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

BÁO CÁO PROJECT 3

MÔN HỌC: HỆ ĐIỀU HÀNH

Sinh viên thực hiện: 20120443 - Nguyễn Tấn Chữ
20120444 - Nguyễn Chí Công
20120447 - Trịnh Quốc Cường
20120383 - Nguyễn Đức Tiến
20120446 - Nguyễn Đình Cường

| Giáo viên hướng dẫn |

Thầy Lê Viết Long

Thành Phố Hồ Chí Minh - 2022

Mục lục

Phân chia công việc **3**

I/ Tìm hiểu và cài đặt tổng quan **4**

1. Tổng quan về NachOS 4
2. Cài đặt và biên dịch NachOS 4
3. Quá trình biên dịch trên NachOS 4
4. Cài đặt tổng quan 5

II/ Cài đặt các System Calls nhập xuất file **6**

1. Cài đặt system call CreateFile 6
2. Cài đặt system call Open và Close 7
3. Cài đặt system call Read và Write 8

III/Đa Chương, lập lịch và đồng bộ hóa trong NachOS **10**

1. Cài đặt đa tiến trình: 10
2. Thiết kế các lớp theo hướng đối tượng, bao gồm 4 lớp sau: 10
3. Cài đặt các system call 14
   a. Cài đặt system call SpaceID Exec(char* name) 14
   b. Cài đặt system calls int Join (SpaceID id) và void Exit(int exitCode). 15
   c. Cài đặt system call int CreateSemaphore(char* name, int semval). 16
   d. Cài đặt system call int Wait(char* name), và int Signal(char* name). 17
4. Viết chương trình ping pong kiểm thử 18
   a. Giải thích code 18
   b. Hướng dẫn sử dụng chương trình 18

IV.Tài Liệu Tham Khảo **21**

Phân chia công việc

MSSV	Họ và tên	Công việc	Mức độ hoàn thành
20120443	Nguyễn Tấn Chữ	- Viết chương trình ping pong, kiểm thử các system call, viết báo cáo	100%
20120444	Nguyễn Chí Công	- Viết system call Join và Exit - Viết system call CreateSemaphore - Viết system call Wait và Signal	100%
20120447	Trịnh Quốc Cường	- Viết chương trình ping pong, kiểm thử các system call, viết báo cáo	100%
20120383	Nguyễn Đức Tiến	- Viết system call CreateFile - Viết system call Open và Close - Viết system call Write và Read	100%
20120446	Nguyễn Đình Cường	- Thiết kế các lớp đa chương, lập lịch và đồng bộ hóa trong Nachos. - Viết system call SpaceID Exec	100%

I/ Tìm hiểu và cài đặt tổng quan

1. Tổng quan về NachOS

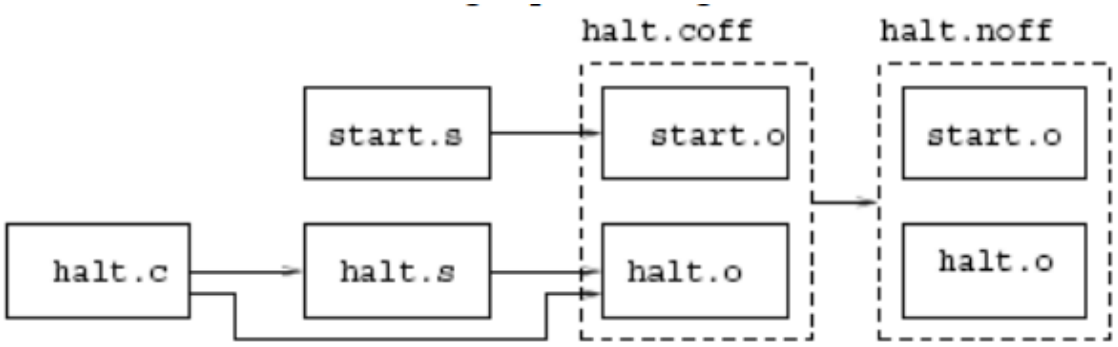
- NachOS (Not Another Completely Heuristic Operating System) là phần mềm mã nguồn mở giả lập một máy tính ảo và một số thành phần cơ bản của hệ điều hành chạy trên máy tính ảo này nhằm giúp cho việc tìm hiểu và xây dựng các thành phần phức tạp của hệ điều hành.

2. Cài đặt và biên dịch NachOS

- Sử dụng VNware Workstation để tạo máy ảo và cài đặt hệ điều hành Ubuntu 14.04 lên trên máy ảo.
- Sau đó tiến hành cài trình biên dịch gcc/g++ trong Ubuntu
- Tiến hành biên dịch NachOS (mã nguồn NachOS đã được cấu hình với cross-compiler-MIPS)

3. Quá trình biên dịch trên NachOS

- Trong thư mục ../nachos-3.4/code/test có file hợp ngữ start.s, khi biên dịch, file này sẽ biên dịch các file mã nguồn <tenfile X>.c thành file hợp ngữ có đuôi <tenfile X>.s
- Sau đó file <tenfile X>.s này sẽ liên kết với file start.s tạo thành 1 file <tenfile X>.coff (bao gồm file <tenfile X>.o và start.o), đây là dạng file thực thi trên Linux vơi kiến trúc MIPS.
- File <tenfile X>.coff sẽ được công cụ coff2noff (được viết sẵn trong NachOS) chuyển thành file <tenfile X>.noff , đây chính là dạng file thực thi trên NachOS với kiến trúc MIPS.
- Quá trình biên dịch này có thể mô tả qua 1 ví dụ với file ‘haft.c’ sau đây:



