

**TRƯỜNG ĐẠI HỌC KHOA HỌC TỰ NHIÊN**

**KHOA CÔNG NGHỆ THÔNG TIN**

**BỘ MÔN HỆ ĐIỀU HÀNH**

**-----🙞🕮🙜-----**

**BÁO CÁO**

**ĐỒ ÁN**



**Đề Tài : Syscall về đa chương, đa luồng**

Sinh viên : **HUỲNH TRỌNG THOẠI – 1512551**

Sinh viên : **VÕ MINH TRÍ – 1512607**

Sinh viên : **NGUYỄN ANH TUẤN – 1512636**

Giảng viên : **LÊ QUỐC HÒA**

Giảng viên : **CHUNG THÙY LINH**

Lớp : **TH2015/4**

**Mục lục**

[**I.** **Mức độ hoàn thành đồ án:** 3](#_Toc502754697)

[**II.** **Phân công** 5](#_Toc502754698)

[**III.** **Mô tả thiết kế** 6](#_Toc502754699)

[**1.** **Mô tả** 6](#_Toc502754700)

[**2.** **Pcb.cc** 6](#_Toc502754701)

[**3.** **Ptable** 7](#_Toc502754702)

[**4.** **Định nghĩa hàm StartProcess** 7](#_Toc502754703)

[**IV.** **Nội dung chi tiết các class chính** 7](#_Toc502754704)

[**1.** **Nội dung file pcb.cc** 7](#_Toc502754705)

[**2.** **Nội dung file ptable.cc** 8](#_Toc502754706)

[**3.** **Sem: dùng để quản lý Semaphore** 9](#_Toc502754707)

[**4.** **Stable: quản lý 10 Semaphore.** 9](#_Toc502754708)

[**V.** **Nội dung** 11](#_Toc502754709)

[**1.** **Cách cài đặt Nachos** 11](#_Toc502754710)

[**2.** **Thêm lớp syschcons vào nachos** 11](#_Toc502754711)

[**3.** **Exceptions và system calls** 12](#_Toc502754712)

[***a.*** ***Viết lại exception.cc*** 12](#_Toc502754713)

[***b.*** ***Viết lại cấu trúc điều khiển của chương trình để nhận các Nachos system calls.*** 12](#_Toc502754714)

[***c.*** ***Viết mã để tăng giá trị biến program counter.*** 12](#_Toc502754715)

[**4.** **Cài đặt đa tiến trình** 13](#_Toc502754716)

[**5.** **Cài đặt system call SC\_Exec** 13](#_Toc502754717)

[**6.** **Cài đặt system call SC\_Join** 14](#_Toc502754718)

[**7.** **Cài đặt system call SC\_Exit** 14](#_Toc502754719)

[**8.** **Cài đặt SC\_CreateSemaphore** 15](#_Toc502754720)

[**9.** **Cài đặt SC\_Up** 15](#_Toc502754721)

[**10.** **Cài đặt system call SC\_Down** 15](#_Toc502754722)

[**11.** **Chương trình shell.** 16](#_Toc502754723)

1. **Mức độ hoàn thành đồ án:**

|  |  |
| --- | --- |
| Chức năng chương trình | Mức độ |
| 1. Viết lại exception.cc. | Đã hoàn thành. |
| 1. Viết lại cấu trúc điều khiển của chương trình để nhận các Nachos system calls. | Đã hoàn thành. |
| 1. Viết mã để tăng giá trị biến program counter. | Đã hoàn thành. |
| 1. Cài đặt đa tiến trình | Đã hoàn thành. |
| 1. Cài đặt system call   SpaceID Exec(char\* name). | Đã hoàn thành. |
| 1. Cài đặt system call   int Join(SpaceID id). | Đã hoàn thành. |
| 1. Cài đặt system call   void Exit(int exitcode). | Đã hoàn thành. |
| 1. Cài đặt system call   int CreateSemaphore(char\* name, int semval). | Đã hoàn thành. |
| 1. Cài đặt system call   int Up(char\* name). | Đã hoàn thành. |
| 1. Cài đặt system call   int Down(char\* name). | Đã hoàn thành. |
| 1. Cài đặt chương trình shell. | Chưa hoàn thành |
| **Tổng kết mức độ hoàn thành yêu cầu đồ án**: | **95%** |

