HỆ ĐIỀU HÀNH

ĐỒ ÁN 3

***GVLT**: Trần Trung Dũng*
***TG**: Lê Giang Thanh*
***TG**: Lê Quốc Hòa*

Bộ môn Mạng máy tính
Khoa Công nghệ thông tin
Đại học Khoa học tự nhiên TP HCM

MỤC LỤC

ĐỒ ÁN 3

1 Thông tin nhóm

MSSV	Họ Tên	Email	Điện thoại
1512479	Nguyễn Duy Tâm	1512479@student.hcmus.edu.vn	01666942492
1512157	Cao Nguyễn Minh Hiếu	1512157@student.hcmus.edu.vn	01652592239

2 Báo cáo đồ án

- Để quản lý các frames bộ nhớ vật lý hệ điều hành Nachos cung cấp một biến toàn cục BitMap* gPhysPageBitMap. Biến này cần được khai báo trong system.h, system.cc và chỉ thị biên dịch trong Makefile.common. Ngoài ra trong đồ án này, ta còn phải sử dụng nhiều biến toàn cục khác (sẽ trình bày ở phần sau) Với các biến toàn khác ta cũng khai báo và chỉ thị biên dịch tương tự.

- Phần nạp chương trình vào bộ nhớ được khai báo và cài đặt trong phương thức khởi tạo của lớp Addrspace. Nhóm đã thêm một phương thức khởi tạo với tham số là char * name (được cài đặt tương tự như hàm khởi tạo đã có), điểm khác biệt ở đây là nó cần xử lý thêm phần mở file thực thi theo tên truyền vào chứ không dùng file đã mở sẵn (kiểu OpenFile *). Vấn đề phát sinh là do nhiều chương trình cùng nạp lên bộ nhớ cùng lúc sẽ có thể bị tranh đoạt toài nguyên vùng nhớ vậy lý. Điều này được giải quyết bằng cách sử dụng semaphore qua biến Semaphore* addrLock để bảo về cùng nhớ vật lý, hai phương thức được dùng để quản lý là Semaphore::P() và Semaphore::V() (tương ứng với hai phương thức Up() và Down() trong lý thuyết). Ngoài ra, vấn đề bộ nhớ vật lý

không còn liên tục sẽ được xử lý thông qua biến toàn cục gPhysPageBitMap, ta dùng phương thức BitMap::Find() để tìm vùng nhớ còn trống thay vì nạp lên liên tiếp như trước.

❖ Trước khi trình bày về cách thiết kế và cài đặt các syscall, sẽ trình bày sơ bộ về các lớp (các biến toàn cục) được thêm mới vào đồ án này.

- **Lớp PCB:** đây là một lớp để dành để lưu thông tin của một process. Lớp này cung cấp một số chức năng tiêu biểu cần lưu ý để sử dụng cho hợp lý trong đồ án này: Exec(…), JoinWait(…), JoinRelease(…), ExitWait(…), ExitRelease(…). Trong đó, phương thức Exec(…) tạo thêm một tiến trình mới chính là chương trình của user program; các phương thức còn lại phục vụ cho quá trình thực thi giữa tiến trình cha và tiến trình con đảm bảo cho trình tự chạy đúng và hoàn thành nhiệm của tất cả các tiến trình. Thực chất trong các 4 phương thức này là sử dụng semaphore để bảo vệ thứ tự thực thi của chúng cho phù hợp.

- **Lớp Ptable:** đây thực chất là một bảng để lưu thông tin của các tiến trình, về mặt cài đặt thì nó sẽ có một mảng lưu các đối tượng kiểu PCB; đối với đồ án này thì kích thước được giới hạn là 10. Lớp này chịu trách nhiệm quản lý quá trình nạp các tiến trình, thực thi, join và exit chúng. Ngoài ra nó còn có các hàm hỗ trợ khác như kiểm tra id, xóa id tiến trình đã xong trong mảng,… Với hình thức lưu trữ dạng mảng thì mỗi đối tượng PCB (hay tiến trình) có một id tương ứng để xác định, khi cần thao tác với tiến trình nào ta chỉ cần thao tác với đối tượng biến toàn cục pTable thông qua id của đối tượng PCB đó.