4. Cài đặt tổng quan

a) Cài đặt trước

- Để sao chép được vùng nhớ từ User sang System và từ System sang User, tiến hành cài đặt lần lượt 2 hàm: Char* User2System(int virtAddr, int limit) và int System2User(int virtAddr, int len, char* buffer) trong file ../code/userprog/exception.cc
- Để tránh việc NachOS sẽ bị vòng lặp gọi thực hiện system call này mãi mãi thì cần tăng program counter trước khi system call trả kết quả về. Tiến hành cài đặt hàm tăng Program counter trong file ../code/userprog/exception.cc

b) Cài đặt system calls:

- Trong file ../code/userprog/syscall.h, tiến hành khai báo các system call sẽ cài đặt.

- Để NachOS có thể call những system call đã khai báo, cần phải thêm 1 số mã lệnh MIPS cho từng system call vào 2 file ../code/test/start.c và ../code/test/start.s. 1 đoạn lệnh ví dụ về system call ReadInt:

```
        .globl readInt
        .ent readInt
readInt:
        addiu $2, $0, SC_readInt
        syscall
        j       $31
        .end readInt

        .globl printInt
        .ent printInt
```

- Các system call còn lại đều có cùng format như đoạn lệnh trên. Tiến hành cài đặt lần lượt vào 2 file ở trên

II/ Cài đặt các System Calls nhập xuất file

1. Cài đặt system call CreateFile

- CreateFile system call sẽ sử dụng Nachos FileSystem Object để tạo một file rỗng. Chuỗi filename đang ở trong user space, có nghĩa là buffer mà con trỏ trong user space trỏ tới phải được chuyển từ vùng nhớ user space tới vùng nhớ system space. System call CreateFile trả về 0 nếu thành công và -1 nếu có lỗi.
- Cài đặt CreateFile:
  + Bước 1: Ta sẽ đọc địa chỉ của tên file được lưu trong thanh ghi r4.
  + Bước 2: Ta thực hiện chép giá trị ở thanh ghi r4 từ vùng nhớ user space tới vùng nhớ system space bằng hàm User2System(). Giá trị được chép là tên file mình sẽ tạo.
  + Bước 3: Sau đó ta sẽ kiểm tra tên file vừa được chép xem tên file có NULL không, file có được tạo ra với cái tên đó không. Nếu như kiểm tra đúng thì sẽ báo lỗi và trả về -1 vào thanh ghi r2. Ngược lại thì thành công và trả về 0 vào thanh ghi r2.

2. Cài đặt system call Open và Close

- User program có thể mở 2 loại file, file chỉ đọc và file đọc và ghi. Mỗi tiến trình sẽ được cấp một bảng mô tả file với kích thước cố định. Đồ án này, kích thước của bảng mô tả file là có thể lưu được đặc tả của 10 files. Trong đó, 2 phần tử đầu, ô 0 và ô 1 để dành cho console input và console output.
- System call mở file phải làm nhiệm vụ chuyển đổi địa chỉ buffer trong user space khi cần thiết và viết hàm xử lý phù hợp trong kernel. System call Open sẽ trả về id của file (OpenFileID = một số nguyên), hoặc là -1 nếu bị lỗi.
- Mở file có thể bị lỗi như trường hợp là không tồn tại tên file hay không đủ ô nhớ trong bảng mô tả file. Tham số type = 0 cho mở file đọc và ghi, = 1 cho file chỉ đọc. Nếu tham số truyền bị sai thì system call phải báo lỗi. System call sẽ trả về -1 nếu bị lỗi và 0 nếu thành công.
- Quy ước tham số type:

  + type = 0: file đọc và ghi

  + type = 1: file chỉ đọc

  + type = 2: stdin

  + type = 3: stdout
- Cài đặt Open:

  + Bước 1: Ta đọc địa chỉ của tên file trong thanh ghi r4, đọc giá trị cho tham số type trong thanh ghi r5.

  + Bước 2: Kiểm tra tham số type có nằm trong đoạn [0, 3] không. Sau đó kiểm tra index của file có nằm trong bảng mô tả file không [0, 9].

  + Bước 3: Nếu thỏa 2 điều kiện trên, thì ta sẽ chép giá trị ở thanh ghi r4 bằng hàm User2System.

  + Bước 4: Ta sẽ kiểm tra file đang mở ở type nào. Nếu type = 0 hoặc type = 1. Thi ta sẽ ghi giá trị = index – 1 vào thành ghi r2. Nếu type = 2 thì ta sẽ ghi giá trị 0 vào thanh ghi r2. Nếu type = 3 thì ta sẽ ghi giá trị 1 vào thanh ghi r2. Thanh ghi r2 sẽ lưu lại id của file.

  + Trường hợp mở file không tồn tại hoặc 2 điều kiện ban đầu không thỏa thì trả về -1 cho thanh ghi r2 để báo lỗi.

- Cài đặt Close:

  + Bước 1: Ta đọc địa chỉ tên file trong thanh ghi r4

  + Bước 2: Sau đó ta kiểm tra file cần đóng có tồn tại không bằng openf[id] được cài đặt trong lớp FileSystem. Ngoài ra, ta sẽ kiểm tra id đó có nằm trong bảng mô tả file không.

