HỆ ĐIỀU HÀNH

Project 2

Danh sách các thành viên:

1. Huỳnh Ngô Trung Trực 19120040
2. Nguyễn Lê Bảo Thi 19120376
3. Lê Trung Hiếu 19120507

MỤC LỤC

1. Thông tin thành viên

MSSV	Họ Tên	Các công việc thực hiện
19120040	Huỳnh Ngô Trung Trực	CreateFile, Sem, STable, Exit, Join, Chương trình minh họa, viết báo cáo
19120376	Nguyễn Lê Bảo Thi	Open, Close, FileTable, CreateSemaphore, Signal, Wait, viết báo cáo
19120507	Lê Trung Hiếu	Read, Write, PTable, PCB, Exec, viết báo cáo

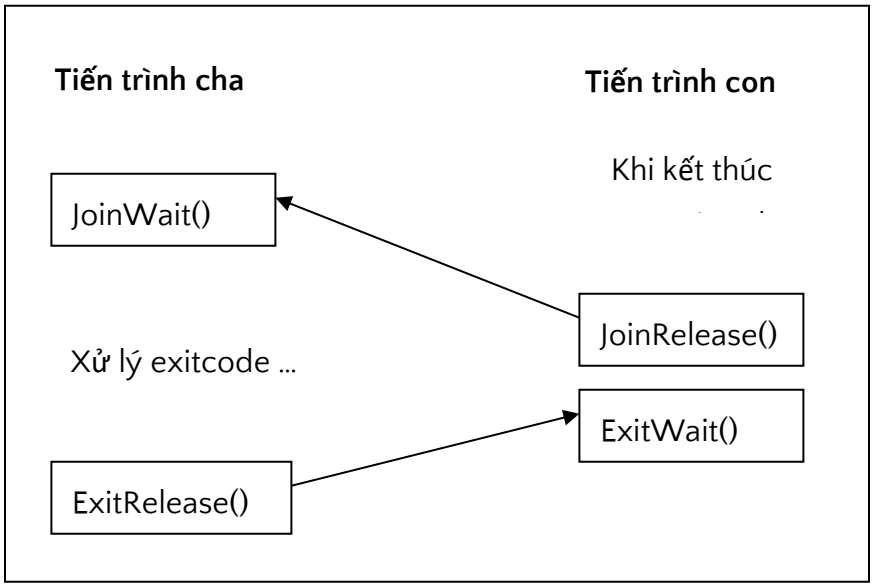
2. Mô tả chi tiết chương trình

2.1. Mô hình và mô tả chi tiết mô hình

2.1.1. Đa chương và đồng bộ hóa

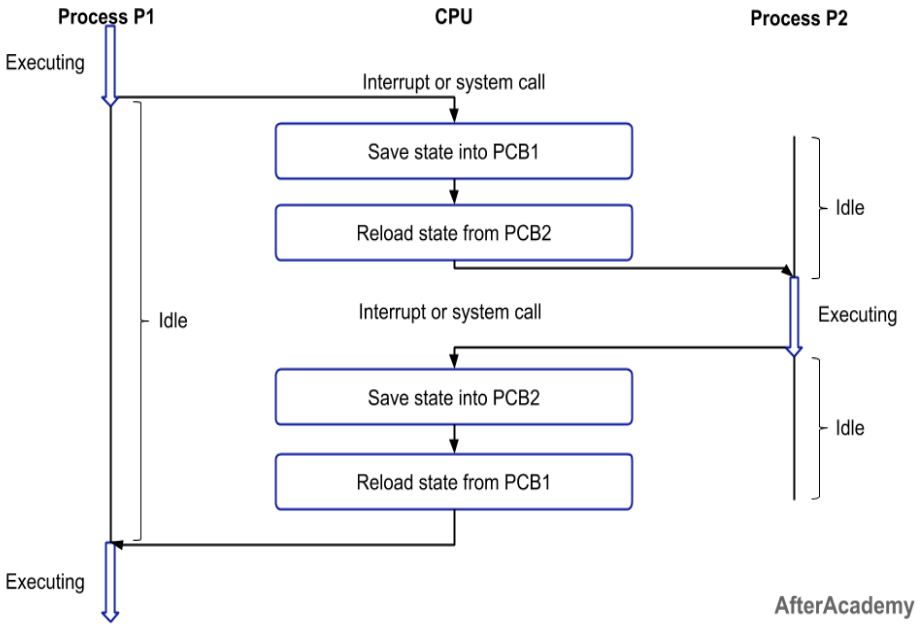
Chúng ta dùng lớp PTable để quản lý thông tin các tiến trình. Lớp PTable quản lý 10 tiến trình, mỗi tiến trình được mô tả chi tiết và các hành động của nó được cài đặt trong lớp PCB và class Bitmap để lưu vết số lượng process hiện hành.

