



fit@hcmus

TRƯỜNG ĐẠI HỌC KHOA HỌC TỰ NHIÊN,
ĐHQG-HCM
KHOA CÔNG NGHỆ THÔNG TIN
BỘ MÔN HỆ THỐNG THÔNG TIN

BÁO CÁO ĐỒ ÁN HỆ ĐIỀU HÀNH

ĐỒ ÁN 2: NACHOS SYSTEM CALLS

Lớp: 22CLC06 – Nhóm: THPT
Giáo viên hướng dẫn: Thầy Lê Viết Long

MỤC LỤC

I.	GIỚI THIỆU ĐỒ ÁN.....	4
1.	HỆ ĐIỀU HÀNH LÀ GÌ?.....	4
2.	SYSTEM CALL LÀ GÌ? NACHOS LÀ GÌ?	4
3.	GIỚI THIỆU VỀ ĐỒ ÁN	4
II.	TÌM HIỂU NACHOS	5
A.	TÌM HIỂU MỘT SỐ THƯ MỤC LIÊN QUAN:.....	5
1.	PROGTEST.CC.....	5
a.	<i>Thông tin</i>	5
b.	<i>Các hàm và biến</i>	5
2.	SYSCALL.H.....	5
a.	<i>Thông tin</i>	5
b.	<i>Mã thủ tục</i>	5
c.	<i>Các thủ tục của kernel</i>	6
d.	<i>Các hàm của chương trình người dùng</i>	6
3.	EXCEPTION.CC.....	7
a.	<i>Thông tin</i>	7
b.	<i>Các thủ tục</i>	7
4.	BITMAP.*	7
a.	<i>Thông tin</i>	7
b.	<i>bitmap.h</i>	7
c.	<i>bitmap.cc</i>	8
5.	OPENFILE.H	8
a.	<i>Thông tin</i>	8
b.	<i>Lớp OpenFile</i>	8
6.	TRANSLATE.*	9
a.	<i>Thông tin</i>	9
b.	<i>translate.h</i>	9
c.	<i>translate.cc</i>	9
7.	MACHINE.*	10
a.	<i>Thông tin</i>	10
b.	<i>machine.h</i>	10
c.	<i>machine.cc</i>	10
8.	MIPSSIM.CC.....	10
a.	<i>Thông tin</i>	10
b.	<i>Các thủ tục</i>	11
9.	CONSOLE.*	11
a.	<i>Thông tin</i>	11
b.	<i>console.h</i>	11
c.	<i>console.cc</i>	11
10.	SYNCHCONS.*	12
a.	<i>Thông tin</i>	12
b.	<i>synchcons.cc</i>	12
B.	TÌM HIỂU THƯ MỤC TEST:	12

1.	SORT.C	13
a.	Mô tả:.....	13
b.	Nguyên lý:	13
2.	MATMULT.C	13
a.	Mô tả:.....	13
b.	Nguyên lý:	13
3.	START.C VÀ START.S	13
a.	Mô tả:.....	13
b.	Mã nguồn:.....	13
4.	SHELL.C	14
a.	Mô tả:.....	14
b.	Mã nguồn:.....	14
5.	HALT.C	14
a.	Mô tả:.....	14
6.	CREATEFILE.C.....	15
a.	Mô tả:.....	15
b.	Nguyên lý và mục đích:	15
C.	TÌM HIỂU CÁCH BIẾN DỊCH NACHOS.....	15
1.	BƯỚC 1: DI CHUYỂN ĐẾN THƯ MỤC CODE	15
2.	BƯỚC 2: GỌI LỆNH MAKE ĐỂ BẮT ĐẦU BIẾN DỊCH	15
a.	Cách các lệnh trong Makefile (code/Makefile) thực hiện:	16
D.	CÁCH ÁP DỤNG SYSCALL VÀO CHƯƠNG TRÌNH	19
1.	SYSTEM CALL INT READINT().....	19
2.	SYSTEM CALL VOID PRINTINT(INT)	19
3.	SYSTEM CALL FLOAT* READFLOAT().....	19
4.	SYSTEM CALL VOID PRINTFLOAT(FLOAT*)	20
5.	SYSTEM CALL CHAR READCHAR().....	21
6.	SYSTEM CALL VOID PRINTCHAR(CHAR)	21
7.	SYSTEM CALL VOID READSTRING(CHAR*, INT).....	21
8.	SYSTEM CALL VOID PRINTSTRING(CHAR*)	21
9.	SYSTEM CALL INT CREATE(CHAR*)	22
10.	SYSTEM CALL OPENFILEID OPEN(CHAR*, INT).....	22
11.	SYSTEM CALL INT CLOSE(OPENFILEID)	22
12.	SYSTEM CALL INT READ(CHAR*, INT, OPENFILEID).....	22
13.	SYSTEM CALL INT WRITE(CHAR*, INT, OPENFILEID)	23
14.	SYSTEM CALL INT WRITEF2FILE(FLOAT*, OPENFILEID).....	23
15.	SYSTEM CALL INT COMPAREFLOAT(FLOAT*, FLOAT*)	23
16.	SYSTEM CALL VOID CLEARFLOAT(FLOAT*)	24
III.	ĐÁNH GIÁ THÀNH VIÊN.....	24
IV.	ĐƯỜNG DẪN TỚI VIDEO DEMO VÀ GITHUB REPOSITORY	24
V.	THAM KHẢO	24

I. Giới thiệu đề án

1. Hệ điều hành là gì?

Hệ điều hành là một phần mềm dùng để điều hành và quản lý các tài nguyên và các ứng dụng, đóng vai trò trung gian trong việc giao tiếp giữa con người với các thiết bị điện tử. Các hệ điều hành cung cấp cho người dùng giao diện, gọi là GUI (Graphical User Interface) để người dùng có thể dễ dàng trao đổi và sử dụng các dịch vụ từ các thiết bị điện tử của mình.

