoạn code của system call PrintString

12 Viết chương trình help

Yêu cầu chi tiết : Viết chương trình help, CT help dùng để in ra các dòng giới thiệu cơ bản về nhóm và mô tả vắn tắt về chương trình sort và ascii. (thật ra: chỉ việc gọi system call **PrintString(char[])**).

Chương trình này nhìn chung khá đơn giản, chỉ việc xuất ra thông tin nhóm và mô tả của 2 chương trình sort và ascii. Về việc xuất thông tin, ta sử dụng system call **PrintString()** đã được viết ở những phần trước.

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

int main() {
        PrintString("\t21127021 - Truong Van Chi\n");
        PrintString("\t21127121 - Phan Van Nguyen\n");
        PrintString("\t21127190 - Le Huu Trong\n");
        PrintString("\t21127290 - Nguyen Gia Hung\n");
        PrintString("\t21127373 - Le Thanh Khoi Nguyen\n");
    PrintString("1. Ascii\n");
    PrintString("\tPrint a table ascii. A special character can't view true.\n");
    PrintString("\tCommand: ./userprog/nachos -rs 1023 -x ./test/ascii\n");
    PrintString("2. Bubble sort\n");
    PrintString("\tInput number elements of array\n");
    PrintString("\tInput all elements\n");
    PrintString("\tPrint a array after sort by bubble sort\n");
    PrintString("\tCommand: ./userprog/nachos -rs 1023 -x ./test/bubblesort\n");

