HỆ ĐIỀU HÀNH

BÁO CÁO ĐỒ ÁN 3

Nhóm :

1512205 Nguyễn Văn Quang Huy

1512262 Võ Anh Khoa

Khoa Công nghệ Thông tin

Đại học Khoa học Tự nhiên TP HCM

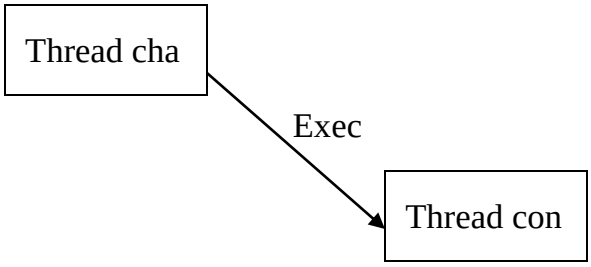
Tháng 12/2017

MỤC LỤC

Thiết kế hệ thống quản lý tiến trình

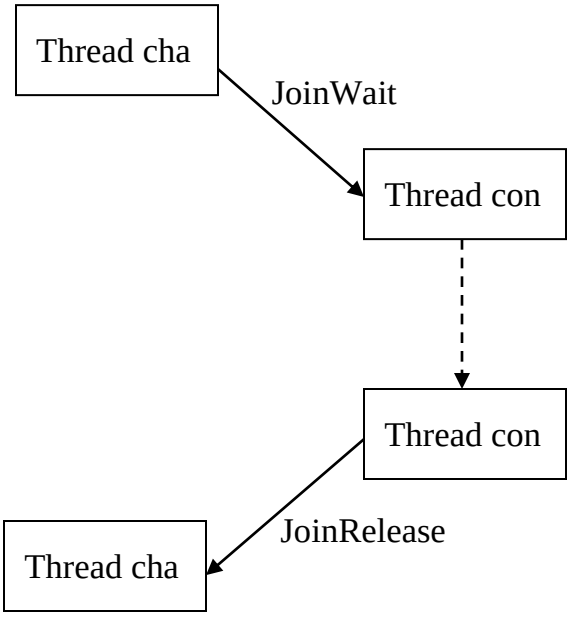
Mô hình quản lý tiến trình:

Execute:



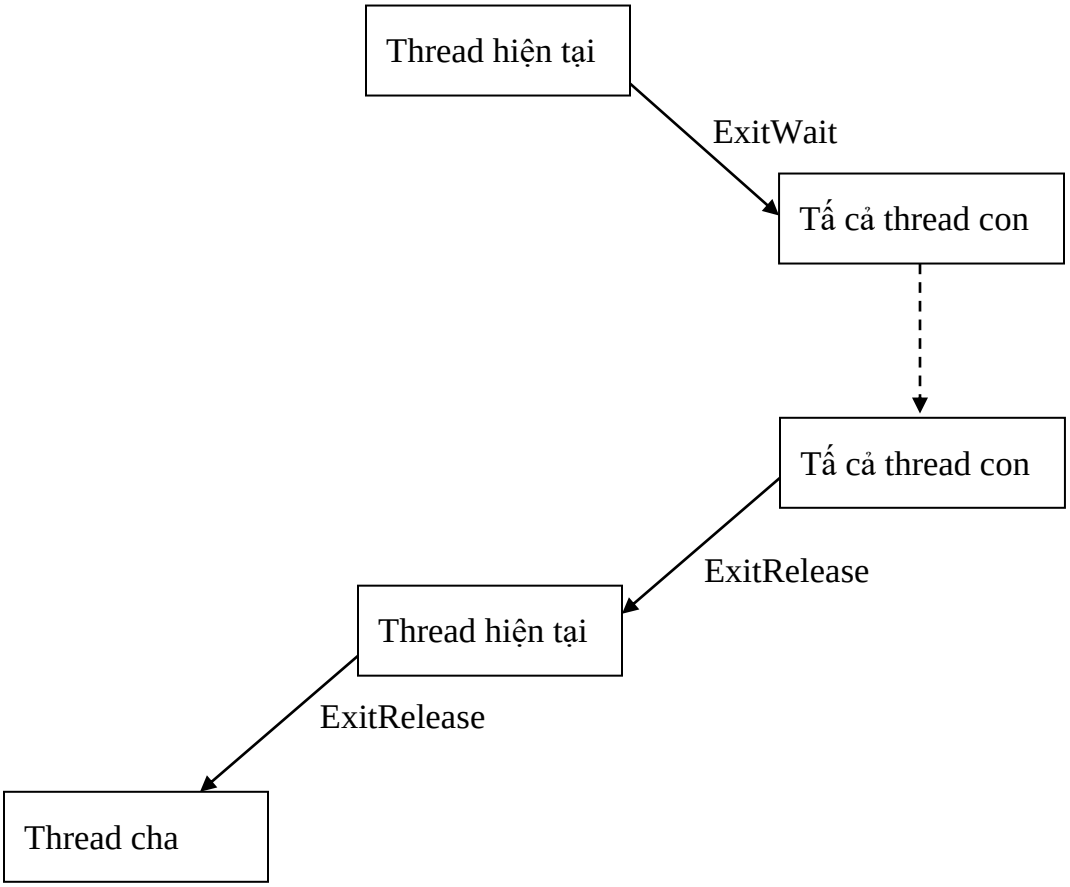
Thread cha sẽ gọi thực thi thread con bằng lệnh Exec và tăng số lượng thread con đang chạy của nó lên 1.