  + Bước 3: Nếu 2 điều kiện đều thỏa thì ta sẽ xóa dữ liệu trong openf[id] và gán lại bằng NULL. Gán thanh ghi r2 bằng 0.

  + Bước 4: Nếu có 1 trong 2 không thỏa và báo lỗi thì ta sẽ gán giá trị -1 vào thanh ghi r2 và báo lỗi.

3. Cài đặt system call Read và Write

- Các system call đọc và ghi vào file với id cho trước. Bạn cần phải chuyển vùng nhớ giữa user space và system space, và cần phải phân biệt giữa Console IO (OpenFileID 0, 1) và File.
- Lệnh Read và Write sẽ làm việc như sau: Phần console read và write, bạn sẽ sử dụng lớp SynchConsole. Được khởi tạo qua biến toàn cục gSynchConsole(bạn phải khai báo biến này). Bạn sẽ sử dụng các hàm mặc định của SynchConsole để đọc và ghi, tuy nhiên bạn phải chịu trách nhiệm trả về đúng giá trị cho user. Đọc và ghi với Console sẽ trả về số bytes đọc và ghi thật sự, chứ không phải số bytes được yêu cầu.
- Trong trường hợp đọc hay ghi vào console bị lỗi thì trả về -1. Nếu đang đọc từ console và chạm tới cuối file thì trả về -2. Đọc và ghi vào console sẽ sử dụng dữ liệu ASCII để cho input và output, (ASCII dùng kết thúc chuỗi là NULL (\0)). Phần đọc, ghi vào file, bạn sẽ sử dụng các lớp được cung cấp trong file system. Sử dụng các hàm mặc định có sẵn của filesystem và thông số trả về cũng phải giống như việc trả về trong synchconsole. Cả read và write trả số kí tự đọc, ghi thật sự. Cả Read và Write trả về -1 nếu bị lỗi và -2 nếu cuối file. Cả Read và Write sử dụng dữ liệu binary.
- Cài đặt Read:

  + Bước 1: Ta đọc tham số buffer từ thanh ghi r4, tham số charcount từ thanh ghi r5 và id từ thanh ghi r6.

  + Bước 2: Ta kiểm tra id có nằm trong bảng mô tả file không. File cần đọc có tồn tại không và file đó có type khác 3 ko.

+ Bước 3: Trường hợp vi phạm 1 trong 3 điều kiện: Chương trình báo lỗi và trả về -1 cho thanh ghi r2.

+ Bước 4: Trường hợp hợp lệ thì ta lấy vị trí con trỏ trong file bằng phương thức GetCurrentPos() trong lớp FileSystem gọi là OldPos. Sau đó chép giá trị từ thanh ghi r4 từ phía User sang System bằng User2System. Giá trị chép là buffer (chuỗi ký tự).

+ Bước 5: Ta xét trường hợp type = 2. Nếu thỏa ta sẽ gọi phương thức Read của gSynchConsole để đọc buffer có độ dài charcount. Sau đó trả số byte đọc được cho thanh ghi r2. Tiếp theo chép buffer từ System sang User bằng System2User().

+ Bước 6: Ta xét trường hợp type = 1 và type = 0. Nếu thỏa ta sẽ lấy vị trí con trỏ hiện tại bằng GetCurrentPos() gọi là NewPos, trả về số byte đọc được cho thanh ghi r2 bằng công thức: NewPos -OldPos và cũng chép buffer từ phía System sang User bằng System2User().

+ Bước 7: Nếu file cần đọc rỗng thì ta sẽ trả về -2 cho thanh ghi r2.

- Cài đặt Write:

+ Bước 1: Ta đọc tham số buffer từ thanh ghi r4, tham số charcount từ thanh ghi r5 và id từ thanh ghi r6.

+ Bước 2: Ta kiểm tra id có nằm trong bảng mô tả file không. File cần đọc có tồn tại không và file đó có type khác 2 và type khác 1 ko.

+ Bước 3: Trường hợp vi phạm 1 trong 3 điều kiện: Chương trình báo lỗi và trả về -1 cho thanh ghi r2.

+ Bước 4: Trường hợp hợp lệ thì ta lấy vị trí con trỏ trong file bằng phương thức GetCurrentPos() trong lớp FileSystem gọi là OldPos. Sau đó chép giá trị từ thanh ghi r4 từ phía User sang System bằng User2System. Giá trị chép là buffer (chuỗi ký tự).

+ Bước 5: Ta xét trường hợp type = 0, thì ta lấy vị trí con trỏ hiện tại bằng GetCurrentPos() gọi là NewPos, trả về số byte ghi được cho thanh ghi r2 bằng công thức: NewPos -OldPos.

+ Bước 6: Ta xét trường hợp type = 3, ta sẽ gọi phương thức Write của gSynchConsole để ghi từng ký tự trong buffer và kết thúc bằng ký tự xuống dòng “\n”, trả về số byte ghi được vào thành ghi r2.

III/Đa Chương, lập lịch và đồng bộ hóa trong NachOS

1. Cài đặt đa tiến trình:

- Chương trình hiện tại giới hạn chỉ thực thi 1 chương trình => cần phải có vài thay đổi trong file addrspace.h và addrspace.cc để chuyển hệ thống từ đơn chương thành đa chương. Cụ thể như sau:

    + Giải quyết vấn đề cấp phát các frames bộ nhớ vật lý sao cho nhiều chương trình có thể nạp lên bộ nhớ cùng một lúc=> sử dụng biến toàn cục Bitmap *gPhysPageBitMap để quản lý các frames.

