ĐỒ ÁN 2

QUY ĐỊNH

- Đồ án được làm theo nhóm: 2-3 sinh viên/ nhóm.
- Bài làm giống nhau giữa các nhóm, tất cả các nhóm liên quan đều bị điểm 0 phần thực hành bất kể lý do gì và xem như KHÔNG ĐẠT môn học cho dù tổng điểm >= 5.

CÁCH THỨC NỘP BÀI

- Nộp bài trực tiếp trên moodle.
- Tên file: MSSV1_MSSV2.rar/zip (Với MSSV1 < MSSV2). Ví dụ: Nhóm gồm 2 sinh viên: 1212001 và 1212002 thì tên file đặt là: 1212001_1212002.zip/rar
- Tổ chức thư mục nộp bài gồm:
    1. Report: chứa báo cáo về bài làm của mình
    2. Source: chứa source code của chương trình
- Nếu làm không đúng những yêu cầu trên, bài làm sẽ không được chấm.

PHẦN I. QUẢN LÝ HỆ THỐNG TẬP TIN

1. Cài đặt system call int CreateFile(char *name). CreateFile system call sẽ sử dụng Nachos FileSystem Object để tạo một file rỗng. Chú ý rằng filename đang ở trong user space, có nghĩa là buffer mà con trỏ trong user space trỏ tới phải được chuyển từ vùng nhớ user space tới vùng nhớ system space. System call CreateFile trả về 0 nếu thành công và -1 nếu có lỗi.

2. Cài đặt system call OpenFileID Open(char *name, int type) và int Close(OpenFileID id). User program có thể mở 2 loại file, file chỉ đọc và file đọc - ghi. Mỗi tiến trình sẽ được cấp một bảng mô tả file với kích thước cố định. Đồ án này, kích thước của bảng mô tả file là có thể lưu được đặc tả của 10 files. Trong đó, 2 phần tử đầu, ô 0 và ô 1 để dành cho console input và console output. System call mở file phải làm nhiệm vụ chuyển đổi địa chỉ buffer trong user space khi cần thiết và viết hàm xử lý phù hợp trong kernel. Bạn sẽ dùng đối tượng filesystem trong thư mục filesys. System call Open sẽ trảvề id của file (OpenFileID = một số nguyên), hoặc là -1 nếu bị lỗi. Mở file có thể bị lỗi như trường hợp là không tồn tại tên file hay không đủ ô nhớ trong bảng mô tả file. Thamsố type = 0 cho mở file đọc và ghi, = 1 cho file chỉ đọc. Nếu tham số truyền bị sai thì system call phải báo lỗi. System call sẽ trả về -1 nếu bị lỗi và 0 nếu thành công.

3. Cài đặt system call int Read(char *buffer, int charcount, OpenFileID id) và int Write(char *buffer, int charcount, OpenFileID id). Các system call đọc và ghi vào file với id cho trước. Bạn cần phải chuyển vùng nhớ giữa user space và system space, và cần phải phân biệt giữa Console IO (OpenFileID 0, 1) và File. Lệnh Read và Write sẽ làm việc như sau: Phần console read và write, bạn sẽ sử dụng lớp SynchConsole. Được khởi tạo qua biến toàn cục gSynchConsole (bạn

phải khai báo biến này). Bạn sẽ sử dụng các hàm mặc định của SynchConsole để đọc và ghi, tuy nhiên bạn phải chịu trách nhiệm trả về đúng giá trị cho user. Đọc và ghi với Console sẽ trả về số bytes đọc và ghi thật sự, chứ không phải số bytes được yêu cầu. Trong trường hợp đọc hay ghi vào console bị lỗi thì trả về -1. Nếu đang đọc từ console và chạm tới cuối file thì trả về -2. Đọc và ghi vào console sẽ sử dụng dữ liệu ASCII để cho input và output, (ASCII dùng kết thúc chuỗi là NULL (\0)). Phần đọc, ghi vào file, bạn sẽ sử dụng các lớp được cung cấp trong file system. Bạn sử dụng các hàm mặc định có sẵn của filesystem và thông số trả về cũng phải giống như việc trả về trong synchconsole. Cả read và write trả số kí tự đọc, ghi thật sự. Cả Read và Write trả về -1 nếu bị lỗi và -2 nếu cuối file. Cả Read và Write sử dụng dữ liệu binary.

4. Cài đặt system call int Seek (int pos, OpenFileID id). Seek sẽ phải chuyển con trỏ tới vị trí thích hợp. pos lưu vị trí cần chuyển tới, nếu pos = -1 thì di chuyển đến cuối file. Trả về vị trí thực sự trong file nếu thành công và -1 nếu bị lỗi. Gọi Seek trên console phải báo lỗi.