Join:



Thread cha tiến hành chờ thread con chạy xong bằng lệnh JoinWait. Thread con sau khi chạy xong sẽ đánh thức thread cha bằng lệnh JoinRelease.

Exit :



Khi thread hiện tại muốn exit thì sẽ tiến hành chờ các thread con đang chạy của nó kết thức bằng lệnh ExitWait. Sau khi các thread con chạy xong sẽ gọi lệnh ExitRelease để đánh thức thread hiện tại, tiến hành kết thúc. Thread hiện tại trước khi kết thúc sẽ gọi ExitRelease để gửi tín hiệu kết thúc cho thread cha biết.

Để quản lý các tiến trình cần có 2 lớp là lớp PCB và lớp PTable.

Lớp PCB dùng để quản lý 1 tiến trình. Khai báo của lớp PCB lưu trong file pcb.h và cài đặt của lớp PCB lưu trong file pcb.cc. Cấu trúc lớp PCB như sau:

	Tên	Kiểu dữ liệu/Giá trị	Ý nghĩa
Hằng số	PCB_MAXNAMESIZE	50	Kích thước lớn nhất của chuỗi chứa tên thread

Thuộc tính	_filename[PCB_MAXNAMESIZE]	char	Chuỗi chứa tên thread
	_thread	Thread*	Con trỏ trỏ đến thread mà đối tượng lớp PCB quản lý
	_parentID	int	Là id của thread cha
	_processID	int	Là id của thread mà đối tượng lớp PCB quản lý
	_joinsem	Semaphore*	Semaphore quản lý việc join thread
	_exitsem	Semaphore*	Semaphore quản lý việc exit thread
	_mutex	Semaphore*	Semaphore quản lý việc nạp thread
	_joinid	int	Là id của thread con đang join
	_joinexitcode	int	Là exit code của thread con được join trả về
	_numwait	int	Là số thread con được execute
	_isExit	int	Đánh dấu thread đã chuyển qua trạng thái exit chưa
Phương thức	PCB()		Hàm khởi tạo mặc định của lớp PCB
	~PCB()		Hàm huỷ của lớp PCB
	Exec(char*, int, int)	int	Thực thi 1 thread với tên được truyền vào kèm id của thread đó và thread cha. Kết quả trả ra 0 nếu thành công và -1 nếu thất bại

