

TRƯỜNG ĐẠI HỌC KHOA HỌC TỰ NHIÊN
KHOA CÔNG NGHỆ THÔNG TIN

-----o0o-----

BÁO CÁO ĐỒ ÁN HỆ ĐIỀU HÀNH

Project 2: Multiprogramming



Giáo viên hướng dẫn: Nguyễn Thanh Quân

Lê Giang Thanh

Lê Hà Minh

Thành phố Hồ Chí Minh, tháng 3 năm 2023

Mục lục

PHẦN 1: THÔNG TIN CHUNG	3
1.1: Thông tin chung	3
1.2: Thông tin nhóm	3
PHẦN 2: ĐÁNH GIÁ MỨC ĐỘ HOÀN THÀNH	4
2.1: Bảng đánh giá mức độ hoàn thành	4
PHẦN 3: THÔNG TIN ĐỒ ÁN	5
3.1: Đa chương	5
3.2: Cài đặt cho đa chương, lập lịch và đồng bộ hoá	5
3.2.1: Class PCB (Process Control Block)	5
3.2.2: Class PTable	6
3.2.3: Class Sem	8
3.2.4: Class STable	8
3.3: Cài đặt Syscall	9
3.3.1: Syscall Exec	9
3.3.2: Syscall Join	9
3.3.3: Syscall Exit	10
3.3.4: Syscall CreateSemaphore	10
3.3.5: Syscall Wait	10
3.3.6: Syscall Signal	11
3.4: Test	11

PHẦN 1: THÔNG TIN CHUNG

1. Thông tin chung:

Tên giảng viên: Nguyễn Thanh Quân, Lê Giang Thanh, Lê Hà Minh

Tên đề án: System calls & File - Network Operations

Thời gian thực hiện: Từ 6/4/2023 tới 5/5/2023

2. Thông tin nhóm:

MSSV	Họ & tên	Email
21127443	Trần Ngọc Trường Thịnh	tntthinh21@clc.fitus.edu.vn
21127712	Lê Quang Trường	lqtruong21@clc.fitus.edu.vn
21127329	Châu Tuấn Kiệt	ctkiet21@clc.fitus.edu.vn

PHẦN 2: ĐÁNH GIÁ MỨC ĐỘ HOÀN THÀNH

1. Bảng đánh giá mức độ hoàn thành:

Part		Describe	Mức độ hoàn thành
1	1	Multiprogramming	100%
2	1	Class PCB (Process Control Block)	100%
	2	Class PTable	100%
	3	Class Sem	100%
	4	Class STable	100%
3	1	Syscall Exec	100%
	2	Syscall Join	100%
	3	Syscall Exit	100%
	4	Syscall CreateSemaphore	100%
	5	Syscall Wait	100%
	6	Syscall Signal	100%
4	1	Test	100%

PHẦN 3: THÔNG TIN ĐỒ ÁN

1. Đa chương:

NachOS là môi trường lập trình chỉ hỗ trợ đơn chương, cụ thể là khi tạo ra một tiến trình thì tiến trình đó sẽ được đặt trên một virtual page và physical page giống nhau. Để hệ điều hành có thể hỗ trợ đa chương, ta thực hiện sửa đổi trong file `addrspace.cc` (hàm `AddrSpace::Load(char *filename)`), các chương trình sẽ không còn được nạp lên các frame liên tục mà sẽ sử dụng bitmap để đánh dấu các frame đã sử dụng và lấy địa chỉ các page còn trống.

Việc nạp chương trình người dùng bao gồm code và data segment lên bộ nhớ chính phải dựa trên địa chỉ `physicalPage` của `pageTable`.

Thay đổi việc kết thúc chương trình người dùng thay vì chỉ xoá `pageTable` thì giờ đây sẽ phải xoá những dữ liệu tương ứng trong `gPhysPageBitMap` trước rồi mới đến `pageTable`.

2. Cài đặt các lớp phục vụ cho đa chương, lập lịch và đồng bộ hoá:

2.1 Class PCB (Process Control Block):

Cài đặt trong `thread/pcb.cc` và `thread/pcb.h`

	Tên	Chức năng
Thuộc tính	<code>Semaphore* joinsem</code>	Quản lý quá trình Join của process.
	<code>Semaphore* exitsem</code>	Quản lý quá trình Exit của process.
	<code>Semaphore* multex</code>	Quản lý truy xuất số lượng tiến trình chờ đang chờ.
	<code>Thread* thread</code>	Quản lý thực thi tiến trình.
	<code>int parentID</code>	Lưu process id của tiến trình cha.
	<code>OpenFile** fileTable</code>	Quản lý các file đang được mở trong tiến trình.