Sơ lược vòng đời của 1 process:



- Nếu là tiến trình đầu tiên thì processID của nó là 0.
- Nếu là tiến trình con x, sau lời gọi system call Exec() thì nó sẽ gọi Join() để vào tiến trình cha (Báo cho tiến trình cha phải đợi cho tiến trình con x kết thúc mới thực thi tiếp). Khi đó, hệ điều hành sẽ gọi pcb[x]->JoinWait() (semaphore đợi cho đến khi x kết thúc) và nhận mã exitcode để xử lý cần thiết và đến khi x gọi Exit() (gửi signal cho tiến trình cho thực thi tiếp và đợi đến khi tiến trình cha cho phép nó thoát) thì hệ điều hành sẽ gọi pcb[x]->JoinRelease() và gọi pcb[x]->ExitWait() để xin phép kết thúc, hệ điều hành lúc đó sẽ gọi pcb[x]->ExitRelease() để cho phép tiến trình con kết thúc.

Khái niệm Context Switch (Chuyển đổi ngữ cảnh): Là một quá trình lưu lại trạng thái hiện hành và chuyển sang trạng thái khác của CPU giúp chii nhiều tiến trình có thể chia sẻ một CPU duy nhất. Đây là đặc tính tiêu biểu của hệ điều hành đa nhiệm (multitasks).



2.1.2. Quản lý hệ thống file mở của tiến trình (Nhập xuất file)

Thay vì sử dụng lớp FileSystem trong filesys để quản lý bảng mô tả file. Chúng ta sẽ thay thế bằng cách quản lý bảng mô tả file của mỗi tiến trình bằng cách sử dụng FileTable của mỗi PCB của mỗi tiến trình. Ở trong lớp PCB có biến fileTable dùng để quản lý hệ thống file mở của tiến trình được mô tả chi tiết và hành động của nó được cài đặt trong lớp FileTable. Mỗi tiến trình sẽ được cấp 1 bảng mô tả file đang được mở với kích thước cố định là 10 file. Trong đó, 2 phần tử đầu, ô 0 và ô 1 để dành cho console input và console output. Các hàm xử lý sẽ tương tự như filesys_stub trong filesystem.

2.2. Các thông số, biến toàn cục và hàm quan trọng

2.2.1. Chỉnh sửa thông số và cài đặt biến toàn cục

Các thông số được chỉnh sửa:

- PageSize: sửa thành 512 để giảm số lượng Page.
- NumPhysPages: sửa thành 512 để tăng số lượngPage có sẵn.

Các biến toàn cục:

- kernel: là một con trỏ của lớp Kernel, chứa tất cả các biến được sử dụng để thực thi chương trình. Trong đồ án này, chúng em có thêm 3 biến vào lớp Kernel là addrLock, pTable và sTable lần lượt là các con trỏ quản lý process và semaphore của toàn bộ chương trình.
- debug: là một con trỏ của lớp Debug, phục vụ việc debug.

2.2.2. Cài đặt Addrspace hoạt động đa chương

Addrspace là lớp dùng để quản lý không gian địa chỉ của người dùng chịu các trách nhiệm như: Cấp phát không quan địa chỉ cho tiến trình, thu hồi không gian đã cấp phát, lưu và phục hồi trạng thái khi xảy ra context-switching.