	JoinWait(int)	void	Join 1 thread con với id được truyền vào. Thread hiện tại sẽ chờ thread con
	JoinRelease(int, int)	void	Giải phóng thread đang chờ nếu đúng id thread con được chờ kèm với exit code được truyền vào
	ExitWait()	void	Thread hiện tại chờ tất cả thread con exit
	ExitRelease()	void	Giải phóng thread cha nếu thread cha đang trong trạng thái exit và chờ các thread con exit
	IncNumWait()	void	Tăng số thread con
	DecNumWait()	void	Giảm số thread con
	GetParentID()	int	Trả ra id của thread cha
	GetProcessID()	int	Trả ra id của thread mà đối tượng lớp PCB quản lý
	GetJoinID()	int	Trả ra id của thread con đang được join
	GetJoinExitCode()	int	Trả ra exit code của thread con được join đã exit
	GetFileName()	char*	Trả ra chuỗi chứa tên thread
	GetThread()	Thread*	Trả ra thread mà đối tượng lớp PCB quản lý

Lớp PTable dùng để quản lý hệ thống toàn bộ tất cả các tiến trình trong hệ điều hành Nachos. Khai báo của lớp PTable lưu trong file ptable.h và cài đặt của lớp PTable lưu trong file ptable.cc. Cấu trúc lớp PTable như sau:

	Tên	Kiểu dữ liệu/Giá trị	Ý nghĩa
Hằng số	MAX_PROCESS	10	Số lượng thread tối đa mà đối tượng PTable quản lý
Thuộc tính	_bm	Bitmap*	Quản lý các đối tượng đã dùng và còn trống chưa dùng trong đối tượng lớp PTable
	_bmsem	Semaphore*	Semaphore quản lý việc nạp thread
	_pcb[MAX_PROCESS]	PCB*	Mảng con trỏ trỏ đến các đối tượng lớp PCB mà đối tượng lớp PTable quản lý
Phương thức	PTable()		Hàm khởi tạo mặc định của lớp PTable
	~PTable()		Hàm huỷ của lớp PTable
	ExecUpdate(char*)	int	Hàm tạo thread mới với đường dẫn được truyền vào. Kết quả trả ra 0 nếu thành công và -1 nếu thất bại
	ExitUpdate(int)	int	Hàm exit thread hiện tại với exit code được truyền vào. Kết quả trả ra 0 nếu thành công và -1 nếu thất bại
	JoinUpdate(int)	int	Hàm join thread hiện tại với thread con và chờ thread con chạy xong. Id của thread con được truyền vào. Kết quả trả ra 0 nếu thành công và -1 nếu thất bại

Thiết kế hệ thống quản lý semaphore

Để quản lý các semaphore cần có 2 lớp là lớp Sem và lớp STable.

Lớp Sem dùng để quản lý 1 semaphore. Khai báo của lớp Sem lưu trong file sem.h và cài đặt của lớp Sem lưu trong file sem.cc. Cấu trúc lớp Sem như sau:

	Tên	Kiểu dữ liệu/Giá trị	Ý nghĩa
Hằng số	SEM_MAXNAMESIZE	50	Kích thước lớn nhất của chuỗi chứa tên semaphore
Thuộc tính	_name[SEM_MAXNAMESIZE]	char	Chuỗi chứa tên semaphore
	_sem	Semaphore*	Con trỏ trỏ đến 1 đối tượng semaphore mà lớp Sem quản lý
Phương thức	Sem()		Hàm khởi tạo mặc định của lớp Sem
	~Sem()		Hàm huỷ của lớp Sem
	Create(char*, int)	int	Hàm tạo semaphore mà đối tượng lớp Sem quản lý. Dữ liệu truyền vào gồm tên và giá trị của semaphore. Kết quả trả ra 0 nếu tạo thành công và -1 nếu thất bại
	Delete()	void	Hàm xoá semaphore mà đối tượng lớp Sem quản lý
	Wait()	void	Giảm giá trị của semaphore. Nếu giá trị đó bằng 0 và không thể giảm được thì cho thread hiện tại chờ