HỆ ĐIỀU HÀNH – PROJECT 2: MULTIPROGRAMMING

Phương thức	void JoinWait()	Chuyển tiến trình sang trạng thái chờ cho đến khi được giải phóng bởi JoinRelease().
	void JoinRelease()	Giải phóng tiến trình khỏi trạng thái chờ bởi JoinWait().
	void ExitWait()	Chuyển tiến trình sang trạng thái chờ đến khi được gọi ExitRelease().
	void ExitRelease()	Giải phóng tiến trình khỏi trạng thái chờ bởi ExitWait().
	int Exec(char *filename, pid)	Thực thi tiến trình tên là filename với process id là pid
	OpenFileID Open(char*name, int type)	Thực thi tiến trình tên là filename với process id là pid.
	int Close(OpenFileID fid)	Mở một file với type 0: đọc và ghi hoặc type 1: chỉ đọc.
	int Read(char* buffer,int charcount , OpenFileID id)	Đóng một file có file id là fid.
	int Write(char* buffer,int charcount, OpenFileID id)	Đọc chuỗi buffer có charcount kí tự từ file có file id là id.
	int Seek(int position, OpenFileID id)	Ghi chuỗi buffer có charcount kí tự từ file có file id là id.

2.2 Class PTable:

- Được cài đặt trong file thread/ptable.h và thread/ptable.cc
- Chức năng: quản lí chương trình người dùng
- Hàm StartProcess(int id) sau khi Fork, lúc scheduler cho phép thực thi, sẽ tạo một AddrSpace để nạp chương trình người dùng và bắt đầu thực thi tiến trình này.

	Tên	Chức năng
Thuộc tính	Bitmap *bm	Đánh dấu các vị trí đã được sử dụng.

HỆ ĐIỀU HÀNH – PROJECT 2: MULTIPROGRAMMING

	PCB *pcb[MAX_PROCESS]	Mảng gồm MAX_PROCESS = 10 PCB cấp cho người dùng.
Phương thức	int JoinUpdate(int id)	Xử lý cho syscall SC_Join, kiểm tra pid hợp lệ, gọi pcb[id]->JoinWait() để tiến trình cha đợi đến khi tiến trình con này kết thúc, sau đó gọi pcb[id]->ExitRelease() để cho phép tiến trình con này được kết thúc.
	void ExitUpdate(int exitcode)	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.
	void ExecUpdate(char *name)	Xử lý cho syscall SC_Exec. . Thực thi tiến trình mới bằng cách gọi thread->Fork((VoidFunctionPtr) &StartProcess, (void*)pid). Trường hợp tên không hợp lệ hoặc không còn chỗ trống thì trả về -1
	OpenFileID Open(int pid, char*name, int type)	Mở một file của tiến trình có id là pid
	int Close(int pid, OpenFileID fid)	Đóng một file có id fid của tiến trình có id là pid.
	int Read(int pid, char* buffer,int charcount, OpenFileID fid)	Đọc chuỗi buffer có charcount kí tự từ file có file id là fid của tiến trình có id là pid.

HỆ ĐIỀU HÀNH – PROJECT 2: MULTIPROGRAMMING

	int Write(int pid, char* buffer, int charcount, OpenFileID fid)	Ghi chuỗi buffer có charcount kí tự từ file có file id là fid trong tiến trình có id là pid.
	int Seek(int pid, int position, OpenFileID fid)	Di chuyển con trỏ đến vị trí position của file có id là fid trong tiến trình có id là pid.

2.3 Class Sem:

- Được cài đặt trong file thread/stable.h và thread/stable.cc.
- Chức năng: quản lý một Semaphore.

	Tên	Chức năng
Thuộc tính	char* name	Tên của semaphore.
	Semaphore *sem	Semaphore mà Sem quản lý.
Phương thức	wait()	Thực hiện sem->P() để chờ.
	signal()	Thực hiện sem->V() để giải phóng.
	getName()	Lấy tên của Semaphore.

2.4 Class STable:

- Được cài đặt trong file thread/stable.h và thread/stable.cc chung với lớp Sem.
- Chức năng: quản lý các Semaphore mà người dùng sử dụng, cho phép người dùng tạo tối đa 10 Semaphore (MAX_SEMAPHORE = 10)

	Tên	Chức năng
Thuộc tính	Bitmap *mSemBitMap	Bitmap quản lý mảng cho biết Semaphore nào đã được dùng.
	Sem *semTable[MAX_SEMAPHORE]	Mảng Semaphore được cấp cho người dùng.
Phương thức	int Create(char *name, int value)	Cấp phát một semaphore tên name với giá trị khởi tạo là

		value. Trả về -1 nếu name đã tồn tại hoặc hết ô trống.
	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.