1. **Phân công**

|  |  |  |
| --- | --- | --- |
| Huỳnh Trọng Thoại 1512551 | Võ Minh Trí 1512607 | Nguyễn Anh Tuấn 1512636 |
| \_ Thay đổi mã cho các exception.  \_Cài đặt đa tiến trình | \_Cài đặt system call CreateSemaphore, Down, Up.  \_ Viết chương trình Shell. | \_Cài đặt system call SpaceID Exec  \_ Cài đặt system call Join,Exit. |

1. **Mô tả thiết kế**

1. **Mô tả**

JoinWait()

ExitRelease()

JoinRelease()

ExitWait()

**Tiến trình cha**

**Tiến trình con**

Xử lý exitcode …

Khi kết thúc

Gán exitcode

1. **Pcb.cc**

* Quản lý một tiến trình bao gồm những thông tin về tiến trình đó.
* Tiến hành excute thông qua phương thức Exec(char\* filename,int pID) trả về ID của tiến trình đó.
* Mỗi tiến trình được cấp phát một stark riêng khi khởi chạy.
* Khởi tạo currentThread để chạy tiến trình “main”.

1. **Ptable**

* Quản lý những tiến trình (tối đa 10 tiến trình).
* Mặc định khới tạo một tiến trình ban đầu cho tiến trình cha.
* Các tiến trình con có dạng PCB\* pcb[];

1. **Định nghĩa hàm StartProcess**

* Mở một file với tên là tham số của hàm StartProcess.
* Sử dụng phương thức khởi tạo AddrSpace(char\* filename) để cấp phát vùng nhớ cho tiến trình đang chạy.
* Thiết lập những giá trị thanh ghi ban đầu cũng như phục hồi trạng thái thanh ghi.
* Chạy tiến trình và không bao giờ trả về giá trị.

1. **Nội dung chi tiết các class chính**
2. **Nội dung file pcb.cc**

* PCB::PCB(int id) : contructor của class PCB
* PCB::~PCB: dectructor của class PCB
* int PCB::Exec(char \*filename, int pID): tạo một tiến trình có tên filename và processID là pID.
* int PCB::GetID() : trả về giá trị là một id của tiến trình.
* int PCB::GetNumWait(): trả về số tiến trình đã chờ đợi.
* void PCB::JoinWait(): Tiến trình cha đợi tiến trình con kết thúc bằng cách lấy thuộc tính joinsem trỏ tới hàm chờ đợi P().
* void PCB::ExitWait(): Tiến trình con kết thúc bằng cách lấy thuộc tính exitsem trỏ tới hàm chờ đợi P().
* void PCB::JoinRelease(): Báo cho tiến trình cha thực thi tiếp bằng cách lấy thuộc tính joinsem trỏ tới hàm giải phóng semaphore V().
* void PCB::ExitRelease(): Cho phép tiến trình con kết thúc bằng cách lấy thuộc tính exitsem trỏ tới hàm giải phóng semaphore V().
* void PCB::IncNumWait(): Tăng số lượng tiến trình chờ đợi.
* void PCB::DecNumWait(): Giảm số lượng tiến trình chờ đợi.
* void PCB:: SetExitCode(int ec) : đặt exitcode của tiến trình.
* int PCB::GetExitCode(): trả về giá trị exitcode của tiến trình.
* void SetFileName(char\* fn) : đặt tên cho tiến trình
* char GetFileName(): trả về tên của tiến trình bằng cách dùng thuộc tính thread trỏ tới hàm getName() của chính nó.