Có nhiều loại hệ điều hành khác nhau tương ứng với các thiết bị điện tử khác nhau. Ở thiết bị di động, các hệ điều hành phổ biến nhất là iOS dành cho dòng sản phẩm iPhone, iPad của Apple, hoặc hệ điều hành Android cho các dòng sản phẩm Samsung hay Xiaomi,... Tương tự, trên máy tính cá nhân hay máy chủ cũng không thể thiếu hệ điều hành. Một số hệ điều hành phổ biến trên các loại thiết bị này bao gồm Windows, MacOS và Linux.

2. System call là gì? Nachos là gì?

Lệnh hệ thống, hay system call (thường được gọi tắt là syscall), là cách một chương trình yêu cầu hệ điều hành thực hiện. Lệnh hệ thống bao gồm các dịch vụ về phần cứng (như truy cập ổ cứng, truy cập camera,...), tạo lập và thực thi các tiến trình mới, và kết nối với hạt nhân tích hợp (như lịch trình của các tiến trình).

Nachos (viết tắt cho “Not Another Completely Heuristic Operating System”) là một hệ điều hành giả lập được trường đại học California - Berkeley xây dựng nhằm giúp sinh viên hiểu được cách thiết kế và triển khai của một hệ điều hành. NachOS được áp dụng vào rất nhiều giáo trình hệ điều hành ở khắp nơi trên thế giới.

3. Giới thiệu về đề án

Bài báo cáo này thể hiện thành quả của nhóm khi tiến hành nghiên cứu về hệ điều hành, cũng như một vài câu lệnh hệ thống, cách mà chúng vận hành, hoạt động và một số câu lệnh phổ biến.

Mục đích của đề án này là hiểu được cách hệ điều hành hoạt động. Nghiên cứu về cách hệ điều hành khởi tạo, cách hệ điều hành cấp phát tài nguyên, môi trường cho các chương trình, các câu lệnh hệ thống,... giúp cho nhóm đạt được sự hiểu biết nhiều hơn về hệ điều hành, đồng thời trau dồi thêm kiến thức cho bản thân.

II. Tìm hiểu Nachos

A. Tìm hiểu một số thư mục liên quan:

Để thực hiện được các thay đổi trên Nachos cho phù hợp với yêu cầu đề án, việc hiểu được mã nguồn cũng như cách hoạt động là một điều bắt buộc. Dưới đây là kết quả của quá trình reverse – engineering mã nguồn của Nachos.

1. progtest.cc

a. Thông tin

Chứa các thủ tục để kiểm tra xem Nachos có thể chạy được user program hay không. Ngoài ra, các thủ tục này còn để kiểm tra các thiết bị phần cứng (bao gồm thiết bị nhập xuất).

b. Các hàm và biến

- `void StartProcess(char* filename)`: tải user program lên bộ nhớ chính và nhảy đến đó.
- `static Console *console, static Semaphore *readAvail, static Semaphore *writeDone`: các biến dùng để kiểm tra console.
- `ConsoleInterruptHandlers`: gọi các tiến trình yêu cầu nhập xuất.
- `void ConsoleTest(char* in, char* out)`: kiểm tra console bằng cách lấy các ký tự từ quá trình nhập qua quá trình xuất; sẽ dừng lại khi người dùng nhập ký tự 'q'.

2. syscall.h

a. Thông tin

Chứa các thủ tục ở kernel, có thể được người dùng gọi thông qua lệnh "syscall". Tập tin này phải được include ở cả user program và chương trình hệ thống (kernel).

b. Mã thủ tục

Dưới đây là các mã thủ tục tương ứng để kernel nhận diện được. Nếu người lập trình muốn thêm một thủ tục thì bổ sung ở trong file này.

```
#define SC_Halt          0      #define SC_PrintInt      12
#define SC_Exit          1      #define SC_ReadFloat     13
#define SC_Exec          2      #define SC_PrintFloat    14
#define SC_Join          3      #define SC_ReadChar      15
#define SC_Create        4      #define SC_PrintChar     16
#define SC_Open          5      #define SC_ReadString    17
#define SC_Read          6      #define SC_PrintString   18
#define SC_Write         7
#define SC_Close         8
#define SC_Fork          9
#define SC_Yield        10
```

c. Các thủ tục của kernel

Các thủ tục dưới đây có thể được người dùng gọi trực tiếp.

- void Halt(): dừng Nachos
- void Exit(int status): user program đã hoàn thành (status = 0 nghĩa là chương trình thoát bình thường)
- SpaceId Exec(char* name): chạy chương trình thực thi được lưu trong Nachos và trả về địa chỉ vùng không gian mà chương trình thực thi chiếm giữ.
- int Join(SpaceID id): trả ra exit status sau khi chương trình thực thi có mã nhận diện là id đã hoàn thành.

d. Các hàm của chương trình người dùng

Các thủ tục tương tác file bao gồm: tạo, mở, đọc, viết và đóng.

Khi một chương trình yêu cầu thiết bị nhập xuất, nó sẽ có 2 tập tin: nhập từ bàn phím và xuất ra màn hình (hay nói cách khác là stdin và stdout). Có thể đọc và ghi thẳng trên các tập tin này mà không cần mở thiết bị console. typedef int OpenFileId được định nghĩa là mã nhận diện đặc biệt để phân biệt tập tin Nachos.