3. Cài đặt Syscall:

(Các syscall được cài đặt ở project 1 sẽ không nằm trong báo cáo này)

3.1 Syscall Exec:

SpaceID Exec(char* name)

- Input:
 - name: tên của file cần chạy.
- Output:
 - -1 nếu bị lỗi và thành công thì trả về SpaceID của chương trình người dùng vừa được tạo nếu thành công.
- Cài đặt:
 - Lấy các tham số địa chỉ buffer lưu string cần ghi từ thanh ghi số 4.
 - Chuyển dữ liệu của buffer từ User space vào Kernel space thông qua hàm User2System().
 - Gọi hàm void SysExec(cchar *name):
 - ◆ Ghi -1 vào thanh ghi số 2 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.
 - Giải phóng bộ nhớ của name
 - Tăng thanh ghi PC

3.2 Syscall Join:

int Join(SpaceID id)

- Input:

- id: process id của tiến trình muốn join vào tiến trình cha.
- Output:
 - Trả về exit code của tiến trình. Nếu id không hợp lệ hoặc tiến trình đang cố join vào tiến trình không phải tiến trình cha thì trả về -1.
- Cài đặt:
 - Nhận tham số từ register.
 - 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 register 2
 - Tăng thanh ghi PC

3.3 Syscall Exit:

void Exit(int exitCode)

- Input:
 - exitCode: exit code trả về cho tiến trình mà nó đã chạy.
- Output:
 - Không có.
- Cài đặt:
 - Lấy tham số exitCode từ register 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.
 - Gọi kernel->currentThread->Finish() để báo scheduler kết thúc tiến trình.
 - Tăng thanh ghi PC.

3.4 Syscall CreateSemaphore:

int CreateSemaphore(char* name, int semVal)

- Input:
 - name: tên của semaphore cần tạo.
 - semVal: giá trị khởi tạo của semaphore.
- Output:
 - Trả về chỉ số của semaphore, nếu semaphore đã tồn tại hoặc đã hết ô trống thì trả về -1.
- Cài đặt:
 - Đọc tham số name và semVal từ register 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.

3.5 Syscall Wait:

int Wait(char* name)

- Input:
 - name: tên của semaphore cần chờ.
- Output:
 - Trả về -1 nếu không thành công, ngược lại trả về 1.
- Cài đặt:
 - Đọc tham số từ register 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.

3.6 Syscall Signal:

int Signal(char* name)

- Input:
 - name: tên của semaphore cần giải phóng.
- Output:
 - Trả về -1 nếu không thành công, ngược lại trả về 1.
- Cài đặt:
 - Đọc tham số từ register 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.

4. Tests:

shell.c: kiểm tra syscall Exec và Join.

```
#include "syscall.h"

int main(){
    Spaceld newProc1;
    Spaceld newProc2;

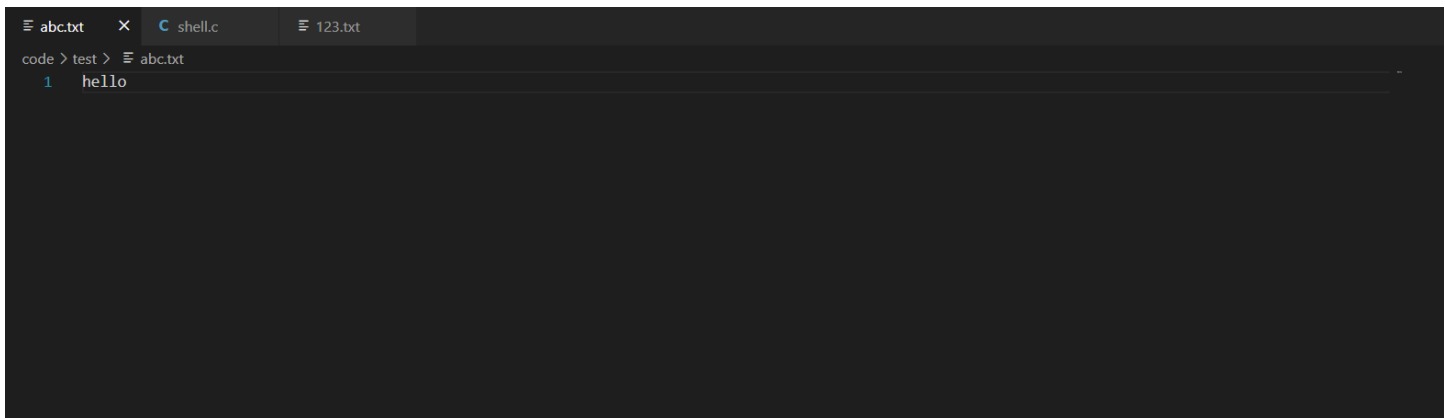
    newProc2 = Exec("cat");
    newProc1 = Exec("copy");

    Join(newProc2);
    Join(newProc1);
}
```