1. **Nội dung file ptable.cc**

* Ptable::Ptable(int size): Contructor của class Ptable.
* Ptable:: ~PTable(): Destructor của class Ptable dùng để giải phóng các thuộc tính là con trỏ.
* int PTable::ExecUpdate(char\* filename): xử lý cho system call SC\_Exec.
* int PTable::ExitUpdate(int ec):
* int PTable::JoinUpdate(int pID):
* int PTable::GetFreeSlot(): Tìm vị trí trống để lưu thông tin cho tiến trình bằng cách sử dụng con trỏ BitMap trỏ hàm tìm vị trí trống Find().
* bool PTable::IsExist(int pID): Kiểm tra xem có tồn tại tiến trình pID này không?
* void PTable::Remove(int pID): Xóa một tiến trình pID ra khỏi mảng quản lý của nó khi mà tiến trình này đã kết thúc.
* char\* PTable::GetName(int pID): Trả về tên tiến trình có processID là pID.

1. **Sem: dùng để quản lý Semaphore**

* Sem::Sem(char\* na, int i): Contructor của class Sem.
* Sem::~Sem(): Destructor của class Sem. Giải phóng con trỏ sem.
* void Sem::wait(): Thực hiện thao tác chờ bằng cách sử dụng con trỏ Semaphore trỏ tới hàm chờ đợi P().
* void Sem::signal(): Thực hiện thao thác giải phóng Semaphore bằng cách sử dụng con trỏ Semaphore trỏ tới hàm giải phóng V().
* char\* Sem::GetName(): Trả về tên của Semaphore.

1. **Stable: quản lý 10 Semaphore.**

* STable::STable(): Contructor của class Stable.
* STable::~STable(): Destructor của class Stable.
* int STable::Create(char\* name, int init): Tạo ra một Semaphore.
* int STable::Wait(char\* name): Thực hiện thao tác chờ của Semaphore có tên là name.
* int STable::Signal(char\* name): Thực hiện thao tác giải phóng Semaphore có tên là name.

1. **Nội dung**
2. **Cách cài đặt Nachos**
   * + Cài đặt VMWARE 8.0
     + Nhập mật khẩu: 123456
     + Share 2 file nachos và synchcons (VM-> setting-> Options-> Shared Folders -> Always enabled-> Add-> Browse-> Next)
     + Tìm vị trí của nó: Computer-> Filesystem-> mnt-> hgfs-> nachos & synchcons
3. **Thêm lớp syschcons vào nachos**
   * + Copy 2 file Synchcons.c và Synchcons.h vào code-> threads.
     + Vào code-> threads-> system.h khai báo #include "synchcons.h".
     + Vào code-> threads-> system.c khai báo một biến cục bộ (Synchconsole \*gsc), cấp phát nó và giải phóng vùng nhớ.
     + Do mình khai báo nó là một biến cục bộ nên trong file system.h phải khai báo (extern Synchconsole \*gsc).
     + code-> Makefile.common khai báo 3 file Synchcons.cc, Synchcons.h, Synchcons.o trong USERPROG\_C, USERPROG\_H, USERPROG\_O.
4. **Exceptions và system calls**
5. ***Viết lại exception.cc***

* Vào code-> machine-> machine.h để xem các loại exception.
* Vào code-> userprog-> exception.cc viết thêm các exception.
* Vào code chuột phải chọn Open in Terminal.
* Chạy bằng lệnh gmake all để kiểm tra.
* Chạy ./userprog/nachos -rs 1023 -x ./test/halt để xuất kết quả.

1. ***Viết lại cấu trúc điều khiển của chương trình để nhận các Nachos system calls.***

* Vào code-> userprog-> syscall.h để xem các loại system calls.
* Chuyển code if…case về switch…case cho tiện trong việc kiểm tra.
* Viết các loại system calls trong case SyscallException.

1. ***Viết mã để tăng giá trị biến program counter.***

* Vào code-> machine-> mipssim.cc để xem đoạn mã.
* Vị trí trước đó gán cho vị trí hiện tại.
* Vị trí hiện tại gán cho vị trí tiếp theo.
* Vị trí tiếp theo cộng 4 byte (1 đơn vị trong PC).
* Nếu không lập trình đúng phần này thì Nachos sẽ bị vòng lặp gọi thực hiện system call này mãi mãi.