    + Xử lý giải phóng bộ nhớ khi user program kết thúc.

    + Thay đổi đoạn lệnh nạp user program lên bộ nhớ. Hiện tại, việc cấp phát không gian địa chỉ giả thiết rằng một tiến trình được nạp vào các đoạn liên tiếp nhau trong bộ nhớ. Một khi hỗ trợ đa chương trình, bộ nhớ sẽ không còn biểu diễn liên tiếp nhau nữa=> tạo một pageTable = new TranslationEntry[numPages], tìm trang trống bằng phương thức Find() của lớp Bitmap, sau đó nạp chương trình lên bộ nhớ chính.

2. Thiết kế các lớp theo hướng đối tượng, bao gồm 4 lớp sau:

- Lớp PCB (system data segment): Lưu các thông tin để quản lý process. Cụ thể như sau:

```
private:
    Semaphore* joinsem;          // semaphore cho quá trình join
    Semaphore* exitsem;          // semaphore cho quá trình exit
    Semaphore* multex;           // semaphore cho quá trình truy xuất đọc quyền
    int exitcode;
    int numwait;                 // số tiến trình đã join
public:
    int parentID;                // ID của tiến trình cha
PCB();
PCB(int id);                     // contructor
~PCB();                          // deconstructor
// nạp chương trình có tên lưu trong biến filename và processID là pid
int Exec(char *filename, int pid);   // Tao 1 thread mới có tên là filename và process là pid
int GetID();                     // Trả về ProcessID của tiến trình gọi thực hiện
int GetNumWait();                // Trả về số lượng tiến trình chờ

void JoinWait();                 // 1. Tiến trình cha đợi tiến trình con kết thúc
void ExitWait();                 // 4. Tiến trình con kết thúc
void JoinRelease();              // 2. Báo cho tiến trình cha thực thi tiếp
void ExitRelease();              // 3. Cho phép tiến trình con kết thúc

void IncNumWait();               // Tăng số tiến trình chờ
void DecNumWait();               // Giảm số tiến trình chờ

void SetExitCode(int ec);        // Đặt exitcode của tiến trình
int GetExitCode();               // Trả về exitcode

void SetFileName(char* fn);      // Đặt tên của tiến trình
char* GetFileName();             // Trả về tên của tiến trình
```

- Lớp Ptable: dùng để quản lý các tiến trình được chạy, gồm thuộc tính pcb là một bảng mô tả tiến trình, có cấu trúc mảng một chiều có số phần tử tối đa là 10, mỗi phần tử của mảng thuộc kiểu dữ liệu PCB. Hàm constructor của lớp sẽ khởi tạo tiến trình cha (là tiến trình đầu tiên) ở vị trí thứ 0 tương đương với phần tử đầu tiên của mảng. Từ tiến trình này, chúng ta sẽ tạo ra các tiến trình con thông qua system call Exec(). Các thuộc tính và phương thức:

```
private:
    BitMap bm;                          // đánh dấu các vị trí đã được sử dụng trong pcb
    PCB* pcb[MAX_PROCESS];
    int psize;
    Semaphore* bmsem;                   // dùng để ngăn chặn trường hợp nạp 2 tiến trình cùng
lúc
public:
// khởi tạo size đối tượng PCB để lưu size process. Gán giá trị ban đầu là null
// nhớ khởi tạo bm và bmsem để sử dụng
PTable(int size);
~PTable();                              // hủy các đối tượng đã tạo
int ExecUpdate(char* name);             // Xử lý cho system call SC_Exit
int ExitUpdate(int ec);                 // Xử lý cho system call SC_Exit
int JoinUpdate(int id);                 // Xử lý cho system call SC_Join
int GetFreeSlot();                      // tìm free slot để lưu thông tin cho tiến trình mới
bool IsExist(int pid);                  // kiểm tra tồn tại processID này không?
void Remove(int pid);                   // khi tiến trình kết thúc, delete processID ra khỏi
mảng quản lý nó
char* GetFileName(int id);              // Trả về tên của tiến trình
```

- Lớp Sem: dùng để quản lý Semaphore. Các thuộc tính và phương thức:

```
private:
    char name[50];
    Semaphore *sem;                    // Tạo Semaphore để quản lý
public:
// khởi tạo đối tượng Sem. Gán giá trị ban đầu là null
// nhớ khởi tạo bm sử dụng
Sem(char* na, int i){
    strcpy(this->name,na);
    sem = new Semaphore(name,i);
}
~Sem(){                                // hủy các đối tượng đã tạo
    delete sem;
}
void wait(){                           // thực hiện thao tác chờ
    sem->P();
}
void signal(){                         // thực hiện thao tác giải phóng Semaphore
    sem->V();
}
char* GetName(){                       // Trả về tên của Semaphore
    return name;
}
```