Dưới đây là các thủ tục tương tác tập tin và console:

- int Create(char* name): tạo tập tin mới
- OpenFileId Open(char* name): mở tập tin Nachos và trả ra OpenFileId có thể được dùng cho việc nhập xuất
- void Write(char *buffer, int size, OpenFileId id): ghi nội dung từ buffer vào tập tin với kích thước là size
- int Read(char *buffer, int size, OpenFileId id): đọc nội dung từ tập tin vào buffer với kích thước là size

- `void Close(OpenFileId id)`: đóng tập tin với mã nhận diện là id
- `int ReadInt()`: đọc số nguyên từ console
- `void PrintInt(int number)`: in số nguyên ra console
- `float ReadFloat()`: đọc số thập phân từ console
- `void PrintFloat(float number)`: in số thập phân ra console
- `char ReadChar()`: đọc ký tự từ console
- `void PrintChar(char character)`: ghi ký tự ra console
- `void ReadString(char *buffer, int length)`: đọc string từ console
- `void PrintString(char *buffer)`: ghi string ra console

3. exception.cc

a. Thông tin

Các hàm chuyển giao user program về cho kernel của Nachos khi user program gọi syscall hoặc có lỗi khiến cho CPU không thể thực hiện tác vụ.

b. Các thủ tục

- `char* User2System(int virtAddr, int limit)`: sao chép string từ vùng nhớ của người dùng qua vùng nhớ của hệ thống
- `int System2User(int virtAddr, int len, char* buffer)`: sao chép string từ vùng nhớ của hệ thống qua vùng nhớ của người dùng
- `void ExceptionHandler(ExceptionType which)`: xử lý các exception, bao gồm syscall hoặc exception do lỗi từ user program

4. bitmap.*

a. Thông tin

Chứa các thủ tục để quản lý bitmap - có nhiệm vụ lưu vết các ô nhớ vật lý. Các bitmap được lưu dưới dạng một số nguyên không dấu, và có thể thực hiện các thao tác trên bit để lấy được thông tin cần thiết.

b. bitmap.h

Lớp BitMap được định nghĩa là một mảng các bit. Thường được sử dụng để quản lý việc cấp phát cho các thành phần, chẳng hạn như các sector hoặc các trang trên bộ nhớ chính. Giá trị của từng bit cho biết sector hoặc trang tương ứng có đang free (tương ứng giá trị 0) hay không (tương ứng giá trị 1).

c. bitmap.cc

Chứa các phương thức quản lý bitmap:

- `BitMap::BitMap(int nitems)`: khởi tạo một bitmap với số bit là `nitems`
- `void BitMap::Mark(int which)`: cài giá trị 1 cho bit ở vị trí `which`
- `void BitMap::Clear(int which)`: cài giá trị 0 cho bit ở vị trí `which`
- `bool BitMap::Test(int which)`: kiểm tra giá trị ở bit `which`, trả về `true` nếu giá trị là 1
- `int BitMap::Find()`: tìm bit đầu tiên có giá trị 0, sau đó gán chúng bằng 1
- `int BitMap::NumClear()`: đếm số bit có giá trị 0
- `void BitMap::Print()`: in toàn bộ bitmap ra, thường dùng cho debug
- `void BitMap::FetchFrom(OpenFile *file)`: đọc bitmap từ một Nachos file
- `void BitMap::WriteBack(OpenFile *file)`: ghi bitmap lên một Nachos file

5. openfile.h

a. Thông tin

Thực hiện các thao tác mở, đóng, đọc và ghi lên từng file. Chúng tương tự với các thao tác trên UNIX.

Có hai cách cài đặt: một loại là "STUB", có chức năng thay đổi các thao tác trên file thành các thao tác trên UNIX; loại thứ hai là các thao tác đọc ghi trên các sector. Cả hai cách cài đặt đều gần tương tự nhau trong lớp `OpenFile`.

b. Lớp `OpenFile`

Các phương thức của lớp:

- `OpenFile(int sector)`: constructor mở file ở sector
- `OpenFile(int sector)`: constructor mở file ở sector có dạng type
- `void Seek(int position)`: đặt con trỏ file ở vị trí tương ứng
- `int ReadAt(char *into, int numBytes, int position)`: đọc một phần của file bắt đầu từ vị trí `position` và lưu vào vị trí `into` trỏ đến, trả về số lượng byte đã đọc
- `int Read(char *into, int numBytes)`: giống như phương thức `ReadAt` nhưng kèm theo việc tăng vị trí con trỏ đọc trên file
- `int Write(char *from, int numBytes)`: ghi một phần của file bắt đầu từ vị trí `position` với nội dung được lấy từ vị trí mà `from` trỏ đến, trả về số lượng byte đã ghi

- `int WriteAt(char *from, int numBytes, int position)`: giống như phương thức `WriteAt` nhưng kèm theo việc tăng vị trí con trỏ ghi trên file
- `int Length()`: trả về kích thước của file (đơn vị: byte)

6. translate.*

a. Thông tin

Hệ thống quản lý việc ánh xạ từ trang ảo (virtual page) qua trang vật lý (physical page). Lớp `TranslationEntry` đóng vai trò là một entry trên cả bảng trang và bộ đệm tìm trang (TLB).

b. translate.h

Chứa định nghĩa của lớp `TranslationEntry`, bao gồm các thuộc tính:

- `int virtualPage`: trang ảo
- `int physicalPage`: trang vật lý
- `bool valid`: mang giá trị true khi tồn tại trên bảng trang
- `bool readOnly`: mang giá trị true khi chỉ được phép đọc
- `bool use`: mang giá trị true khi trang này được đọc hoặc ghi (dùng để chọn trang nạn nhân khi thay trang)
- `bool dirty`: mang giá trị true khi trang này có sự thay đổi và phải được cập nhật lại khi đem ra bộ nhớ phụ

c. translate.cc

Chứa các hàm để ánh xạ từ trang ảo ra trang vật lý. Có hai dạng ánh xạ: bảng trang tuyến tính (linear page table) và TLB.

Các hàm trong `translate.cc`:

- `unsigned int WordToHost(unsigned int word)` và `unsigned short ShortToHost(unsigned short shortword)`: dịch từ `word` và `shortword` sang dạng little endian (và ngược lại).
- `bool Machine::ReadMem(int addr, int size, int *value)`: đọc một số lượng `size` các byte từ bộ nhớ ảo tại địa chỉ `addr` và lưu vào `value`. Trả về false nếu dịch từ địa chỉ ảo qua địa chỉ vật lý không thành công.
- `bool Machine::WriteMem(int addr, int size, int value)`: ghi một số lượng `size` các byte nằm trong `value` lên bộ nhớ ảo tại địa chỉ `addr`. Trả về false nếu dịch từ địa chỉ ảo qua địa chỉ vật lý không thành công.