    Halt();
}
```

thông tin nhóm

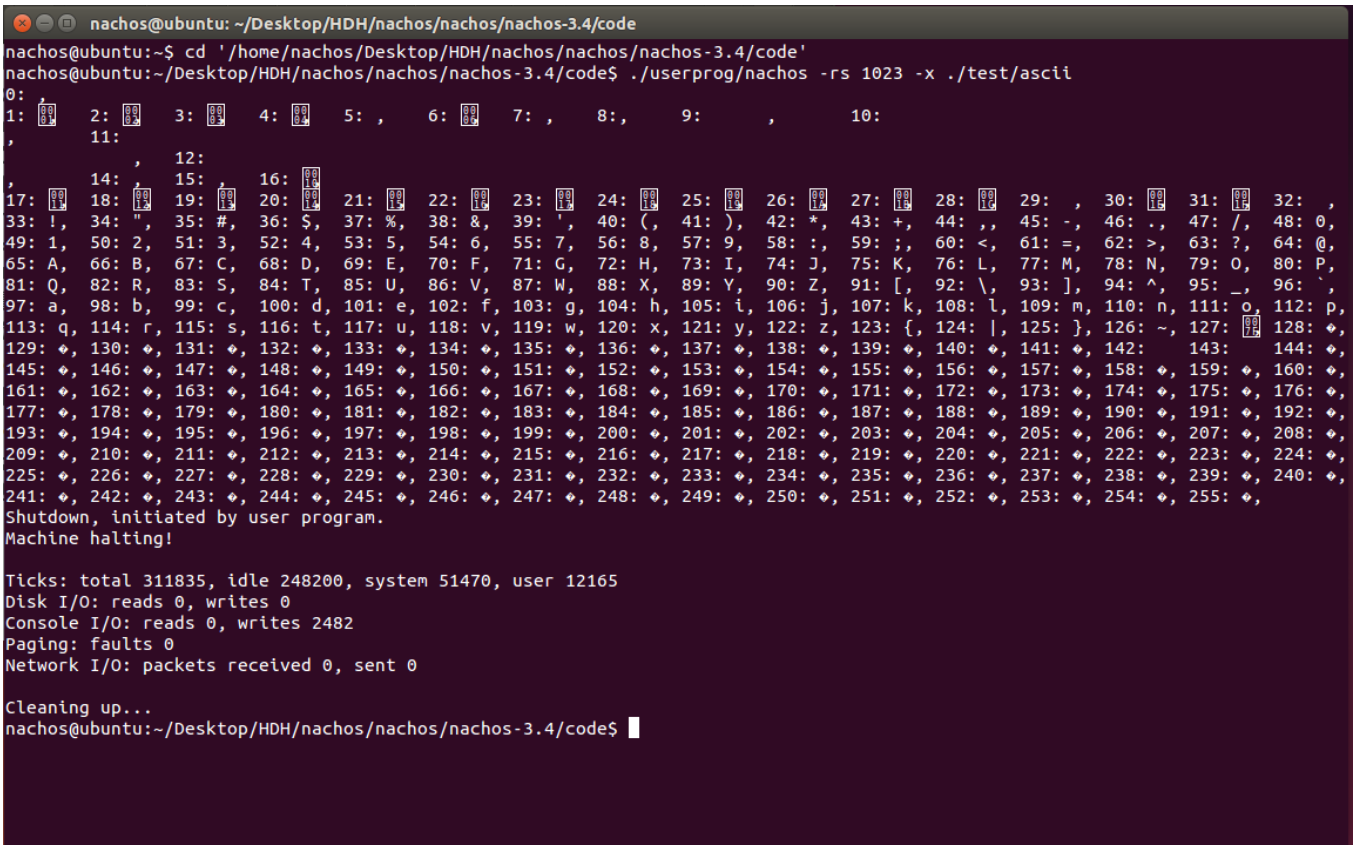
giới thiệu sơ lược các chương trình sort và ascii

Hình 13: Source code chương trình help

13 Viết chương trình ascii

Yêu cầu chi tiết : Viết chương trình ascii để in ra bảng mã ascii.

Bảng mã ascii bao gồm 256 kí tự vậy nên ta chỉ cần gọi vòng lặp for và các system call để tạo chương trình.



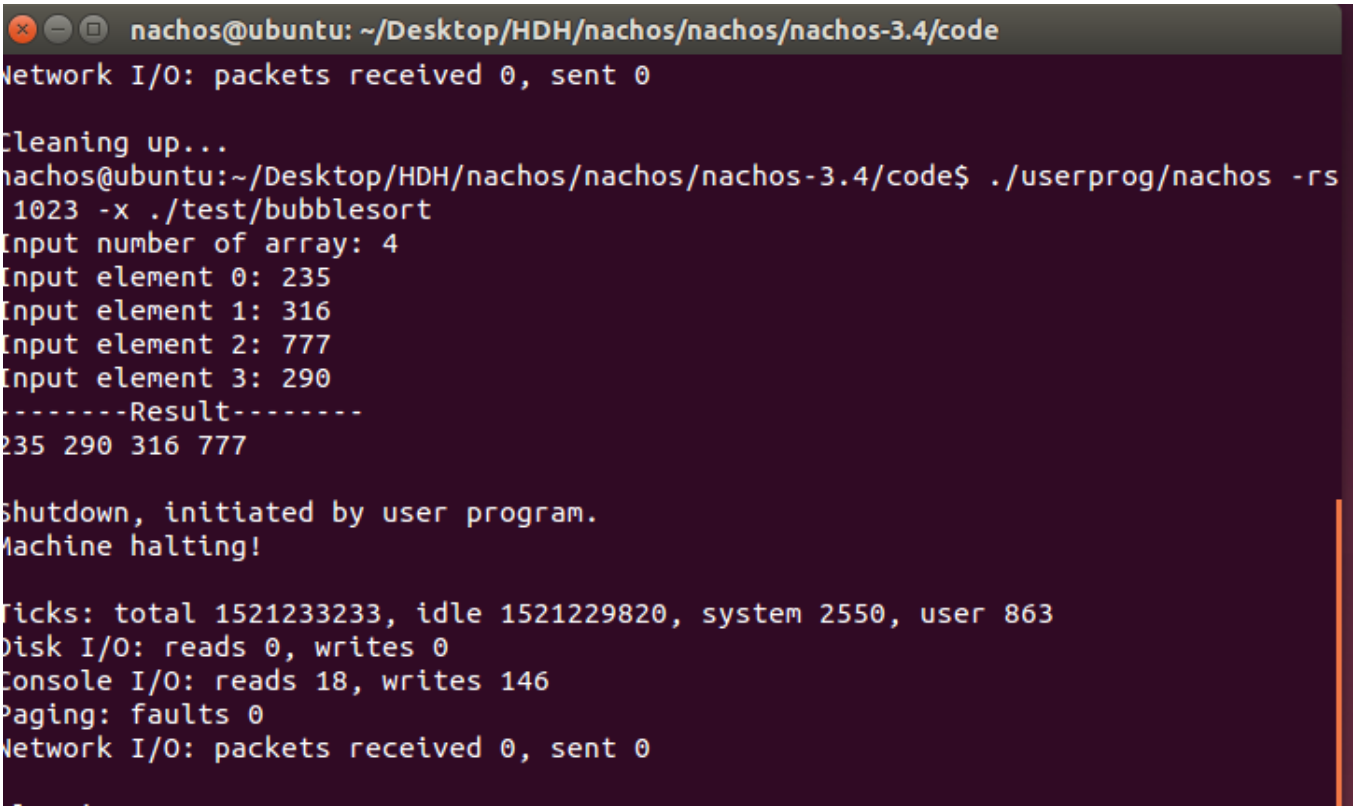
Hình 14: Source code chương trình ascii

14 Viết chương trình sort

Yêu cầu chi tiết : Viết chương trình bublesort với yêu cầu sau

- Cho phép người dùng nhập vào một mảng n số nguyên với n là số do người dùng nhập vào (n =100)

- Sử dụng thuật toán bubble sort để sắp xếp mảng trên.



Hình 15: Source code chương trình bubblesort

Tài liệu tham khảo

[1] *Folder "Các fiile hướng dẫn project 2".*
URL : *https://courses.ctda.hcmus.edu.vn/mod/folder/view.php?id=66333*