- Lớp Stable: gồm thuộc tính semTab là một bảng mô tả semaphore, có cấu trúc mảng một chiều có số phần tử tối đa là 10, mỗi phần tử của mảng thuộc kiểu dữ liệu Sem. Các thuộc tính và phương thức:

```
private:
    BitMap* bm;                          // quản lý slot trống
    Sem* semTab[MAX_SEMAPHORE];          // quản lý tối đa 10 đối tượng Sem
public:
// khởi tạo size đối tượng Sem để quản lý 10 Semaphore. Gán giá trị ban đầu là null
// nhớ khởi tạo bm để sử dụng
STable();

~STable();                               // hủy các đối tượng đã tạo
int Create(char* name, int init);        // Kiểm tra Semaphore "name" chưa tồn tại thì tạo
Semaphore mới. Ngược lại, báo lỗi.
int Wait(char* name);                    // Nếu tồn tại Semaphore "name" thì gọi this->P()để
thực thi. Ngược lại, báo lỗi.
int Signal(char* name);                  // Nếu tồn tại Semaphore "name" thì gọi this->V()để
thực thi. Ngược lại, báo lỗi.
int FindFreeSlot(int id);                // Tìm slot trống.
```

Các biến toàn cục cần khai báo:

- Semaphore* addrLock;
- BitMap* gPhysPageBitMap;
- STable* semTab;
- PTable* pTab;

3.Cài đặt các system call

a.Cài đặt system call SpaceID Exec(char* name)

Khai báo hàm: **SpaceID Exec(char* name) ở** ./userprog/syscall.h

Exec system call sử dụng lớp PCB và Ptable để gọi thực thi một chương trình mới trong một system threadmới.

Trước khi cài đặt system call Exec ta cần cài đặt các phương thức:

Cài đặt Exec(char* name, int pid) ở lớp PCB:

- Gọi mutex->P(); để giúp tránh tình trạng nạp 2 tiến trình cùng 1 lúc.
- Kiểm tra thread đã khởi tạo thành công chưa, nếu chưa thì báo lỗi là không đủ bộ

nhớ, gọi mutex->V() và return.

- Đặt processID của thread này là id.
- Đặt parrentID của thread này là processID của thread gọi thực thi Exec
- Gọi thực thi Fork(StartProcess_2,id) => Ta cast thread thành kiểu int, sau đó khi xử ký hàmStartProcess ta cast Thread về đúng kiểu của nó.
- Trả về id.

Cài đặt ExecUpdate(char* name) ở lớp Ptable:

- Gọi mutex->P(); để giúp tránh tình trạng nạp 2 tiến trình cùng 1 lúc.
- Kiểm tra tính hợp lệ của chương trình "name".
- Kiểm tra sự tồn tại của chương trình "name" bằng cách gọi phương thức Open của lớp fileSyste

So sánh tên chương trình và tên của currentThread để chắc chắn rằng chương trình này không gọi thực thi chính nó.

- Tìm slot trống trong bảng Ptable.
- Nếu có slot trống thì khởi tạo một PCB mới với processID chính là index của slot này, parrentID là
- processID của currentThread.Đánh dấu đã sử dụng.
- Gọi thực thi phương thức Exec của lớp PCB.
- Gọi bmsem->V().
- Trả về kết quả thực thi của PCB->Exec.

Quá trình xử lý của system call Exec:

- Đọc địa chỉ tên chương trình "name" từ thanh ghi r4.
- Tên chương trình lúc này đang ở trong user space. Gọi hàm *User2System* đã được khai báo trong
- lớp *machine* để chuyển vùng nhớ user space tới vùng nhớ system space.Nếu bị lỗi thì báo "Không mở được file" và gán -1 vào thanh ghi 2.
- Nếu không có lỗi thì gọi pTab-> ExecUpdate(name), trả về và lưu kết quả thực thi phương thức này

vào thanh ghi r2.

b. Cài đặt system calls int Join (SpaceID id) và void Exit(int exitCode).

SC_Join

Khai báo hàm: **int Join (SpaceID id)** ở /userprog/syscall.h:

Join system call sử dụng lớp PCB và Ptable để thực hiện đợi và block dựa trên tham số "SpaceID id". Trước khi cài đặt system call này ta phải cài đặt các phương thức sau:

Cài đặt JoinWait() ở lớp PCB:

- Gọi joinsem->P () để tiến trình chuyển sang trạng thái block và ngừng lại, chờ JoinRelease để thực hiện tiếp.

Cài đặt ExitRelease() ở lớp PCB:

Gọi exitsem-->V() để giải phóng tiến trình đang chờ.

Cài đặt JoinUpdate(int id) ở lớp Ptable.:

- Ta kiểm tra tính hợp lệ của processID id và kiểm tra tiến trình gọi Join có phải là cha của tiến trình có processID là id hay không. Nếu không thỏa, ta báo lỗi hợp lý và trả về -1.
- Tăng numwait và gọi JoinWait() để chờ tiến trình con thực hiện.
- Sau khi tiến trình con thực hiện xong, tiến trình đã được giải phóng
- Xử lý exitcode.
- ExitRelease() để cho phép tiến trình con thoát

Quá trình xử lý của system call Join:
Đọc id của tiến trình cần Join từ thanh ghi r4.
Gọi thực hiện pTab->JoinUpdate(id) và lưu kết quả thực hiện của hàm vào thanh ghi r2.

SC_Exit

Khai báo hàm: **void Exit(int exitCode) *ở*** ./userprog/syscall.h:

Exit system call sử dụng lớp PCB và Ptable để thực hiện thoát tiến trình nó đã join.Trước khi cài đặt system call này ta phải cài đặt các phương thức sau:

Cài đặt JoinRelease() ở lớp PCB:

- Gọi joinsem->V() để giải phóng tiến trình gọi JoinWait().