- `ExceptionType Machine::Translate(int virtAddr, int* physAddr, int size, bool writing)`: dịch từ địa chỉ ảo qua địa chỉ vật lý thông qua bảng trang hoặc TLB. Sau khi kiểm tra và nhận thấy không có lỗi thì gán true cho bit use hoặc dirty và lưu địa chỉ vật lý trong `physAddr`. Nếu có lỗi thì trả ra exception tương ứng. Nếu `writing = true` thì kiểm tra trạng thái `readOnly` trong entry.

7. machine.*

a. Thông tin

Mô phỏng lại các thành phần của Nachos (RAM, register...). Vì Nachos là hệ điều hành đơn chương (ở thời điểm hiện tại), nên chỉ có một instruction được thực thi tại một thời điểm.

b. machine.h

Bao gồm các lớp:

- `Instruction`: câu lệnh của MIPS
- `Machine`: mô phỏng lại các thành phần của máy tính

c. machine.cc

Bao gồm cài đặt các phương thức của lớp `Machine`:

- `Machine::Machine(bool debug)`: constructor để tạo ra một "máy tính" mới; `debug = true` nếu muốn sử dụng debugger sau mỗi instruction
- `void Machine::RaiseException(ExceptionType which, int badVAddr)`: đưa quyền điều khiển cho kernel khi có system call hoặc có exception;
- `void Machine::Debugger()`: debugger dành riêng cho user program
- `void Machine::DumpState()`: in ra trạng thái CPU khi thực thi user program
- `int Machine::ReadRegister(int num)`: đọc thanh ghi của user program
- `void Machine::WriteRegister(int num, int value)`: ghi lên thanh ghi của user program

8. mipssim.cc

a. Thông tin

Mô phỏng lại CPU theo tập lệnh MIPS R2/3000.

b. Các thủ tục

- `void Machine::run()`: mô phỏng quá trình thực thi của user program trên Nachos; được gọi bởi kernel khi user program khởi động.
- `static int TypeToReg(RegType reg, Instruction *instr)`: lấy giá trị thanh ghi trong một instruction.
- `void Machine::OneInstruction(Instruction *instr)`: thực thi các instruction từ user program.
- `void Machine::DelayedLoad(int nextReg, int nextValue)`: mô phỏng quá trình delay
- `void Instruction::Decode()`: giải mã MIPS instruction
- `static void Mult(int a, int b, bool signedArith, int* hiPtr, int* loPtr)`: thực hiện phép nhân trên R2000.

9. console.*

a. Thông tin

Mô phỏng lại thiết bị nhập xuất qua terminal.

Console là thiết bị không đồng nhất. Khi một ký tự được ghi lên thiết bị, thủ tục trả về ngay lập tức và interrupt handler sẽ được gọi khi I/O hoàn tất. Khi đọc, interrupt handler được gọi khi một ký tự mới xuất hiện.

Người dùng có thể gọi bất kỳ thủ tục nào khi I/O interrupt xuất hiện.

b. console.h

Lớp Console là một thiết bị nhập xuất. Quá trình nhập xuất được mô phỏng qua việc đọc/ghi lên file UNIX.

c. console.cc

- `Console::Console(char *readFile, char *writeFile, VoidFunctionPtr readAvail, VoidFunctionPtr writeDone, int callArg)`: khởi tạo console mới.
 - `readFile`: UNIX file mô phỏng nhập (NULL tức là dùng stdin)
 - `writeFile`: UNIX file mô phỏng xuất (NULL tức là dùng stdout)
 - `readAvail`: interrupt handler được gọi khi có ký tự mới xuất hiện
 - `writeDone`: interrupt handler được gọi khi ký tự đã được ghi để ký tự tiếp theo được ghi.

- `void Console::CheckCharAvail()`: kiểm tra xem ký tự tiếp theo có thể được nhập hay không, bằng cách kiểm tra kích thước của bộ đệm có đủ lớn hay không. Gọi interrupt handler cho việc nhập khi ký tự được đưa vào buffer.
- `void Console::WriteDone()`: thủ tục được gọi khi interrupt handler cho việc xuất được gọi, để báo hiệu cho kernel việc in ký tự đã hoàn thành.
- `char Console::GetChar()`: đọc ký tự từ input buffer.
- `void Console::PutChar(char ch)`: ghi ký tự lên màn hình và lên lịch cho interrupt

10. `synchcons.*`

a. Thông tin

Các hàm truy xuất giữa console và người dùng trong Nachos. Được định nghĩa ở lớp `SynchConsole` trong file `synchcons.h`.

b. `synchcons.cc`

- `SynchConsole::SynchConsole()`: tạo một thiết bị console đồng bộ hóa với nhập xuất tiêu chuẩn (`stdin/stdout`).
- `SynchConsole::SynchConsole(char *in, char *out)`: tương tự như trên, nhưng với tên file nhập và xuất
- `int SynchConsole::Write(char *from, int numBytes)`: ghi một số lượng `numBytes` các byte từ buffer lên thiết bị I/O, trả về số lượng byte đã ghi
- `int SynchConsole::Read(char *into, int numBytes)`: đọc một số lượng `numBytes` các byte từ thiết bị I/O lên buffer, trả về số lượng byte đã đọc

B. Tìm hiểu thư mục test:

Đề thuận tiện cho việc xây dựng mã nguồn cũng như đảm bảo rằng hệ điều hành ảo có thể chạy ổn định khi người dùng sử dụng, thư mục test cung cấp các chương trình hỗ trợ cho việc kiểm tra, chạy thử cũng như đảm bảo độ ổn định của hệ điều hành giả lập Nachos.

Mục đích của thư mục: Gây áp lực lên bộ nhớ ảo bằng cách cho hệ thống thực hiện lệnh xử lý một dãy dữ liệu số lượng lớn.

