

ĐẠI HỌC QUỐC GIA THÀNH PHỐ HỒ CHÍ MINH
TRƯỜNG ĐẠI HỌC KHOA HỌC TỰ NHIÊN
KHOA CÔNG NGHỆ THÔNG TIN



20127662 - NGUYỄN ĐÌNH VĂN

20127469 - PHẠM MINH ĐỨC

20127061 - LƯU MINH PHÁT

BÁO CÁO ĐỒ ÁN

HỆ ĐIỀU HÀNH

Thành phố Hồ Chí Minh – 2022

ĐẠI HỌC QUỐC GIA THÀNH PHỐ HỒ CHÍ MINH
TRƯỜNG ĐẠI HỌC KHOA HỌC TỰ NHIÊN
KHOA CÔNG NGHỆ THÔNG TIN



20127662 - NGUYỄN ĐÌNH VĂN

20127469 - PHẠM MINH ĐỨC

20127061 - LƯU MINH PHÁT

BÁO CÁO ĐỒ ÁN 2

|Đề tài|

Đa chương

|Giáo viên hướng dẫn|

Gv.Lê Viết Long

Gv.Phạm Tuấn Sơn

Hệ Điều Hành

Thành phố Hồ Chí Minh - 2022

Mục lục

1. THÔNG TIN THÀNH VIÊN:	2
1.1. Thông tin	2
1.2. Mức độ hoàn thành	3
2. CÀI ĐẶT CHUNG	4
2.1. Cài đặt trước	4
2.2. Các giá trị thanh ghi	4
2.3. Chuẩn bị cho đa chương	4
3. Cài đặt đa chương	6
3.1. Cài đặt chi tiết:	6
4. DEMO CHƯƠNG TRÌNH:	8
5. MÔI TRƯỜNG LẬP TRÌNH:	9
6. BẢNG PHÂN CÔNG CÔNG VIỆC CỦA THÀNH VIÊN:	9
7. TÀI LIỆU THAM KHẢO:	9

1. THÔNG TIN THÀNH VIÊN:

1.1. Thông tin

Họ và tên	MSSV	Email
Lưu Minh Phát	20127061	20127061@student.hcmus.edu.vn
Nguyễn Đình Văn	20127662	20127662@student.hcmus.edu.vn
Phạm Minh Đức	20127469	20127469@student.hcmus.edu.vn

1.2. Mức độ hoàn thành

	CHỨC NĂNG	MỨC ĐỘ HOÀN THÀNH
1.Cài đặt system call SpaceID Exec(char* name)	a. Tạo ra một không gian địa chỉ mới	100%
	b. Load chương trình vào khoảng bộ nhớ mới được cấp phát	100%
	c. Tạo thread mới (bằng phương thức Thread::Fork()) để thực thi chương trình.	100%
2. Cài đặt đa tiến trình.	a. Giải quyết vấn đề cấp phát các frames bộ nhớ vật lý	100%
	b. Xử lý giải phóng bộ nhớ khi user program kết thúc	100%
	c. Thay đổi đoạn lệnh nạp user program lên bộ nhớ.	100%

2. CÀI ĐẶT CHUNG

2.1. Cài đặt trước

- Tạo 1 **AddressSpace** , Khung trang trong NachOS tên là PhysPages – trang vật lý ,sử dụng class BitMap để đánh dấu các khung trang còn trống.

2.2. Các giá trị thanh ghi

- R2: Lưu mã syscall đồng thời lưu kết quả trả về của mỗi syscall nếu có.
- R4: Lưu tham số thứ nhất.
- R5: Lưu tham số thứ hai.
- R6: Lưu tham số thứ ba.
- R7: Lưu tham số thứ tư.
- R8: Lưu tham số thứ năm

2.3. Chuẩn bị cho đa chương


- Tạo đồng bộ hóa

Hàm Semaphore(char * debugName, int initialValue) (Threads→synch.h)

Hàm void P(): Giảm biến đếm semaphore xuống, block tiến trình nếu như biến đếm này bằng 0.

Hàm void V(): Tăng biến đếm semaphore lên và gọi một tiến trình thực thi nếu tiến trình này đang chờ thực thi từ hàng đợi queue.

- Tạo thread
Các thread được tạo từ “Cài đặt trước”
- Tạo các BitMap: (userprog→bitmap.h)
 - Lớp này để lưu vết các tiến trình hiện hành.
- Tạo các hàm cần dùng:

- **void Mark(int which):** Đánh dấu khung trang này được sử dụng.
 - **int Find():** Tìm một khung trang trống và đánh dấu nó đã được sử dụng.
 - **int NumClear():** Trả về tổng số khung trang còn trống trên bộ nhớ.
 - **Tạo PCB: (userprog.h)**
Thêm vào ex và userprog các tiến trình:
 - **int pid:** Định danh của tiến trình để phân biệt các tiến trình.
 - **Thread* thread;** Lưu tiến trình được nạp.
 - **int parentID:** id của tiến trình cha.
 - **FileName:** Lưu tên của tiến trình.
 - **3 thuộc tính Semaphore:** Để quản lý quá trình Join, Exit và nạp chương trình.
 - **Tạo PTable:**
 - Dùng để quản lý các tiến trình được chạy trong hệ thống.
-  **PCB* pcb[MAX_PROCESSES]** là một bảng mô tả tiến trình có cấu trúc mảng một chiều có số phần tử tối đa $MAX_PROCESSES = 10$ theo yêu cầu của đồ án. Mỗi phần tử là một con trỏ lưu trữ đối tượng của lớp PCB. Hàm constructor của lớp sẽ khởi tạo tiến trình cha (là tiến trình đầu tiên) ở vị trí 0 tương đương với phần tử đầu tiên của mảng. Từ tiến trình này, chúng ta sẽ tạo ra các tiến trình con thông qua system call Exec() .

