BỘ GIÁO DỤC VÀ ĐÀO TẠO

ĐẠI HỌC QUỐC GIA TP.HCM

TRƯỜNG ĐẠI HỌC KHOA HỌC TỰ NHIÊN

KHOA CÔNG NGHỆ THÔNG TIN

----------※※※----------

BÁO CÁO ĐỒ ÁN MÔN HỌC

HỆ ĐIỀU HÀNH

PROJECT 02
Multiprogramming

Giảng viên lý thuyết: Lê Giang Thanh

Giảng viên hướng dẫn thực hành: Lê Hà Minh

Trợ giảng: Nguyễn Thanh Quân

MỤC LỤC

I. Thành viên nhóm:

Thành viên	MSSV	Lớp
PHẠM PHÚ TOÀN	21127047	21CLC08
NGUYỄN TRẦN AN HÒA	21127183	21CLC08
TÔ KHÁNH LINH	21127638	21CLC08

II. Đánh giá mức độ hoàn thành:

Phần		Mô tả	Hoàn thành
1	1	Thêm Exceptions để chương trình hoàn thiện	**100%**
	2	Quản lý phân bổ bộ nhớ và giải phóng bộ nhớ, quản lý dữ liệu và đồng bộ hóa	**100%**
2	1	syscall Exec	**100%**
	2	syscall Join, Exit	**100%**
	3	syscall CreateSemaphore	**100%**
	4	syscall Wait, Signal	**100%**
	5	syscall Exec with argument	**0%**
3		shell program	**100%**

Mức độ hoàn thành: 95%

III. Lập trình

1. Phần 1: Exceptions, Phân bổ, giải phóng, quản lý bộ nhớ

1.1 . Thêm Exceptions

- NoException()
- PageFaultException()
- ReadOnlyException()
- BusErrorException()
- AddressErrorException()
- OverflowException()
- IllegalInstrException()
- NumExceptionTypes()
-

1.2 . Lớp AddrSpace

Dùng để cấp vùng nhớ và nạp chương trình người dùng, để hỗ trợ đa chương, ta cần có những thay đổi sau trong hàm AddrSpace::AddrSpace(char *filename).

• Vì các chương trình không còn được nạp lên các frame liên tục trên bộ nhớ, nên phải thay đổi physicalPage lấy địa chỉ frame còn trống từ Bitmap toàn cục gPhysPageBitMap.

• Nạp chương trình người dùng, bao gồm code, init data và readonly data, lên mainMemory dựa trên physicalPage của pageTable.

• Giải phóng bộ nhớ, trả lại frame đã đánh dấu cho gPhysPageBitMap khi chương trình người dùng kết thúc.

2. Phần 2: Các SYSCALL

2.1 . Syscall Exec

• Input:

▪ char* name: tên file của chương trình cần chạy.

• Output:

▪ SpaceID của chương trình người dùng vừa được tạo nếu thành công và -1 nếu bị lỗi

• Cài đặt:

▪ Tham số địa chỉ buffer lưu string cần ghi lấy từ thanh ghi số 4.

▪ Gọi kernel->pTab->ExecUpdate(name) để thực thi tiến trình.

▪ Chuyển dữ liệu của buffer từ User space vào Kernel space thông qua hàm User2System().

• Ghi vào thanh ghi số 2 là –1 nếu name = NULL hoặc khi quá trình mở một file bị lỗi.

• Ghi id của tiến trình mới vào thanh ghi số 2 nếu chạy thành công.

▪ Tăng thanh ghi PC

2.2 . Syscall Join

• Input:

▪ int id: process id của tiến trình muốn join vào tiến trình cha.

• Output:

▪ Trả về -1 nếu id không hợp lệ hoặc tiến trình đang join vào tiến trình không phải là tiến trình cha. Ngược lại trả về exit code của tiến trình.

• Cài đặt:

▪ Nhận tham số từ thanh ghi 4.

▪ Gọi kernel->pTab->JoinUpdate(id) để join vào tiến trình cha, chỉ khi tiến trình con này kết thúc thì tiến trình cha mới được tiếp tục thực thi.

▪ Ghi kết quả vào thanh ghi 2

▪ Tăng thanh ghi PC

2.2 . Syscall Exit

• Input:

▪ int exitCode: exit code trả về cho tiến trình mà nó đã chạy.

• Output:

▪ Không có tham số trả về.

• Cài đặt:

▪ Lấy tham số exitCode từ thanh ghi 4.

▪ Gọi kernel->pTab->ExitUpdate(exitCode) để giải phóng tiến trình và xin tiến trình cha để kết thúc.

▪ Gọi kernel->currentThread->FreeSpace() để thu hồi bộ nhớ.

▪ Tăng thanh ghi PC

2.3 . Syscall CreateSemaphore

• Input:

▪ char *name: tên semaphore cần tạo.

▪ int semVal: giá trị khởi tạo của semaphore.

• Output:

▪ Trả về chỉ số của semaphore, nếu semaphore đã tồn tại hoặc đã hết ô trống hoặc nếu name = NULL thì trả về -1

• Cài đặt:

▪ Đọc tham số name và semVal từ thanh ghi 4 và 5.

▪ Gọi kernel->semTab->Create(name, semVal) đã được cài đặt trong lớp Stable để tạo semaphore mới.

▪ Ghi kết quả trả về vào thanh ghi số 2

▪ Tăng thanh ghi PC

2.4 . Syscall Wait

• Input:

▪ char *name: tên semaphore chờ.

• Output:

▪ Trả về -1 nếu không thành công hoặc nếu name = NULL, ngược lại trả về 1.