Cài đặt ExitWait () ở lớp PCB:

- Gọi exitsem-->V() để tiến trình chuyển sang trạng thái block và ngừng lại, chờ ExitReleaseđể thực hiện tiếp.

Cài đặt ExitUpdate(int exitcode) ở lớp Ptable:

- Nếu tiến trình gọi là main process thì gọi Halt().
- Ngược lại gọi SetExitCode để đặt exitcode cho tiến trình gọi.
- Gọi JoinRelease để giải phóng tiến trình cha đang đợi nó(nếu có) và ExitWait() để xin tiến trình chacho phép thoát.

Quá trình xử lý của system call Exit:

- Đọc exitStatus từ thanh ghi r4
- Gọi thực hiện pTab->ExitUpdate(exitStatus) và lưu kết quả thực hiện của hàm vào thanh ghi r2.

c. Cài đặt system call int CreateSemaphore(char* name, int semval).

Như trong project 1 bạn phải cập nhật file start.s, start.c và syscall.h để thêm system call mới. Bạn tạo cấu trúc dữ liệu để lưu 10 semaphore. System call **CreateSemaphore** trả về 0 nếu thành công, ngược lại thì trả về -1.

Syscall CreateSemaphore:

- Khai báo prototype CreateSemaphore(char* name, int semval) trong ./userprog/syscall.h

- Cài đặt hàm Create(char *name, int pid) ở lớp STable.
- Quá trình xử lý của system call CreateSemaphore:
    1. Đọc địa chỉ "name" từ thanh ghi r4.
    2. Đọc giá trị "semval" từ thanh ghi r5.
    3. Tên địa chỉ "name" lúc này đang ở trong user space. Gọi hàm *User2System* đã được khai báo trong
    4. lớp *machine* để chuyển vùng nhớ user space tới vùng nhớ system space.
    5. Gọi thực hiện hàm semTab->Create(name,semval) để tạo Semaphore, nếu có lỗi thì báo lỗi.
    6. Lưu kết quả thực hiện vào thanh ghi r2.

d. Cài đặt system call int Wait(char* name), và int Signal(char* name).

Tham số name là tên của semaphore. Cả hai system call trả về 0 nếu thành công và -1 nếu lỗi. Lỗi có thể xảy ra nếu người dùng sử dụng sai tên semaphore hay semaphore đó chưa được tạo. Xem gợi ý khai báo lớp quản lý các semaphore mà Nachos tạo ra để cung cấp cho chương trình người dùng ở phụ lục.

Syscall Wait

- Khai báo prototype Wait(char* name) trong ./userprog/syscall.h
- Cài đặt hàm Wait(char *name, int pid) ở lớp STable.
- Cài đặt hàm wait() ở lớp Sem.
- Quá trình xử lí của Syscall wait:
    1. Đọc địa chỉ "name" từ thanh ghi r4.
    2. Tên địa chỉ "name" lúc này đang ở trong user space. Gọi hàm *User2System* đã được khai báo tronglớp *machine* để chuyển vùng nhớ user space tới vùng nhớ system space.
    3. Kiểm tra Semaphore "name" này có trong bảng sTab chưa, nếu chưa có thì báo lỗi.
    4. Gọi phương thức Wait() của lớp Stable.
    5. Lưu kết quả thực hiện vào thanh ghi r2.

Syscall Signal

- Khai báo prototype Signal(char* name) trong ./userprog/syscall.h
- Cài đặt hàm Signal(char *name, int pid) ở lớp STable.
- Cài đặt hàm signal() ở lớp Sem.
- Quá trình xử lí của syscall Signal:
    1. Đọc địa chỉ "name" từ thanh ghi r4.
    2. Tên địa chỉ "name" lúc này đang ở trong user space. Gọi hàm *User2System* đã

được khai báo tronglớp *machine* để chuyển vùng nhớ user space tới vùng nhớ system space.

3. Kiểm tra Semaphore "name" này có trong bảng sTab chưa, nếu chưa có thì báo lỗi.
4. Gọi phương thức Signal() của lớp Stable.
5. Lưu kết quả thực hiện vào thanh ghi r2.