Ý tưởng cài đặt:

- Tạo 2 constructor mới của Addrspace gồm có Addrspace(char* filename) và Addrspace(OpenFile* executable).
- 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. Vì khi hỗ trợ đa chương, bộ nhớ sẽ không cho một tiến trình được nạp vào các đoạn liên tiếp nhau nữa.

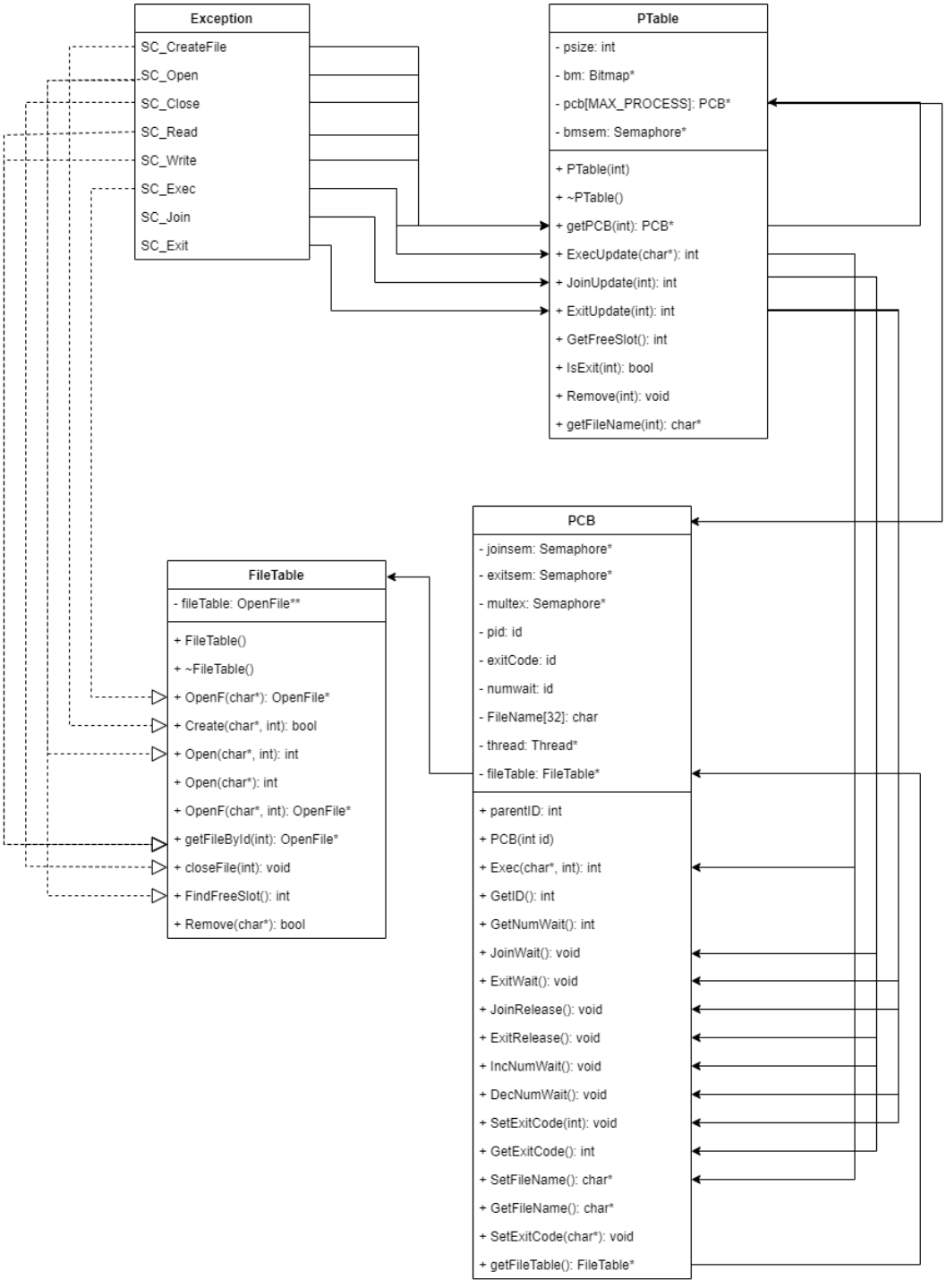
Thực hiện:

- Tạo một pageTable = new TranslationEntry[numPages], tìm trang còn trống bằng phương thức FindAndSet() của lớp Bitmap có sẵn, sau đó nạp chương trình lên bộ nhớ chính bằng pageTable[i].physicalPage=gPhysPageBitmap->FindAndSet()
- Kiểm tra số trang có lớn hơn số trang còn trống nếu có thông báo lỗi.
- Xử lý Readonly Data

2.2.3. StartProcess:

Khởi tạo một user space trong 1 system thread, tạo một Addrspace mới và gán space của current thread hiện tại trỏ vào addrspace mới tạo.

2.3. Thiết kế và mô tả chi tiết thiết kế



2.3.1. Lớp FileTable

Ý nghĩa: Dùng để quản lý file được mở của một tiến trình. (Gồm 1 thuộc tính là bảng mô tả file gồm 10 file của một tiến trình)

Các thuộc tính quan trọng:

- OpenFile** openFile: Dùng để quản lý mô tả file của một tiến trình

Các hàm quan trọng:

- FileTable(): Khởi tạo mảng gồm các OpenFile* với số lượng phần tử là 10. Lúc đầu bảng mô tả file gồm có phần tử thứ nhất là stdin, phần tử thứ hai là stdout và các phần tử còn lại bằng NULL.
- OpenFile* OpenF(char* name): Mở một file (mở bằng UNIX)
- bool Create(char* name, int initialSize): Tạo một file (tạo bằng UNIX). Nếu tạo được trả về TRUE, không tạo được trả về FALSE.
- int Open(char* name, int type): Hàm mở file được overload lại để mở file với 2 loại khác nhau. Nếu có thể tạo file và còn slot trống thì sẽ tạo file và trả về slot trống đó nếu không thì sẽ trả về -1.
- OpenFile* getFileById(int id): Trả về OpenFile* là giá trị đang được lưu trữ trong vị trí id.
- void Close(int id): Giải phòng vùng nhớ của vị trí file của tiến trình đang gọi muốn xóa. Và gán nó lại bằng NULL.
- int FindFreeSlot(): Hàm tìm slot còn trống trong bảng mô tả file. Nếu có sẽ trả về slot trống, không thì trả về -1.

2.3.2. Lớp PCB

Các thuộc tính quan trọng:

- int pid: định danh các tiến trình
- Thread* thread để lưu tiến trình đã nạp
- int parentID: id của tiến trình cha
- char[32] FileName: tên của tiến trình
- Semaphore joinsem, exitsem, multex: dùng để quản lý quá trình join, exit, nạp chương trình.

Các hàm quan trọng:

- PCB(int id): khởi tạo PCB, gán giá trị cho các thuộc tính và semaphore.
- int Exec(char* filename,int pid): Tạo 1 thread mới và cấp phát vùng nhớ mới cho thread này.

- void JoinWait(): chờ đến khi tiến trình con JoinRelease()
- void ExitWait(): chờ đến khi tiến trình cha ExitRelease()
- void JoinRelease(): tăng joinsem lên để tiến trình cha tiếp tục thực hiện xử lý exitcode.
- void ExitRelease(): tăng exitsem lên để tiến trình con được kết thúc.
- FileTable* getFileTable(): truy cập filetable của pcb, dùng khi xử lý các system call về file.