• Cài đặt:

▪ Đọc tham số từ thanh ghi 4.

▪ Gọi kernel->semTab->Wait(name) được cài đặt ở lớp Stable để thực hiện chờ cho semaphore name.

▪ Ghi kết quả trả về vào thanh ghi số 2

▪ Tăng thanh ghi PC

2.4 . Syscall Signal(char* name)

• Input:

▪ char * name: tên của semaphore cần giải phóng.

• Output:

▪ Trả về -1 nếu không thành công hoặc nếu name = NULL, ngược lại trả về 1.

• Cài đặt:

▪ Đọc tham số từ thanh ghi 4.

▪ Gọi kernel->semTab->Signal(name) được cài đặt ở lớp Stable để thực hiện giải phóng cho semaphore name.

▪ Ghi kết quả trả về vào thanh ghi số 2.

▪ Tăng thanh ghi PC

2.5 . Syscall ExecV with argument

Đã làm: lấy được tham số và truyền vào từ hàm (argc, argv), tạo được thread và chạy được file argv[0]

Chưa làm được: chưa truyền tham số argv[1] - argv[argc-1] vào file thực thi argv[1] khi chạy.

3. Các lớp hỗ trợ

a) Lớp PCB(Process Control Block)

Dùng để quản lý tiến trình. Được cài đặt ở (userprog/pcb.h và pcb.cc)

	Tên	Ý nghĩa

Thuộc Tính	Sephamore* joinsem	Quản lý Join của process
	Sephamore* exitsem	Quản lý Exit của process
	Sephamore* mutex	Quản lý truy xuất số lượng tiến trình đang chờ
	int exitcode	
	int numwait	Số tiến trình đã join
	int processID	Lưu process tiến trình cha
	char filename[128]	Tên file
	Thread *thread	Quản lý tiến trình thực thi
Phương Thức	int Exec(char* name)	Thực thi tiến trình name
	void IncNumWait()	Tăng biến chờ
	void DecNumWait()	Giảm biến chờ
	void JoinWait()	Chuyển sang trạng thái chờ đến khi được giải phóng bởi JoinRelease
	void ExitWait()	Chuyển tiến trình sang trạng thái chờ đến khi được gọi ExitRelease()

	void JoinRelease()	Giải phóng tiến trình khỏi trạng thái chờ bởi JoinWait().
	void ExitRelease()	Giải phóng tiến trình khỏi trạng thái chờ bởi ExitWait().

b) Lớp PTable

Quản lý các tiến trình được chạy của chương trình người dùng. Với MAX_PROCESS=10. Được cài đặt ở(userprog/Ptable.h và Ptable.cc)

	Tên	Ý nghĩa
Thuộc Tính	PCB *pcb[MAX_PROCESS]	Mảng gồm MAX_PROCESS = 10 PCB cấp cho người dùng.
	int psize	Kích thước của bảng
	Sephamore *bmem	Đảm bảo sự đồng bộ của một vài tiến trình
Phương Thức	int ExecUpdate(char * name)	Xử lí cho syscall SC_Exec. Trường hợp tên không hợp lệ hoặc không còn chỗ trống thì trả về -1. Nếu hợp lệ thì thực thi tiến trình mới.
	void ExitUpdate(int ec)	Xử lý cho syscall SC_Exit, lấy process id của tiến trình hiện hành,gọi pcb[id]->JoinRelease() để tiến trình cha có thể tiếp tục, sau đó pcb[id]->ExitWait() để xin tiến trình cha cho phép

		dừng.
	int JoinUpdate(int id)	Cập nhập tiến trình tham gia
	int GetFreeSlot()	Tìm vị trí trống
	void Remove(int id)	Xóa tiến trình

c) Lớp Stable

Dùng để quản lý các Semaphore mà người dùng sử dụng, cho phép người dùng tạo tối đa MAX_SEMAPHORE = 10. Được cài đặt ở (userprog/stable.h và stable.cc).

	Tên	Ý nghĩa
Thuộc Tính	Sem* sem[MAX_SEMAPHORE]	Mảng Semaphore được cấp cho người dùng.
Phương Thức	int Create(char* name, int init)	Cấp phát một semaphore tên name với giá trị khởi tạo là value. Nếu name đã tồn tại hoặc hết ô trống thì sẽ trả về -1.
	int Wait(char* name)	Semaphore name thực hiện chờ, trả về -1 nếu name không hợp lệ ngược lại trả về 1.
	int Signal(char* name)	Giải phóng semaphore name, trả về -1 nếu name không hợp lệ .
	int FindFreeSlot()	Tìm slot trống trên mảng semTable, trả

		về -1 nếu không tìm thấy

d) Lớp Sem

Dùng để quản lý một Semaphore. Được cài đặt ở (userprog/sem.h và sem.cc).

	Tên	Ý nghĩa
Thuộc Tính	Semaphore* sem	Semaphore mà Sem quản lý.
	char name[50]	Tên của semaphore.
Phương Thức	void Wait()	Thực hiện sem->P() để chờ
	void Signal()	Thực hiện sem->V() để giải phóng.
	char* GetName()	Lấy tên của Semaphore.

IV. Bảng phân công công việc:

Thành viên	MSSV	Công việc
PHẠM PHÚ TOÀN	21127047	Exec, Join, Exit
NGUYỄN TRẦN AN HÒA	21127183	Wait, Signal
TÔ KHÁNH LINH	21127648	CreateSemaphore, báo cáo

V. Các nguồn tài liệu tham khảo:

- Tài liệu của thầy Nguyễn Thanh Quân
- Github lê duy thức
- Youtube: Nguyễn Thành Chung