5. Viết chương trình createfile để kiểm tra system call CreateFile. Bạn sẽ dùng tên file được nhập vào từ console nếu như phần console IO của bạn là chạy được.

6. Viết chương trình echo, mỗi khi nhập một dòng từ console thì console xuất lại dòng đó

7. Viết chương trình cat , yêu cầu nhập vô filename, rồi hiển thị nội dung của file đó

8. Viết chương trình copy, yêu cầu nhập tên file nguồn và file đích và thực hiện copy

9. Viết chương trình reverse, yêu cần nhập tên file nguồn và file đích, đọc nội dung file nguồn, đảo nghịch nội dung và ghi vào file đích.

PHẦN II. ĐA CHƯƠNG, ĐỒNG BỘ HÓA

I. Hiểu về đa chương

Chương trình người dùng trên HĐH thật sẽ chạy dưới dạng tiến trình (process), còn chương trình người dùng của Nachos chạy trong một tiểu trình (thread). Do đó, để thực thi đa chương ta cần biết cách tạo tiểu trình và hiểu về sự điều phối các tiểu trình trong Nachos.

Chúng ta cần viết **system call Exec** để thực thi một chương trình mới trong process hiện tại. Thực chất system call Exec làm các việc sau:

- Tạo ra một không gian địa chỉ mới
- Load chương trình vào khoảng bộ nhớ mới được cấp phát
- Sau đó tạo tiểu trình mới (bằng phương thức Thread::Fork()) để thực thi chương trình.

Lưu ý là sau khi tiểu trình mới chạy, giữa tiểu trình cha và tiểu trình con có thể xảy ra hiện tượng race condition do việc truy cập bộ nhớ, do đó chúng ta ***cần phải đồng bộ giữa 2 tiểu trình***.

Hiện tại, Nachos sử dụng thuật toán round-robin với time slice cố định để điều phối giữa các tiểu trình. Khi một tiểu trình đang thực thi trên máy ảo MIPS mà time slice dành cho nó đã hết thì một interrupt được phát sinh, tạo ra một trap vào system mode. Trong kernel space, bộ lập lịch của CPU (scheduler) ngừng tiểu trình hiện tại và lưu lại trạng thái (state) của nó. Sau đó scheduler tìm tiểu trình kế tiếp trên danh sách, phục hồi trạng thái của nó và cho CPU tiếp tục chạy.

Các tập tin trong đồ án này:

- **progtest.cc** kiểm tra các thủ tục để chạy chương trình người dùng
- **syscall.h** system call interface: các thủ tục ở kernel mà chương trình người dùng có thể gọi
- **exception.cc** xử lý system call và các exception khác ở mức user, ví dụ như lỗi trang, trong phần mã chúng tôi cung cấp, chỉ có 'halt' system call được viết
- **bitmap.*** các hàm xử lý cho lớp bitmap (hữu ích cho việc lưu vết các ô nhớ vật lý) **filesys.h**
- **machine.*** mô phỏng các thành phần của máy tính khi thực thi chương trình người dùng: bộ nhớ chính, thanh ghi, v.v.
- **mipssim.cc** mô phỏng tập lệnh của MIPS R2/3000 processor
- **console.*** mô phỏng thiết bị đầu cuối sử dụng UNIX files. Một thiết bị có đặc tính (i) đơn vị dữ liệu theo byte, (ii) đọc và ghi các bytes cùng một thời điểm, (iii) các bytes đến bất đồng bộ
- **synchconsole.*** nhóm hàm cho việc quản lý nhập xuất I/O theo dòng trong Nachos.
- **../test/*** Các chương trình C sẽ được biên dịch theo MIPS và chạy trong Nachos.
- **thread.*:** Các hàm liên quan tới việc quản lý các tiểu trình bên trong hệ thống Nachos như: Cấp phát stack, đưa một tiểu trình vào trạng tháy sleep, thay đổi trạng thái hoạt động của một tiểu trình, lưu trữ trạng thái khi xuất hiện "context switching".
- **synch.*:** Gồm các lớp thực hiện việc đồng bộ như: Semaphore, Lock, …
- **scheduler.***: Gồm các hàm quản lý việc lập lịch và điều phối các tiểu trình.
- **list.*:** Lớp dùng để quản lý danh sách các đối tượng.
- **addrspace.*:** Lớp dùng để quản lý việc cấp phát và thu hồi bộ nhớ cho tiến trình.