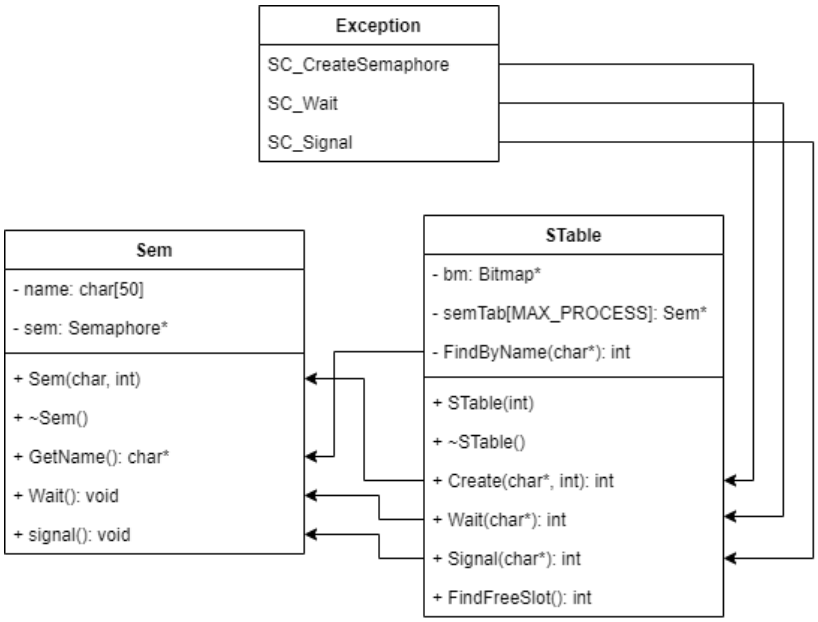
2.3.3. Lớp PTable

Các thuộc tính quan trọng:

- int pSize: Kích thước của pTable
- Bitmap *bm: Để đánh dấu các ô đã sử dụng trong pTable.
- PCB* pcb[MAX_PROCESS]: mảng quản lý các pcb, MAX_PROCESS ở đồ án này là 10, tức là 1 filetable quản lý 10 pcb.
- Semaphore* bmsem: semaphore tránh nạp nhiều tiến trình cùng 1 lúc, tại 1 thời điểm chỉ có thể nạp 1 tiến trình (Thêm pcb vào pTable).

Các hàm quan trọng:

- PTable(int size): Khởi tạo một ptable với kích thước size, khởi tạo các biến cần thiết và tạo một pcb đại diện cho chương trình cha (scheduler) ở vị trí 0.
- int ExecUpdate(char*): Thực thi cho system call SC_EXEC, kiểm tra chương trình được gọi có tồn tại trong máy không. Kiểm tra thử xem chương trình gọi lại chính nó không? . Kiểm tra còn slot trống để lưu tiến trình mới không (max là 10 process). Nếu thỏa các điều kiện trên thì ta lấy index của slot trống là processID cảu tiền trình mới tạo này, giả sử là ID. Và gọi phương thức EXEC của lớp PCB với đối tượng tương ứng quản lý process này (pcb[index]->EXEC(...)).
- int ExitUpdate(int): Được sử dụng bởi SC_Exit, kiểm tra thử PID có tồn tại không, sau đó chúng ta mới xử lý kết thúc tiến trình, gọi JoinRelease để giải phòng sự chờ đợi của tiến trình cha và ExitWait để xin phép tiến trình cha cho kết thúc, sau khi gọi thủ tục này, chúng ta phải giải phóng tiểu trình hiện tại và nếu là main process thì chúng ta gọi hàm Halt luôn.
- int JoinUpdate(int): Được sử dụng cho system call SC_Join, trong thủ tục này thì tiến trình cha goi pcb[i]->JoinWait() để chờ cho tới khi tiến trình con kết thúc (trước khi tiến trình con kết thúc thì tiến trình con gọi JoinRelease()). Sau đó tiến trình cha gọi ExitRelease() để cho phép tiến trình con được kết thúc. Ta không được cho phép tiến trình join vào chính nó, hoặc join vào tiến trình khác không phải là cha của nó.
- int GetFreeSlot(): Tìm ô trống trong ptable để lưu pcb mới.
- PCB* getPCB(int id): lấy 1 pcb ở vị trí id.



2.3.4. Lớp Sem

Ý nghĩa: Dùng để quản lý Semaphore.

Các thuộc tính quan trọng:

- char[50] name: Tên của semaphore.
- Semaphore* sem: Tạo Semaphore để quản lý.