1. **Cài đặt đa tiến trình**

* Khai báo 2 biến toàn cục Lock\* addrLock, BitMap \*gPhysPageBitMap, cấp phát và giải phóng nó trong system.h và system.cc.
* Thay đổi class Bitmap và Lock trong addrspace.h và addrspace.cc trong userprog bằng cách thêm hàm AddrSpace(char \*filepath) để nó cấp phát cho chương trình một vùng nhớ cho nó thực thi.
* Thêm thuộc tính: int processID vào threads.h để khai báo id của một tiến trình.
* Thêm tên 8 tập tin vào thư mục thread và vào Makefile.common để viết quá trình tạo ra file pcb.o,ptable.o,sem.o,stable.o.
* Để tiện quản lý đa tiến trình, ta khai báo thêm 2 biến toàn cục là con trỏ PTable,STable trong threads.h, threads.cc.

1. **Cài đặt system call SC\_Exec**

* Đọc địa chỉ tên chương trình từ thanh ghi r4 bằng machine->ReadRegister(4) và sẽ trả về số nguyên địa chỉ của kí tự đầu tiên tên chương trình.
* Gọi hàm User2System đã được khai báo trong lớp machine để chuyển vùng nhớ user space tới vùng nhớ system space.
* Kiểm tra có mở được file không bằng cách sử dụng con trỏ fileSystem trỏ tới hàm Open(char\*). Nếu như kết quả trả về là NULL thì xuất ra thông báo “Không mở tập tin” và ghi giá trị -1 vào thanh ghi r2.
* Chạy tiến trình khi sử dụng con trỏ ptable trỏ tới ExecUpdate(char\*), trả về id và ghi nó vào thanh ghi r2.

1. **Cài đặt system call SC\_Join**

* Đọc id của tiến trình cần join từ thanh ghi r4 bằng machine->ReadRegister(4).
* Gọi thực hiện ptable->JoinUpdate(id) và lưu kết quả trả về là một exitcode vào thanh ghi r2 bằng machine->WriteRegister(2,exitcode).

1. **Cài đặt system call SC\_Exit**

* Đọc exitStatus của tiến trình đã join từ thanh ghi r4 bằng machine->ReadRegister(4).
* Gọi thực hiện ptable->ExitUpdate(exitStatus) và lưu kết quả trả về là một exitcode vào thanh ghi r2 bằng machine->WriteRegister(2,exitcode).

1. **Cài đặt SC\_CreateSemaphore**

* Đọc địa chỉ của tên Semaphore cần tạo từ thanh ghi r4 bằng machine->ReadRegister(4).
* Gọi hàm User2System đã được khai báo trong lớp machine để chuyển vùng nhớ user space tới vùng nhớ system space.
* Đọc giá trị semval từ thanh ghi r5 bằng machine ->ReadRegister(5).
* Gọi thực hiện hàm sTable->Create(namefile,semval) để tạo Semaphore, nếu có lỗi thì báo lỗi.
* Lưu kết quả là id của Semaphore vào thanh ghi r2 bằng machine->WriteRegister(2,id);

1. **Cài đặt SC\_Up**

* Đọc địa chỉ của tên Semaphore cần tạo từ thanh ghi r4 bằng machine->ReadRegister(4).
* Gọi hàm User2System đã được khai báo trong lớp machine để chuyển vùng nhớ user space tới vùng nhớ system space.
* Gọi phương thức Signal() của class Stable, trả về một biến check.
* Nếu check = -1 thì báo lỗi và lưu -1 vào thanh ghi r2.
* Ngược lại, lưu kết quả vào thanh ghi r2 bằng machine->WriteRegister(2,check) .

1. **Cài đặt system call SC\_Down**

* Đọc địa chỉ của tên Semaphore cần tạo từ thanh ghi r4 bằng machine->ReadRegister(4).
* Gọi hàm User2System đã được khai báo trong lớp machine để chuyển vùng nhớ user space tới vùng nhớ system space.
* Gọi phương thức Wait() của class Stable, trả về một biến check.
* Nếu check = -1 thì báo lỗi và lưu -1 vào thanh ghi r2.
* Ngược lại, lưu kết quả vào thanh ghi r2 bằng machine->WriteRegister(2,check) .

1. **Chương trình shell.**

* Xây dựng chương trình nhập 1000 chữ A và 1000 chữ B xen kẽ nhau