II. Đa chương và đồng bộ

Tham Khảo:

- **[5] Da Chuong & Dong Bo Hoa**
- **[3] Cach Viet Mot SystemCall**

Trong phần này chúng ta sẽ thiết kế và cài đặt để hỗ trợ đa chương trình trên Nachos. Các bạn phải viết thêm các system calls về quản lý tiến trình và giao tiếp giữa các tiến trình. Nachos hiện tại chỉ là môi trường đơn chương. Chúng ta sẽ lập trình để cho mỗi tiến trình được duy trì trong system thread của nó. Chúng ta phải quản lý việc cấp phát và thu hồi bộ nhớ, quản lý phần dữ liệu và đồng bộ hóa các tiến trình/ tiểu trình. Lưu ý nên thiết kế giải pháp trước khi lập trình. Chi tiết như sau:

1. Thay đổi mã cho các exception khác (không phải system call exceptions) để tiến trình có thể hoàn tất, chứ không halt máy như trước đây. Một run time exception sẽ không gây ra việc HĐH phải shut down. Xử lý cho việc đồng bộ hóa khi tiến trình kết thúc.

2. Cài đặt đa tiến trình. Chương trình hiện tại giới hạn bạn chỉ thực thi 1 chương trình, bạn phải có vài thay đổi trong file addrspace.h và addrspace.cc để chuyển hệ thống từ đơn chương thành đa chương. Bạn sẽ cần phải:
   a. Giải quyết vấn đề cấp phát các frames bộ nhớ vật lý, sao cho nhiều chương trình có thể nạp lên bộ nhớ cùng một lúc.
   b. Phải xử lý giải phóng bộ nhớ khi chương trình người dùng kết thúc.
   c. Phần quan trọng là thay đổi đoạn lệnh nạp chương trình người dùng lên bộ nhớ. Hiện tại, việc cấp phát không gian địa chỉ giả thiết rằng một chương trình được nạp vào các đoạn liên tiếp nhau trong bộ nhớ. Một khi chúng ta hỗ trợ đa chương trình, bộ nhớ sẽ không còn biểu diễn liên tiếp nhau nữa. Nếu chúng ta lập trình không đúng thì khi nạp một chương trình mới có thể phá hỏng HĐH.

3. Cài đặt system call **SpaceID Exec(char* name).** Exec gọi thực thi một chương trình mới trong một system thread mới. Bạn cần phải đọc hiểu hàm "StartProcess" trong progtest.cc để biết cách khởi tạo một user space trong 1 system thread. Exec trả về -1 nếu bị lỗi và thành công thì trả về Process SpaceID của chương trình người dùng vừa được tạo. Đây là thông tin cần phải quản lý trong lớp Ptable.

4. Cài đặt system calls **int Join(SpaceID id)** và **void Exit(int exitCode)**. **Join** sẽ đợi và block dựa trên tham số "SpaceID id". **Exit** trả về exit code cho tiến trình nó đã join. Exit code là 0 nếu chương trình hoàn thành thành công, các trường hợp khác trả về mã lỗi. Mã lỗi được trả về thông qua biến exitcode. Join trả về exit code cho tiến trình nó đã đang block trong đó, -1 nếu join bị lỗi. Một user program chỉ có thể join vào những tiến trình mà đã được tạo bằng system call Exec. Bạn không thể join vào các tiến trình khác hoặc chính tiến trình mình. Bạn phải sử dụng semaphore để phối hợp hoạt động giữa Join và Exit của chương trình người dùng.

5. Cài đặt system call **int CreateSemaphore(char* name, int semval)**. Bạn tạo cấu trúc dữ liệu để lưu 10 semaphore. System call **CreateSemaphore** trả về 0 nếu thành công, ngược lại thì trả về -1.

6. Cài đặt system call **int Up(char* name),** và **int Down(char* name)**. Tham số name là tên của **semaphore**. Cả hai system call trả về 0 nếu thành công và -1 nếu lỗi. Lỗi có thể xảy ra nếu người dùng sử dụng sai tên semaphore hay semaphore đó chưa được tạo.

7. Cài đặt một chương trình **shell** đơn giản để kiểm tra các system call đã cài đặt ở trên. **Shell** nhận một lệnh tại một thời điểm và thực thi chương trình tương ứng. **Shell** nên **"join"** mỗi chương trình và đợi cho đến khi chương trình kết thúc. Khi hàm **Join** trả về, hiển thị ra exitcode nếu nó khác 0. **Shell** của bạn cũng phải cho phép chương trình chạy background. Bất kì lệnh nào bắt đầu bằng '&' thì chạy theo dạng background. (ví dụ: &create, thì chương trình create chạy theo mode background).

PHẦN III. BÁO CÁO

Cách thiết kế hệ thống quản lý tập tin.

Mô tả bạn đã thiết kế và cài đặt như thế nào, tại sao làm như vậy. Vui lòng không copy mã chương trình của các bạn vào phần báo cáo.

Nộp chương trình: khi bạn hoàn thành, xóa các object file và các file thực thi. Tar hoặc nén file đó lại theo mã số sinh viên của 1 bạn trong nhóm.

Chú ý: Không để cho user có thể làm sụp HĐH, system call nên xử lý càng nhiều trường hợp càng tốt.