Bằng cách này ta có thể tìm ra khiếm khuyết của hệ thống bộ nhớ hoặc khiếm khuyết trong xử lý lỗi của hệ thống, như lỗi trang, lỗi cache, sập hệ thống, ...

Đồng thời ta cũng có thể biết được mức độ mạnh yếu của hệ thống ảo thông qua tốc độ xử lý dữ liệu, đọc viết file, ...

1. sort.c

a. Mô tả:

Chương trình kiểm tra nhằm mục đích sắp xếp một dãy số nguyên với mục tiêu gây áp lực lên bộ nhớ (RAM) của môi trường ảo. Nếu thành công, ta có thể đọc một chuỗi số nguyên mất trật tự rồi sắp xếp chúng lại vào file.

b. Nguyên lý:

Khai báo dãy gồm 1024 số tượng trưng cho độ lớn của bộ nhớ vật lý, sắp xếp theo thuật toán bubble sort.

2. matmult.c

a. Mô tả:

Chương trình thử nhằm tính nhân ma trận trên một dãy số nguyên số lượng lớn. Nếu thành công, ta có thể đọc các ma trận từ hệ thống rồi viết kết quả lại vào đó.

b. Nguyên lý:

Khai báo 3 ma trận 2 chiều 20x20 nhằm lưu 2 ma trận nhập và 1 ma trận kết quả. Nhập 2 ma trận, gán ma trận kết quả bằng 0 rồi thực hiện phép nhân ma trận.

3. start.c và start.s

a. Mô tả:

start.s: Hợp ngữ hỗ trợ cho chương trình người dùng có thể chạy trên Nachos.
Gồm các lệnh hệ thống và lệnh start mà người dùng có thể gọi thay vì lấy mã nguồn từ thư viện C.

b. Mã nguồn:

__start : Tiến hành chạy chương trình C bằng cách gọi hàm main.

Phải được gọi đầu tiên để được tải ở vị trí 0. Nachos luôn luôn bắt đầu chương trình bằng cách nhảy vào ô 0.

Một số lệnh hệ thống căn bản: Lệnh hợp ngữ dùng để gọi lệnh hệ thống đến hạt nhân Nachos. Mỗi lệnh hệ thống có một lời đưa câu lệnh vào thanh ghi r2 đồng thời không động tới tham số của lệnh hệ thống (nghĩa là tham số 1 ở thanh ghi r4, tham số 2 ở thanh ghi r5, tham số 3 r6, tham số 4 r7). Giá trị trả về ở thanh ghi r2, tuân thủ biện pháp của C trên MIPS.

Gồm các lệnh:

# Halt : Hàm dừng	# Close : Đóng file
# Exit : Hàm thoát	# ReadInt : Đọc số nguyên
# Exec : Hàm thực thi	# PrintInt : In số nguyên
# Join : Hàm hợp	# ReadFloat : Đọc số thực
# Create : Hàm tạo	# PrintFloat : In số thực
# Open : Mở file	# ReadChar : Đọc ký tự
# Read : Đọc file	# PrintChar : In ký tự
# Write : Viết file	# ReadString : Đọc chuỗi
# WriteF2File : Viết số thực lên file	# PrintString : In chuỗi
# CompareFloat : So sánh số thực	# Fork : Tạo tiến trình con
# ClearFloat : Xoá số thực	# Yield : Thông báo hoàn thành sử dụng tài nguyên
__main : hàm tạm để chạy được trên gcc	

4. shell.c

a. Mô tả:

Gồm các hàm để phiên dịch lệnh từ người dùng để truyền đạt cho hạt nhân Nachos thông qua các lệnh hệ thống được khai báo ở start.cc.

b. Mã nguồn:

Gồm một hàm main lập vô tận, khai báo biến tiến trình mới newProc, khai báo dữ liệu nhập xuất từ console consoleInput, consoleOutput, một chuỗi prompt để ngăn cách giữa các lệnh, một ký tự ch và một bộ đệm buffer dài 60 ký tự.

Tiến hành đọc ghi lệnh vào bộ đệm, sau khi kết thúc thêm vào sau ký tự cuối cùng ký tự kết thúc \0 rồi tiến hành thực thi lệnh.

5. halt.c

a. Mô tả:

Một chương trình đơn giản để kiểm tra xem chương trình người dùng có chạy được bình thường không.

Gồm đơn giản một hàm tắt hệ điều hành.

b. Lưu ý:

Các hàm người dùng mà có dữ liệu toàn cục thỉnh thoảng không chạy được trên môi trường Nachos. Một cách khắc phục là di dời các cấu trúc dữ liệu vào trong một tiến trình. Tuy nhiên điều này cần một lượng dung lượng lớn nên cần cẩn thận trong khoản xử lí.

6. createfile.c

a. Mô tả:

Tạo một file có đường dẫn “../test/test.txt” sau đó dừng lại.

b. Nguyên lý và mục đích:

Sử dụng lệnh hệ thống Create và lệnh in chuỗi PrintString để hiển thị kết quả. Nhằm đảm bảo có thể tạo file ổn định.

C. Tìm hiểu cách biên dịch NachOS

Để biên dịch NachOS, ta cần thực hiện 2 bước sau:

1. Bước 1: Di chuyển đến thư mục code

Nhập lệnh sau vào terminal:

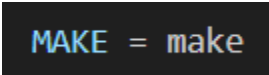
```
% cd [...]nachos/nachos-3.4/code
```

Ví dụ:

```
% cd Desktop/OS/nachos/nachos-3.4/code
```

Với: [...] là đường dẫn tới thư mục nachos được giải nén từ nachos.zip.

2. Bước 2: Gọi lệnh make để bắt đầu biên dịch



Thực hiện lệnh này sau khi đã di chuyển vào thư mục code nơi chứa các file quan trọng cho việc biên dịch:

Makefile: Makefile chạy đầu tiên (Makefile chính để biên dịch toàn bộ nachos).