3. Cài đặt đa chương

3.1. Cài đặt chi tiết:

Bước 1: Khai báo các biến toàn cục trong `./threads/system.h` và tạo đối tượng trong `system.cc`.

Bước 2: Cài đặt 2 lớp PCB và PTable và tiến hành khai báo trong file để quản lý tiến trình “`Makefile.common`” 2 lớp vừa thêm.

Bước 3: Resize lại số khung trang và kích thước của Sector

Bước 4: Chỉnh sửa lại class Thread trong (`threads`→`thread.h`)

- Thêm `int processID` để quản lý ID phân biệt giữa các tiến trình.
- Thêm `int exitStatus` để kiểm tra exit code của tiến trình.
- Cài đặt hàm `void FreeSpace()` để giải phóng vùng nhớ trên bộ nhớ mà tiến trình đang dùng.

Bước 5: Cài đặt hàm `StartProcess_2(int id)` (`userprog`→`progtest.cc`); Là hàm dùng để hàm Fork trở hàm này đến vùng nhớ của tiến trình con.

Cách làm:

- Tạo ra các tiến trình con qua system call `Exec()` . trong `Exec` , ta tạo 1 `new Thread x` rồi cho `x->Fork(con` trở đến hàm `StartProcess (VoidFunctionPtr)` , ID của tiến trình – nếu có (`int`)).
- Cấp phát vùng nhớ Stack cho tiến trình . Trong lúc cấp phát vùng nhớ Stack này , Fork gán con trở hàm `VoidFunctionPtr` , và số nguyên `int` (số nguyên này là tham số của hàm được trở bởi `VoidFunctionPtr`)vào các thanh ghi đặc biệt, sau đó chuyển tiến trình vào `ReadyList`

Bước 6: Cài đặt lớp `AddressSpace` (`userprog`→`addrspace.cc` và `addrspace.h`).

- Việc này nên thực hiện trước khi tạo `Thread::Fork()`

Cách làm:

- Xử lý giải phóng bộ nhớ khi user program kết thúc.

- Thay đổi đoạn lệnh nạp user program lên bộ nhớ. Hiện tại, việc cấp phát không gian địa chỉ giả thiết rằng một tiến trình được nạp vào các đoạn liên tiếp nhau trong bộ nhớ. Một khi hỗ trợ đa chương trình, bộ nhớ sẽ không còn biểu diễn liên tiếp nhau nữa. -> Tạo một `pageTable = new TranslationEntry[numPages]`, tìm trang còn trống bằng phương thức `Find()` của lớp `Bitmap`, sau đó nạp chương trình lên bộ nhớ chính.
`pageTable[i].physicalPage = gPhysPageBitMap->Find();`

Bước 7: Cài đặt system call

**System call Exec:*

- Khai báo trong (`userprog→syscall.h`)
 - `SpaceId Exec(char *name);`
- Cài đặt hàm `Exec(char *name, int pid)` ở lớp `PCB`
- Cài đặt hàm `ExecUpdate(char* name)` ở lớp `PTable`.
 - Input: Địa chỉ vùng nhớ user của file
 - Output: -1 = Lỗi, 0 = Thành công
 - Mục đích: Tạo ra file với tham số là tên file

**System call Join:*

- Khai báo ở (`userprog→syscall.h`)
 - `int Join(SpaceID id).`
- Cài đặt: `JointWait(); ExitRelease()` ở lớp `PCB`
- Cài đặt: `JoinUpdate(int id)` ở lớp `PTable`

**System call Exit:*

- Khai báo ở (`userprog→syscall.h`)
 - `void Exit(int exitCode).`
- Cài đặt hàm: `JoinRelease(), ExitWait()` ở lớp `PCB`.
- Cài đặt: `ExitUpdate(int exitcode)` ở lớp `PTable`

[illegible]

5. MÔI TRƯỜNG LẬP TRÌNH:

- Ngôn ngữ lập trình: c, c++
- Môi trường giả lập: Oracle VM VirtualBox
- Hệ điều hành: Ubuntu (32-bit) Nachos-3.4

6. BẢNG PHÂN CÔNG CÔNG VIỆC CỦA THÀNH VIÊN:

Chức năng	Thành Viên
1.a,b,c	Lưu Minh Phát
2.a,b	Nguyễn Đình Văn
2.c Viết báo cáo	Phạm Minh Đức

7. TÀI LIỆU THAM KHẢO:

- Thiết kế đa chương
<http://candlecharts.com.vn/tiendung/kodeblog/2007/11/07/nachos/>
- Tham khảo code và trình bày:
<https://github.com/nguyenthanhchungfit/Nachos-Programing-HCMUS/tree/master/project/nachos-3.4/code>
<https://github.com/dangkhoasdc/nachos>
- Tài liệu tham khảo thông hiểu nachos cơ bản :
Moddle HCMUS – Hệ Điều Hành