Copy: sao chép dữ liệu từ file abc.txt sang file 123.txt.

Cat: đọc dữ liệu từ file abc.txt rồi xuất ra màn hình.

● File abc.txt



A screenshot of a code editor with three tabs: 'abc.txt', 'C shell.c', and '123.txt'. The 'abc.txt' tab is active, showing a single line of code: '1 hello'. The editor's interface includes a file explorer on the left and a terminal at the bottom.

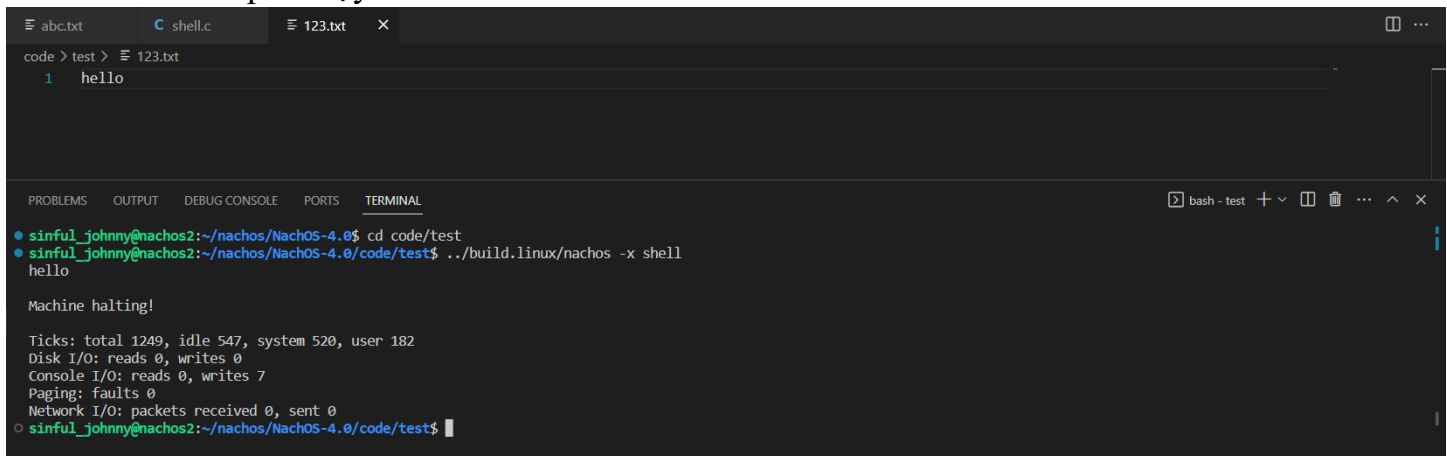
● Hình ảnh file 123.txt và console lúc nhập lệnh



A screenshot of a code editor with three tabs: 'abc.txt', 'C shell.c', and '123.txt'. The '123.txt' tab is active, showing a single line of code: '1'. Below the editor, the terminal window shows the following commands and output:

```
sinful_johnny@nachos2:~/nachos/NachOS-4.0$ cd code/test
sinful_johnny@nachos2:~/nachos/NachOS-4.0/code/test$ ../build.linux/nachos -x shell
```

● Kết quả chạy:



A screenshot of a code editor with three tabs: 'abc.txt', 'C shell.c', and '123.txt'. The '123.txt' tab is active, showing a single line of code: '1 hello'. Below the editor, the terminal window shows the following commands and output:

```
sinful_johnny@nachos2:~/nachos/NachOS-4.0$ cd code/test
sinful_johnny@nachos2:~/nachos/NachOS-4.0/code/test$ ../build.linux/nachos -x shell
hello

Machine halting!

Ticks: total 1249, idle 547, system 520, user 182
Disk I/O: reads 0, writes 0
Console I/O: reads 0, writes 7
Paging: faults 0
Network I/O: packets received 0, sent 0
sinful_johnny@nachos2:~/nachos/NachOS-4.0/code/test$
```