- **Lớp Sem:** thực chất lớp này là một các cấu trúc lại một đối tượng semaphore để bổ sung thêm thông tin như tên semaphore và các phương thức tương ứng, tạo sự thuận thiện cho việc quản lý. Đề yêu cầu dùng tên là Lock nhưng Lock đã có nên nhóm đổi tên thành Sem.

- **Lớp STable:** tượng tự như lớp PTable dùng để quản lý các đối tượng PCB thì lớp STable dùng để quản lý các semaphore. Hình thức lưu trữ là mảng được sử dụng và mỗi semaphore có một id tương ứng để quản lý. Khi cần tương tác với semaphore nào, ta chỉ cần tương tác với đối tượng biến toàn cục semTab thông qua id của semaphore tương ứng. Kích thước tối đa lưu trữ trong đồ án này là 10 semaphore.

a) Syscall SpaceID Exec(char name)*

- Syscall này thực hiện quá trình chạy một user program bằng cách sao chép tên của chương trình (tên file thực thi của nó) vào kernel space bằng hàm User2System do ban đầu tên chương trình đang nằm trong user space. Việc thực thì này được gọi thông qua phương thức PTable::ExecUpdate(…) của biến pTable. Trong đó, nó sẽ nạp chương trình vào bộ nhớ, tạo các đối tượng chứa thông tin đặc tả (PCB) rồi tạo tiến trình thực thi (PCB::Exec).

- Kết quả trả về của hàm cho biết quá trình thực thi thành công hay thất bại. Một số trường hợp có thể gây thất bại là tên file thực thi không tồn tại, thiếu bộ nhớ, đầy mảng trong pTable (vì MAX_PROCESS của nó là 10),…

b) system calls int Join(SpaceID id) và void Exit(int exitCode)

- Syscall Join(...) sử dụng tham số đầu vào là SpaceID id, thực chất nó chính là chỉnh số của đối tượng PCB chứa đặc tả của tiến trình trong mảng lưu trữ của đối tượng pTable.

Sau đó gọi phương thức PTable::JoinUpdate(...). Hoạt động trong phương thức này là tiến trình hiện tại sẽ chờ tiến trinh con kết thúc rồi cho phép nó kết thúc.

- Syscall Exit(...) tương tự như syscall join, exit được thực hiện nhờ gọi phương thức của lớp PTable::ExitUpdate(...). Trong đó sẽ cho dừng tiến trình và xóa id của nó ra khỏi bảng lưu trữ của pTable, xóa ở đây bao gồm cả việc giải phóng vùng nhớ cho phần tử PCB* đang giữ địa chỉ vùng nhớ chứa đặc tả của tiến trình.

c) system call int CreateSemaphore(char name, int semval)*

- Syscall CreateSemaphore nhận hai giá trị đầu vào là tên và giá trị của semaphore. Do chuỗi ký tự tên semaphore nằm ngoài user space nên phải sao chép vào kernel space bằng hàm User2System. Quá trình tạo semphore này được biến toàn cục STable * semTab xử lý, nó sẽ tạo một đối tượng Sem mới dựa trên thông tin truyền vào và lưu trữ vào mảng để xử lý về sau.

d) system call int Up(char name), và int Down(char* name)*

- Syscall Up và Down đều nhận vào tên của semaphore nên để tương tác cần sao chép tên này từ user space vào kernel space bằng hàm User2System. Quá trình này thực chất là đối tượng semTab sẽ tìm trong bảng lưu trữ của nó xem Sem nào có tên như vậy để gọi phương thức Semaphore::Wait() (cho Down) và Semaphore::Signal (cho Up) (mà thực chất cũng chỉ là gọi phương thức Semaphore::P() và Semaphore::V()).