	Signal()	void	Tăng giá trị của semaphore và đánh thức thread đang chờ nếu có
	GetName()	char*	Lấy ra chuỗi chứa tên semaphore

Lớp STable dùng để quản lý hệ thống toàn bộ tất cả các semaphore trong hệ điều hành Nachos. Khai báo của lớp STable lưu trong file stable.h và cài đặt của lớp STable lưu trong file stable.cc. Cấu trúc lớp STable như sau:

	Tên	Kiểu dữ liệu/Giá trị	Ý nghĩa
Hằng số	MAX_SEMAPHORE	10	Số lượng semaphore tối đa có thể tạo được trong đối tượng lớp STable
Thuộc tính	_bm	Bitmap*	Quản lý các đối tượng đã dùng và còn trống chưa dùng trong đối tượng lớp STable
	_semTab[MAX_SEMAPHORE]	Sem*	Mảng con trỏ trỏ đến các đối tượng lớp Sem mà đối tượng lớp STable quản lý
Phương thức	STable()		Hàm khởi tạo mặc định của lớp STable
	~STable()		Hàm huỷ của lớp STable
	Create(char*, int)	int	Hàm tạo một đối tượng lớp Sem với tên và giá trị semaphore được truyền vào. Kết quả trả ra 0 nếu tạo thành công và -1 nếu thất bại
	Wait(char*)	int	Tiến hành wait semaphore có tên được truyền vào. Kết quả trả ra 0 nếu tạo thành công và -1 nếu thất bại
	Signal(char*)	int	Tiến hành signal semaphore có tên được truyền vào. Kết quả trả ra 0 nếu

			tạo thành công và -1 nếu thất bại

Bước 1

Thực hiện tất cả các bước ở đồ án 1 và đồ án 2.

Bước 2

Thêm các lớp PCB, STable, Sem, STable vào hệ điều hành Nachos:

- Di chuyển 8 file pcb.cc, pcb.h, ptable.cc, ptable.h, sem.cc, sem.h, stable.cc và stable.h vào thư mục nachos/nachos-3.4/code/threads

- Chỉnh sửa lại file Makefile.common trong thư mục nachos/nachos-3.4/code:

USERPROG_H: add ../threads/sem.h

USERPROG_H: add ../threads/stable.h

USERPROG_H: add ../threads/pcb.h

USERPROG_H: add ../threads/ptable.h

USERPROG_C: add ../threads/sem.cc

USERPROG_C: add ../threads/stable.cc

USERPROG_C: add ../threads/pcb.cc

USERPROG_C: add ../threads/ptable.cc

USERPROG_O: add sem.o

USERPROG_O: add stable.o

USERPROG_O: add pcb.o

USERPROG_O: add ptable.o

Bước 3

Trong file system.h trong thư mục nachos/nachos-3.4/code/threads ta chỉnh sửa như sau:

- Thêm file synch.h, bitmap.h, stable.h, ptable.h vào Nachos để có thể dùng được lớp Semaphore, Bitmap, STable, PTable

/*include synch.h*/

/*include bitmap.h*/

/*include stable.h*/

/*include ptable.h*/

- Declare biến con trỏ toàn cục addrLock kiểu Semaphore, gPhysPageBitMap kiểu Bitmap, semTab kiểu STable, pTab kiểu PTable.