Các hàm quan trọng:

- Sem(char* na, int i): Khởi tạo đối tượng Sem, gán giá trị ban đầu là NULL.
- void Wait(): Thực hiện thao tác chờ.
- void signal(): Thực hiện thao tác giải phóng Semaphore.
- char* GetName(): Trả về tên của Semaphore.

2.3.5. Lớp Stable

Ý nghĩa: Quản lý các semaphore (Gồm thuộc tính semTab là một bảng mô tả semaphore)

Các thuộc tính quan trọng:

- Bitmap* bm: Để đánh dấu các ô đã sử dụng.
- Sem* semTab[MAX_SEMAPHORE]: Mảng chứa các Sem.

Các hàm quan trọng:

- Stable(): Khởi tạo size đối tượng Sem để quản lý 10 semaphore. Gán giá trị ban đầu là NULL

- int FindByName(char* name): Tìm ID của semaphore bằng tên. Nếu không có trả về -1
- int Create(char* name, int init): Kiểm tra Semaphore có tên name tồn tại chưa, nếu chưa tồn tại thì tạo Semaphore mới
- int Signal(char* name): Kiểm tra semaphore có tên name tồn tại thì gọi signal()
- int FindFreeSlot(): Tim slot trống

2.4. Các system call

2.4.1. int CreateFile(char *name):

Ý nghĩa: Giúp user program tạo ra một file rỗng có tên là name.

Tham số truyền vào: Tên của file cần tạo.

Kết quả: Trả về 0 nếu tạo thành công và trả về -1 nếu có lỗi.

2.4.2. OpenFileID Open(char *name, int type)

Ý nghĩa: Giúp User program có thể mở 2 loại file, file chỉ đọc và file đọc và ghi (dựa vào tham số type được truyền vào). System call này có nhiệm vụ chuyển đổi địa chỉ buffer trong user space khi cần thiết và viết hàm xử lý phù hợp trong kernel (Tìm vị trí trống trong bảng mô tả file của tiến trình và tiến hành mở file nếu còn slot trống).

Tham số truyền vào: Tên của file muốn mở, loại file muốn mở (type = 0 cho mở file đọc và ghi, type = 1 cho file chỉ đọc)

Kết quả: Trả về id của file (là vị trí được lưu trữ mở file trong filetable của tiến trình đang chạy). Nếu bị lỗi như tên file không tồn tại, không còn slot trống trong bảng mô tả file, truyền loại file sai thì báo lỗi.

2.4.3. int Close(OpenFileID id)

Ý nghĩa: Giúp User program đóng file đang mở bằng id của file đó. System call này có nhiệm vụ gọi filetable của tiến trình đang chạy để đóng file và giải phóng vùng nhớ của id của file đó trong bảng mô tả file.

Tham số truyền vào: id của file

Kết quả: Trả về giá trị là -1 nếu bị lỗi (đóng không thành công) và trả về 0 nếu đóng thành công.

2.4.4. int Read(char *buffer, int charcount, OpenFileID id)

Ý nghĩa: Giúp User program đọc từ file có id được xác định vào buffer với số byte yêu cầu là charcount.

Tham số truyền vào: buffer, số ký tự được yêu cầu đọc, file id.

Kết quả: Nếu đọc thành công thì trả về số byte thực sự đọc được, khi đọc mà có lỗi thì trả về -1, còn nếu đọc tới cuối file thì trả về -2.

2.4.5. int Write(char *buffer, int charcount, OpenFileID id)

Ý nghĩa: Giúp user program ghi vào file có id được xác định từ buffer với số byte charcount.

Tham số truyền vào: buffer, số ký tự cần ghi, file id.

Kết quả: Nếu ghi thành công thì trả về số byte thực sự ghi được, khi ghi mà có lỗi xảy ra thì trả về -1.

2.4.6. SpaceID Exec(char* name)

Ý nghĩa: Gọi thực thi một chương trình mới trong thread mới, trả về -1 nếu có lỗi và nếu thành công thì trả về SpaceID của chương trình người dùng vừa được tạo.

Tham số truyền vào: đường dẫn của chương trình thực thi (name), ở đồ án này scheduler.c chạy chương trình student.c ở cùng thư mục nên chỉ cần tên của chương trình là đủ.