Makefile.common, Makefile.dep: chứa các định nghĩa cần thiết cho một số Makefile khác.

Nhập lệnh sau vào terminal (sau khi đã sửa dòng lệnh MAKE trong Makefile thành như trên):

```
% make
```

Ví dụ:

Desktop/OS/nachos/nachos-3.4/code\$ make

Sau khi nhập, Makefile trong thư mục code sẽ bắt đầu chạy và nhập lệnh lên terminal.

✚ Cách các lệnh trong Makefile (code/Makefile) thực hiện:

```
all:
    cd threads; $(MAKE) depend
    cd threads; $(MAKE) nachos
```

Đầu tiên, Makefile (code/Makefile) sẽ gọi 2 lệnh:

```
#DEFINES = -DTHREADS
DEFINES = -DUSER_PROGRAM -DFILESYS_NEEDED -DFILESYS_STUB
INCPATH = -I../bin -I../fileysys -I../userprog -I../threads -I../machine
HFILES = $(THREAD_H) $(USERPROG_H)
CFILES = $(THREAD_C) $(USERPROG_C)
C_OFILES = $(THREAD_O) $(USERPROG_O)

include ../Makefile.common
include ../Makefile.dep
```

```
depend: $(CFILES) $(HFILES)
    $(CC) $(INCPATH) $(DEFINES) $(HOST) -DCHANGED -M $(CFILES) > makedep
    echo '/^# DO NOT DELETE THIS LINE/+2,$$d' >eddep
    echo '$$r makedep' >>eddep
    echo 'w' >>eddep
    ed - Makefile < eddep
    rm eddep makedep
    echo '# DEPENDENCIES MUST END AT END OF FILE' >> Makefile
    echo '# IF YOU PUT STUFF HERE IT WILL GO AWAY' >> Makefile
    echo '# see make depend above' >> Makefile
```

<cd threads> để terminal di chuyển vào thư mục <code/threads>.

<make depend> để chạy file Makefile (code/threads/Makefile) nhằm tạo các file object (.o) dùng cho việc tạo file nachos.

Lưu ý 1: <make depend> khi được gọi sẽ dò tìm định nghĩa của <depend> trong Makefile <code/threads/Makefile>, nếu không khai báo (ở đây là không include Makefile.common) thì sẽ bị lỗi.

Lưu ý 2: định nghĩa của <depend> được quy định trong Makefile.common, <make depend> tương đương với việc gọi lệnh {\$(CFILES) [...] >> Makefile}.

Lưu ý 3: Các tham số \$(CFILES),... đã được quy định đầu Makefile trong threads và Makefile.dep.

Sau khi lệnh <make depend> chạy xong và các file .o đã được tạo thành công thì terminal sẽ rời khỏi thư mục <code/threads> về lại thư mục <code> để chạy dòng lệnh tiếp theo.

Tiếp theo, Makefile (code/Makefile) sẽ gọi 2 lệnh:

```
PROGRAM = nachos
```

```
$(PROGRAM): $(OFILES)
    $(LD) $(OFILES) $(LDFLAGS) -o $(PROGRAM)
```

<cd threads> để terminal di chuyển vào thư mục <code/threads>.

<make nachos> để liên kết các file .o lại thành file thực thi nachos (code/threads/nachos).

Lưu ý 1: <make nachos> sẽ hoạt động khá tương tự <make depend>, chỉ khác ở định nghĩa của lệnh <make> đối với <nachos> và <depend>.

Lưu ý 2: định nghĩa của <nachos> đã được quy định trong Makefile.common, <make nachos> tương đương với việc gọi lệnh {\$(OFILES) [...] -o \$(PROGRAM)}.

Sau khi lệnh <make nachos> chạy thành công thì trình biên dịch sẽ rời khỏi <code/threads> về lại thư mục <code> và tiếp tục biên dịch lệnh tiếp theo.

8 dòng lệnh tiếp theo sẽ thực hiện tương tự như đối với <code/threads> (2 dòng lệnh đầu).

```
CC=gcc
# Sparc/Solaris
# CFLAGS= -I./ -I../threads -DHOST_IS_BIG_ENDIAN
# Linux
CFLAGS=-I./ -I../threads -g

LD=gcc

all: coff2noff

# converts a COFF file to Nachos object format
coff2noff: coff2noff.o
    $(LD) coff2noff.o -o coff2noff

# converts a COFF file to a flat address space (for Nachos version 2)
coff2flat: coff2flat.o
    $(LD) coff2flat.o -o coff2flat

# dis-assembles a COFF file
disassemble: out.o opstrings.o
    $(LD) out.o opstrings.o -o disassemble
```

Đối với dòng lệnh thứ 9, Makefile (code/Makefile) sẽ gọi 2 lệnh

<cd bin> để terminal di chuyển vào thư mục <code/bin>.

<make all> để chạy Makefile (code/bin/Makefile) nhằm tạo file thực thi coff2noff dùng để chuyển các file thực thi khi biên dịch thư mục <code/test> thành file .noff để chúng có thể chạy trên môi trường ảo được nachos tạo ra.

Lưu ý 1: <make all> sẽ dò tìm lệnh cần make, lệnh này sẽ khai báo sau <all:> trong Makefile, ở đây chỉ có coff2noff.

Lưu ý 2: <make all> lúc này sẽ dò tìm đến vị trí của coff2noff, tạo file .o theo format giống với file .o dùng để tạo Nachos (coff2flat và disassemble sẽ ko bị tìm đến do đó không chạy) và biên dịch thành file thực thi.

Lưu ý 3: <make all> <make coff2noff> {\$(LD) [...] coff2noff}.

Đối với dòng lệnh thứ 9, Makefile (code/Makefile) sẽ gọi 2 lệnh để biên dịch các file trong thư mục <code/test> tương tự như đối với <code/bin>.