/*declare addrLock*/

/*declare gPhysPageBitMap*/

/*declare semTab*/

/*declare pTab*/

Trong file system.cc trong thư mục nachos/nachos-3.4/code/threads ta chỉnh sửa như sau:

- Define con trỏ addrLock, gPhysPageBitMap, semTab, pTab:

/*define addrLock*/

/*define gPhysPageBitMap*/

/*define semTab*/

/*define pTab*/

- Tạo đối tượng addrLock, gPhysPageBitMap, semTab, pTab:

/*construction - addrLock*/

/*construction - gPhysPageBitMap*/

/*construction - semTab*/

/*construction - pTab*/

- Huỷ đối tượng addrLock, gPhysPageBitMap, semTab, pTab:

/*destruction - addrLock*/

/*destruction - gPhysPageBitMap*/

/*destruction - semTab*/

/*destruction - pTab*/

Bước 4

Chỉnh các file trong thư mục nachos/nachos-3.4/code/userprog như sau:

- File addrspace.cc: Chỉnh sửa hàm khởi tạo và hàm huỷ để có thể chạy được đa thread:

/*Edit constructor for multithreading*/

/*Edit destructor for multithreading*/

- File progtest.cc: Chỉnh sửa hàm StarProcess để đưa thread chạy đầu tiên vào pTab làm thread chính của hệ điều hành Nachos:

/*Edit to move main thread into pTab*/

- File syscall.h: Định nghĩa thêm kiểu dữ liệu mới SpaceID, các syscall mới, các hàm gọi các syscall này:

/*add syscall codes*/

/*add a new type - SpaceID*/

/*add syscall function declarations*/

- File exception.cc: Cài đặt quá trình thực thi các syscall mới bên trong hệ thống:

/*load file name from user to kernel*/

/*execute process*/

/*join process*/

/*exit process*/

/*use SEM_MAXNAMESIZE to get name*/

/*create semaphore*/

/*use SEM_MAXNAMESIZE to get name*/

/*wait semaphore*/

/*use SEM_MAXNAMESIZE to get name*/

/*signal semaphore*/

/*syscall exception*/

Ghi chú: Để tránh trùng lặp tên hàm đã có sẵn trong mã nguồn, tên các hàm gọi các syscall được thay đổi để đảm bảo chương trình biên dịch được: ExecProc, JoinProc, ExitProc, CreateSemaphore, Wait, Signal

Bước 5

Trong file start.c và start.s trong thư mục nachos/nachos-3.4/code/test ta chỉnh sửa như sau:

- Cài đặt thêm vào các hàm gọi các syscall mới ở phía người dùng để phục vụ cho việc thực thi thread, join thread, thoát thread, tạo semaphore, wait semaphore và signal semaphore.

/*add syscall function definitions*/

Bước 6

Chỉnh sửa 1 số file khác để đảm bảo hệ điều hành có thể chạy được thêm thread mới:

- File directory.h trong thư mục filesys: Chỉnh sửa hằng số FileNameMaxLen lên 50 để có thể nhận được tên đường dẫn file dài hơn

- File machine.h trong thư mục machine: Chỉnh sửa hằng số NumPhysPages lên 128 để đảm bảo có thể nạp được nhiều thread hơn vì hệ điều hành Nachos lúc đầu chỉ có thể chạy đơn chương

Bước 7

Viết và chạy thử các chương trình sử dụng các syscall mới được thêm vào ở trên:

- Viết các file mã nguồn C của các chương trình và để vào trong thư mục nachos/nachos-3.4/code/test:

/*main program*/

/*sinhvien program*/

/*voinuoc program*/

- Chỉnh sửa file Makefile trong thư mục nachos/nachos-3.4/code/test để đưa vào biên dịch mã nguồn C của các chương trình:

#main

#sinhvien

#voinuoc.

Bước 8

Biên dịch lại từ đầu toàn bộ tất cả: Mở terminal với thư mục hiện hành là nachos/nachos-3.4/code nhập lệnh: gmake all

Bước 9

Chạy thử các chương trình: Mở terminal với thư mục hiện hành là nachos/nachos-3.4/code nhập lệnh:

./userprog/nachos -x test/main

Tổng kết

- Những vấn đề đã làm được:

    + Code và chạy được thành công cả 6 syscall mới

    + Code và chạy được thành công chương trình “Thống kê sử dụng máy nóng lạnh”

- Những vấn đề chưa làm được: Không có