Kết quả trả về:

2.4.7. int Join(SpaceID id)

Ý nghĩa: join sẽ đợi và block dựa trên tham số "SpaceID id".Chỉ có thể join vào tiến trình đã tạo ra nó bằng lệnh Exec, không thể join vào chính mình hoặc một tiến trình nào khác.

Tham số truyền vào: Space id

Kết quả trả về: trả về exitcode cho tiến trình nó đang block, trả về -1 nếu join bị lỗi

2.4.8. void Exit(int exitCode)

Ý nghĩa: trả về exit code cho tiến trình nó đã join.

Tham số truyền vào: exit code

Kết quả trả về: nếu exitcode là 0 tức là chương trình thành công, còn nếu là một mã khác thì có lỗi, tiến trình cha sẽ xử lý các exitcode này.

2.4.9. int CreateSemaphore(char* name, int semval)

Ý nghĩa: System call này để tạo semaphore mới, tạo cấu trúc dữ liệu để lưu 10 semaphore.

Tham số truyền vào: Truyền vào tên của semaphore

Kết quả: Trả về 0 nếu thành công, ngược lại thì trả về -1

2.4.10. int Wait(char* name)

Ý nghĩa: System call này thực hiện thao tác chờ

Tham số truyền vào: Truyền vào tên của semaphore

Kết quả: Trả về 0 nếu thành công, trả về -1 nếu lỗi (Người dùng sử dụng sai tên semaphore, semaphore đó chưa được tạo)

2.4.11.int Signal(char* name)

Ý nghĩa: System call này dùng để thực hiện thao tác giải phóng tiến trình đang chờ

Tham số truyền vào: Truyền vào tên của semaphore

Kết quả: Trả về 0 nếu thành công, trả về -1 nếu lỗi (Người dùng sử dụng sai tên semaphore, semaphore đó chưa được tạo)

2.5. Chương trình minh họa

Chúng em đã triển khai chương trình minh hoạ để giải quyết bài toán sau:

Trong một ký túc xá, có một vòi nước và n sinh viên, n sẽ input từ người dùng (n<5). Mỗi sinh viên sẽ cần lấy 10 lít nước từ vòi, nhưng ở mỗi lần lấy sinh viên chỉ có thể lấy tối đa 1 lít nước và phải trả vòi nước lại (sau khi trả vòi nước sinh viên có quyền yêu cầu sử dụng vòi nước ngay lập tức). Thời gian lấy một lít nước của mỗi sinh viên là ngẫu nhiên(có thể dùng vòng lặp for để mô phỏng thời gian đợi). Hãy viết một hệ thống đáp ứng được các yêu cầu trên.

Để giải quyết bài toán, chúng em đã tạo 2 file chương trình là scheduler.c và file student.c. File scheduler chứa chương trình chính, sẽ gọi các tiến trình phụ để chạy chương trình student. File student.c chứa chương trình mô phỏng việc lấy nước của các sinh viên. Ý nghĩa mỗi lệnh được chúng em bình luận trong các mã nguồn.

3. Hướng dẫn sử dụng chương trình

Bước 1: Ở thư mục /NachOS-4.0/code/build.linux gõ “make clean”, rồi “make depend” và cuối cùng là “make”.

Bước 2: Từ thư mục hiện tại cd vào thư mục test bằng lệnh “cd ../test”, sau đó chạy lệnh “make”.

Bước 3: Từ thư mục test, dùng lệnh “../build.linux/nachos -rs 1023 -x scheduler” để chạy chương trình demo scheduler.

Bước 4: Nhập số sinh viên n và enter, kết quả sẽ được xuất ra màn hình và file output.txt.

4. Những gì làm được và chưa làm được

Đã làm được 100% các yêu cầu như trong đề bài, nhưng khi chạy chương trình scheduler nhiều lần thì có một số lần bị lỗi.

5. Tài liệu tham khảo

Tài liệu trong moodle.

Lập trình Nachos HCMUS - YouTube

nguyenthanhchungfit/Nachos-Programing-HCMUS: This project includes: (github.com)