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

Lưu ý 1: Các lệnh cần make sẽ khác một chút so với coff2noff, ở đây chúng cần thông qua file thực thi coff2noff để tạo thành file thực thi dùng để chạy trên NachOS.

Lưu ý 2: Quá trình biên dịch các file trong test sẽ giống như sau:

D. Cách áp dụng syscall vào chương trình

Program Counter sẽ được tăng và cuối switch case của type của SyscallException

1. System call int ReadInt()

Khi được gọi, NachOS sẽ dùng lớp SynchronConsole để nhận đầu vào từ bàn phím.

Đối với syscall này, console sẽ cho nhập tối đa 11 kí tự (INT_MIN = -2147483647) và lưu vào chuỗi buffer.

Sau khi lấy được chuỗi ký tự đầu vào, syscall sẽ kiể

Đối với syscall này, console sẽ cho nhập tối đa 40 số (FLOAT_MIN = -3.4e38) và lưu vào chuỗi buffer.

Sau khi lấy được chuỗi ký tự đầu vào, syscall sẽ dùng dùng hàm atof() để chuyển chuỗi thành số float và lưu vào vùng nhớ heap được trả bởi con trỏ *number.

Syscall sẽ ép kiểu địa chỉ lưu trong con trỏ thành int và ghi vào thanh ghi r2.

Lý do nhóm trả về con trỏ float dù nó thuộc system space là bởi vì:

- Do registers giả lập của NachOS thuộc kiểu int nên không thể trả về 1 số float trực tiếp được, nếu làm vậy sẽ bị lỗi Illegal Instruction
- Nếu ép kiểu số float thành int (giữ nguyên các bit) bằng reinterpret_cast hay bất kỳ cách nào khác và ghi vào thanh ghi r2 trả về thì khi chương trình của user ép kiểu lại thành float sẽ rơi vào lỗi Illegal Instruction bất kể là ép kiểu trực tiếp hay ép kiểu thông qua con trỏ void.
- Nếu áp dụng cách tương tự với syscall ReadString thì khi chương trình của user ép kiểu lại vẫn sẽ rơi vào lỗi Illegal Instruction.
- Nếu chương trình user không ép kiểu mà thay vào đó khởi tạo con trỏ để trỏ đến vùng nhớ chứa dữ liệu (ReadFloat trả về int (ép kiểu từ float) hoặc dùng cách tương tự với ReadString để WriteMem vào chuỗi tạm) thì khi dereference sẽ gặp lỗi Address Error hoặc Illegal Instruction.

Bất lợi:

- Không thể dereference bởi vì sẽ gặp lỗi Illegal Instruction (bởi vì vùng nhớ được trỏ tới thuộc về system space), do đó đây là một lỗ hổng nguy hiểm có thể dễ dàng được sử dụng để làm sụp hệ điều hành.
- Một lỗ hổng khác là nếu user dùng syscall ClearFloat để trả vùng nhớ lại cho hệ điều hành nhưng vẫn tiếp tục gọi các syscall liên quan thì sẽ dẫn đến run-time error. Ở đây có thể là Address Error hoặc Illegal Instruction tùy trường hợp
- Do không thể dereference nên cần tạo thêm các syscall nhận float* (1 số int) để làm các công việc đơn giản như so sánh 2 số float.

4. System call void PrintFloat(float*)

Khi được gọi, syscall sẽ đọc thanh ghi r4 để lấy địa chỉ (đang là 1 số int) và ép kiểu lại thành float*.

Hàm sprintf sẽ được dùng để chuyển số float thành chuỗi ký tự và lưu vào buffer.

Sau khi lấy được chuỗi ký tự đầu ra, NachOS sẽ dùng lớp SynchConsole để in chuỗi ra màn hình.

5. System call char ReadChar()

Khi được gọi, NachOS sẽ dùng lớp SynchConsole để nhận đầu vào từ bàn phím.

Đối với syscall này, console sẽ cho nhập tối đa 256 ký tự, nếu số ký tự nhập vào khác 1 thì sẽ trả về 0 (tương đương ký tự '\0'), ngược lại trả về ký tự đã nhập

6. System call void PrintChar(char)

Khi được gọi, syscall sẽ đọc thanh ghi r4 và ép kiểu thành char

Sau khi lấy được chuỗi ký tự đầu ra, NachOS sẽ dùng lớp SynchConsole để in chuỗi ra màn hình.

7. System call void ReadString(char*, int)

Khi được gọi, syscall sẽ đọc thanh ghi r4 để lấy địa chỉ của buffer đầu vào và thanh ghi r5 để lấy độ dài của buffer đầu vào.

Syscall sẽ tạo một buffer có độ dài là độ dài là buffer + 1 (ký tự kết thúc chuỗi) để lưu chuỗi nhập từ console.

Syscall dùng lớp SynchConsole để nhận chuỗi từ console.

Sau khi nhận chuỗi thì sẽ dùng hàm System2User để ghi vào user space tại địa chỉ của buffer đầu vào.

Ngoài ra, syscall vẫn ghi vào r2 số ký tự đọc được. Do đó nếu cần vẫn có thể trả về int thay cho void.

8. System call void PrintString(char*)

Khi được gọi, syscall sẽ đọc thanh ghi r4 để lấy địa chỉ của chuỗi đầu vào.

Syscall sẽ sử dụng hàm User2System để copy dữ liệu trong chuỗi đầu vào sang buffer.

Syscall sẽ sử dụng lớp SynchConsole để in buffer từng ký tự một đến khi gặp ký tự '\0'

Ngoài ra, syscall vẫn ghi vào r2 số ký tự in được. Do đó nếu cần vẫn có thể trả về int thay cho void.

9. System call `int Create(char*)`

Khi được gọi, syscall sẽ đọc thanh ghi r4 để lấy địa chỉ của chuỗi chứa tên file đầu vào.

Syscall sẽ sử dụng hàm `User2System` để copy dữ liệu trong chuỗi đầu vào sang filename và kiểm tra tính hợp lệ, nếu có thì tiếp tục còn không thì trả về -1.