4. Viết chương trình ping pong kiểm thử

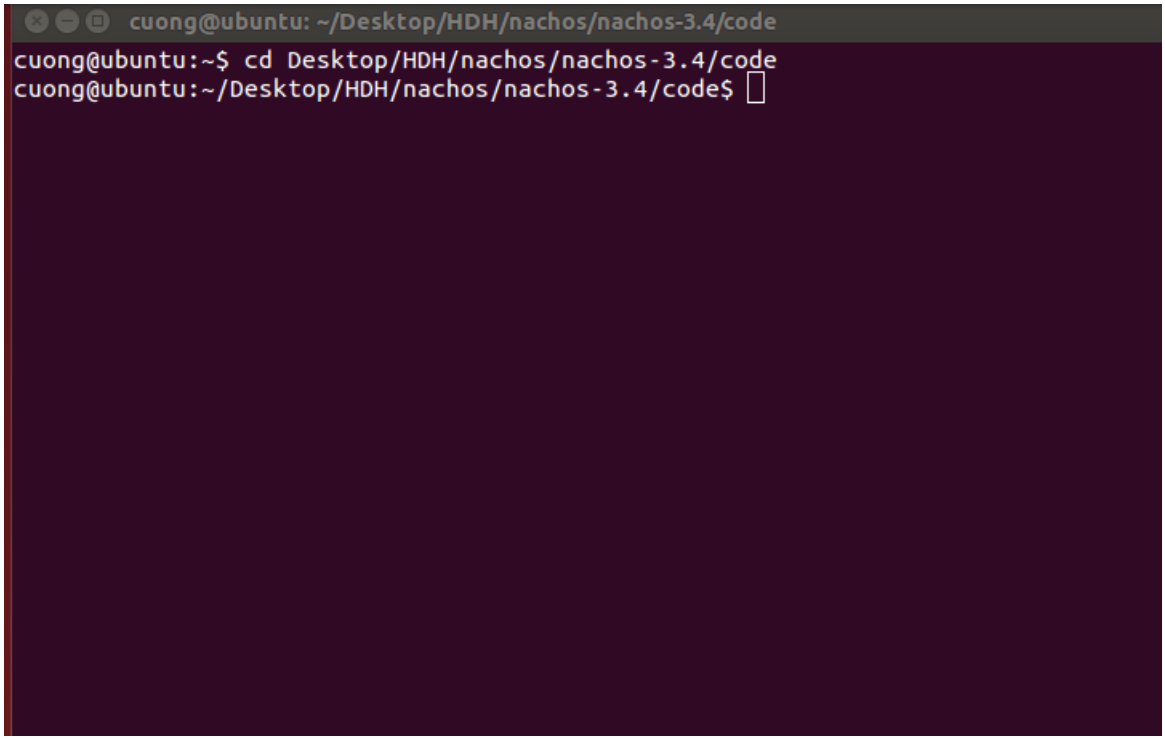
Chương trình Ping (Xuất 1000 chữ A) và Pong (1000 chữ B) ra màn hình để kiểm chứng đa chương. Kết quả: A,B được xuất xen lẫn nhau.

a. Giải thích code

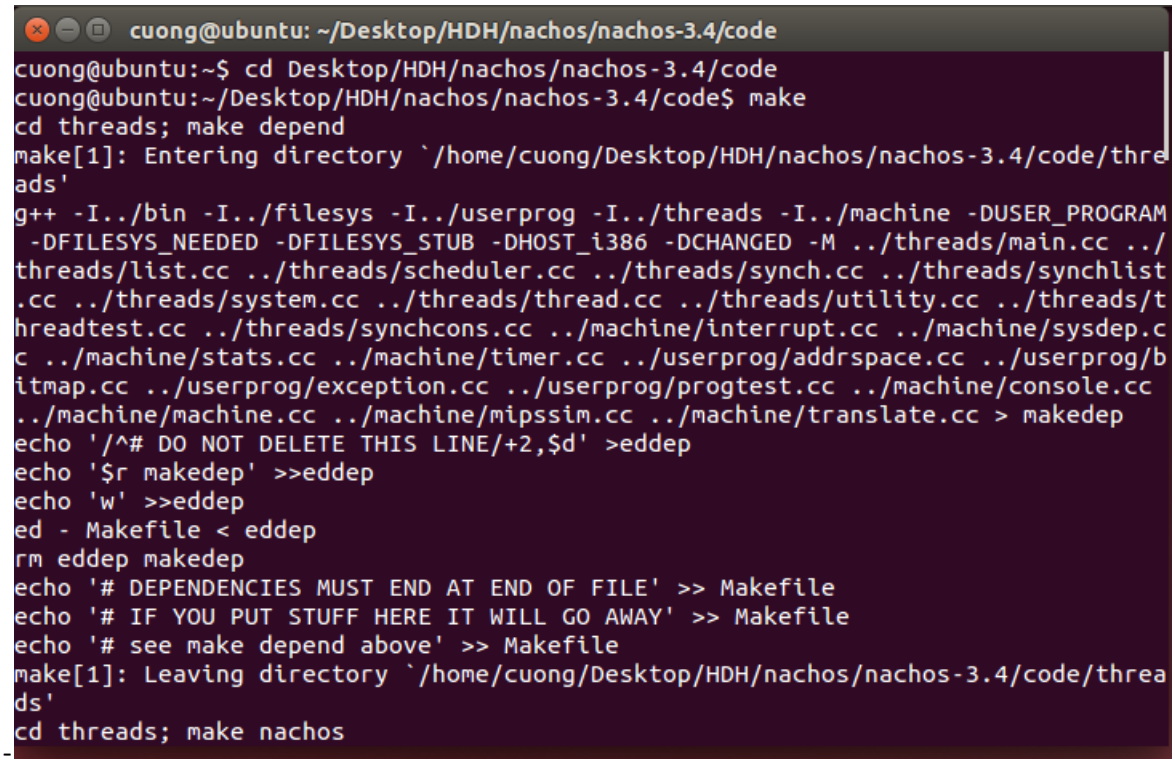
- Đầu tiên, chúng em tạo 3 file là main, ping và pong. File ping và pong dùng để in ra màn hình chữ A và B tương ứng với từng file. File Main có tác dụng gọi file ping và pong để thực thi chương trình.
- Trong file main, chúng em sẽ tạo ra 2 biến là pingId và pongId. Sau đó, gọi thực thi file ping và pong bằng hàm Exec và trả kết quả về 2 biến pingId và pongId.
- Tiếp đó, kiểm tra 2 biến pingId và pongId, nếu nó bằng -1 thì tức là không đọc được file theo đường dẫn truyền vào Exec. Lúc này màn hình console sẽ trả ra lỗi không đọc được ở file nào và kết thúc chương tình. Nếu đọc được file, nó sẽ trả ra ID của Process.
- Tiếp đó, ta gọi hàm Join và truyền vào ID của Process để xử lý phần công việc còn lại.