Sau khi kiểm tra tính hợp lệ của chuỗi filename thì gọi phương thức `Create()` của lớp `FileSystem` để tạo file mới với kích thước 0 byte. Nếu không tạo được file thì trả về -1, ngược lại sẽ trả về 0.

10. System call `OpenFileId Open(char*, int)`

Khi được gọi, syscall sẽ đọc thanh ghi r4 để lấy địa chỉ của chuỗi đầu vào.

Syscall sẽ sử dụng hàm `User2System` để copy dữ liệu trong chuỗi đầu vào sang filename và kiểm tra tính hợp lệ, nếu có thì tiếp tục còn không thì trả về -1.

Sau khi kiểm tra tính hợp lệ của chuỗi thì tìm chỗ trống trong bảng file, nếu không có thì trả về -1, nếu có thì kiểm tra kiểu mở file.

Nếu kiểu mở file là 2 hoặc 3 thì trả về 0 hoặc 1 (tương ứng với 2 file `stdin` hoặc `stdout` trên bảng trang), nếu là 0 hoặc 1 thì dùng hàm `Open` của `FileSystem` để mở file, lưu file vào vị trí trống trên bảng trang và trả về chỉ số của vị trí đó.

11. System call `int Close(OpenFileId)`

Khi được gọi, syscall sẽ đọc thanh ghi r4 để lấy id (chỉ số của file trên bảng trang).

Nếu id nằm ngoài bảng trang (không thuộc 0-9) hoặc thuộc về `stdin` hoặc `stdout` (0 hoặc 1) thì trả về -1.

Nếu bảng trang tại chỉ số id là trống thì trả về -1, nếu không thì đóng file và xóa file tại vị trí tương ứng trên bảng trang.

12. System call `int Read(char*, int, OpenFileId)`

Khi được gọi, syscall sẽ đọc thanh ghi r4 để lấy địa chỉ của chuỗi đầu ra trên user space, thanh ghi r5 để lấy độ dài chuỗi và thanh ghi r6 để lấy id trên bảng trang.

Nếu id nằm ngoài bảng trang (không thuộc 0-9) hoặc thuộc về `stdin` (0) thì trả về -1. Nếu là `stdin` thì xử lý tương tự syscall `ReadString`.

Nếu bảng trang tại chỉ số id là trống thì trả về -1, nếu không thì dùng một con trỏ của lớp OpenFile để lưu file và dùng hàm Read() của OpenFile để lấy dữ liệu từ file lưu vào buffer.

Sau khi có được dữ liệu từ file, kiểm tra kết thúc file hoặc chuỗi. Sau đó dùng hàm System2User để WriteMem vào địa chỉ chuỗi đầu ra trên user space.

Trả về -2 nếu kết thúc file, số ký tự đọc được nếu chưa kết thúc file.

13. System call int Write(char*, int, OpenFileId)

Khi được gọi, syscall sẽ đọc thanh ghi r4 để lấy địa chỉ của chuỗi đầu ra trên user space, thanh ghi r5 để lấy độ dài chuỗi và thanh ghi r6 để lấy id trên bảng trang.

Nếu id nằm ngoài bảng trang (không thuộc 0-9) hoặc thuộc về stdout (1) hoặc là loại file chỉ đọc (type = 1) thì trả về -1. Nếu là stdout thì xử lý tương tự syscall WriteString.

Nếu bảng trang tại chỉ số id là trống thì trả về -1, nếu không thì dùng một con trỏ của lớp OpenFile để lưu file.

Sử dụng hàm User2System để lưu dữ liệu ghi vào file vào buffer sau đó dùng hàm Write() của OpenFile để ghi dữ liệu vào file.

Trả về -2 nếu số ký tự ghi vào được khác số ký tự của chuỗi đầu vào.

14. System call int WriteF2File(float*, OpenFileId)

Khi được gọi, syscall sẽ đọc thanh ghi r4 để lấy địa chỉ của vùng nhớ (là 1 số int) chứa số float và ép kiểu thành float*, đọc thanh ghi r5 để lấy id của file trên bảng trang.

Dùng hàm sprintf để chuyển số float thành chuỗi ký tự lưu trong buffer.

Sau đó ghi dữ liệu vào file tương tự như syscall Write.

15. System call int CompareFloat(float*, float*)

Khi được gọi, syscall sẽ đọc thanh ghi r4 để lấy địa chỉ của vùng nhớ chứa số thứ nhất và thanh ghi r5 để lấy địa chỉ của vùng nhớ chứa số thứ hai sau đó ép kiểu cả 2 thành float*.

So sánh 2 số float, nếu số thứ nhất lớn hơn thì trả về 1, nếu hai số bằng nhau thì trả về 0 còn nếu số thứ nhất bé hơn thì trả về -1.

16. System call void ClearFloat(float*)

Khi được gọi, syscall sẽ đọc thanh ghi r4 để lấy địa chỉ của vùng nhớ (là 1 số int) chứa số float và ép kiểu thành float*.

Nếu con trỏ không chứa giá trị NULL thì giải phóng vùng nhớ

III. Đánh giá thành viên

MSSV	Họ và tên	Tỷ lệ đóng góp
22127121	Đào Việt Hoàng	25%
22127320	Bùi Tá Phát	25%
22127432	Đặng Thanh Tú	25%
22127486	Đào Ngọc Thiện	25%

IV. Đường dẫn tới video demo và GitHub repository

- Video demo: <https://youtu.be/uKtCYqtZlGE?si=yUTKYo-a8CAIObDo>
- GitHub: [LancelotMoretti/NachOS-Development: OS developing NachOS project \(github.com\)](https://github.com/LancelotMoretti/NachOS-Development)

V. Tham khảo

- [http://www.cs.washington.edu/homes/tom/nachos/Teacher's notes](http://www.cs.washington.edu/homes/tom/nachos/Teacher's%20notes)
- Giáo trình do giáo viên cung cấp