b. Hướng dẫn sử dụng chương trình

- Đầu tiền, ta cần cấp quyền cho các permission thực thi.
- Tiếp theo, mở terminal và dẫn đến thư mục code trong file nachos-3.4



- Thực hiện lệnh make để tạo file thực thi cho chương trình



-

-

- Gõ lệnh ./userprog/nachos -rs 1023 -x test/main để chạy chương trình nằm trong file main

```
'
cuong@ubuntu:~/Desktop/HDH4/nachos/nachos-3.4/code$ ./userprog/nachos -rs 1023 -
x test/main


Size: 2048 | numPages: 4 | PageSize: 512 | Numclear: 256

Physic Pages 0
Physic Pages 1
Physic Pages 2
Physic Pages 3
Ping-Pong:


Size: 1536 | numPages: 3 | PageSize: 512 | Numclear: 252

Physic Pages 4
Physic Pages 5
Physic Pages 6
A

Size: 1536 | numPages: 3 | PageSize: 512 | Numclear: 249

Physic Pages 7
Physic Pages 8
Physic Pages 9
AAAAAAAAAABBBAABAABBBBAAAAABAAAAAABAAAAAABBBBBBBAABBBAAABBBAABAABBBAAAAAABBAAAAA
AAABBBBBAAAABAABBBABAABBBBBBBBBABAABAABBBBBBBBBAABBBBBAAABBBBBBBBBBBBBBBBAAABAAA
BAAAAABBBBBBAABBBBBBBBBAAAAAABBBABAABAAABBAAAAABAAAAAABBBAABBBBBAAABABBBBAAABABBB
BAAAABBBAAAABAAAAAAAAAAAABBBBBAAAAABBBABBBBBAAABBBBAAAAAABBBBBBBBAABBBBBBBBBBBB
BBAABAABBBBAAAAABBAAAABBBBBBBABBAAAABBBBBBBBBABABBAAAAABBBBBAAABAAAAAAABBBAABBB
ABBBBBBBBBBAAABAAABABBBBBBBBBBAABBAAAABBBBBBBBBAABBBAAAAAAAAABAABAAAABBAABBBAA
ABBBBBAAAAAABBBBBBBBAABBBBABBBBAAAAAAAAAAABBAAAABBAAABBBBBAABBBBBBBBAAAAAAAAAA
BBBBBBAAAAAAAAAAABBBAAAAAABBBAAAAAABBBBBBBBBBAAAABBAAAABBAABBBBAAAAAABBBBBBB
AABBBBAABBBBBBBBBBAABBBAABAAAAAABBAAAAAAAABBAAAABBBBBBBBAAAAAAAABBABBAABBBBBBB
BBBBAAAABAAAAAAAABBBBBAAAABBBBBAAABBAAABAAAABABBBBAAAAAAABBBABBBBBBBBBAAABA
AAABAAABBBABBBBBBBBBBBBBAABBBBBBAAAAABBBBAABBBBBBBBBBBBBBBBBBAAAAAAAAABBBBBB
BAABBBBBBAABBBBBBAAABBBBBBBBBBBBBBAAAABBBBBAAAAAABBBBBBABBBBBAAAAAAABBBBBBBBB
BBBBBBBBBBAAAAABBBBAAAAAABBAABAAABABBAAAAABBBBABAABBBBBBBBABBBBAAAABBBBBAABB
BBBBABBAABBBBBBBBAAABBBBAABBBBAAAAAAAABAABBBBAAAABBABBBBAAABBBBAABBBABAAAABB
BBAAAABBBBBAABBBBBBBBBAABBAAAAAAAABBBAAAAABAAABBBBBAAAABBBBBBAAAAAABBBBBBAA
AAAAAAAAAAAAAAABBAAAAAAABBBBAAAAAAAABBBBBBBBBAAABBAAAAAAAAAABBAAAAAABBBBBB
BBAAABBABBABAAAAAABBBBBBBAAAAAAAAAABBABBBBBBBBBABBBBBBAAAAAAAAAAAAAAAAAAA
AAAAABBABBBBAABBBBABBBAAABBBAABBBBBBBBBBBBBBBBAAABBBBBBBBAABBBABBBAABBBAAAAA
ABBABBAAAABBBBBBBBBBAAAABAAAAAAAAAABAAABBAAAAABBBBBBBAAAABAAAAAAAABBBAABBB
BBAAABAAAABBBBBABBABBAAABBBAAAAAAAAAABBBBBABBBBABBBAAABBBBBAAABBBBAAAAAAA
BBBBBBBBBBABBBAAABBBBBBAAAAAAAAABAAAABBBBAAABBBBBBAAAABAAABBBABABBBBBAAAAAAB
```

IV.Tài Liệu Tham Khảo

1. Biên dịch và cài đặt NachOS – file được cung cấp bởi giáo viên hướng dẫn