ĐỒ ÁN 3 – HỆ ĐIỀU HÀNH
LỚP CNTN2015

1. Mô tả

- Đồ án được làm theo nhóm: tối đa 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.

2. Cách thức nộp bài

- Nộp bài trực tiếp trên moodle.
- Tên file: MSSV1_MSSV2_MSSV3.rar/zip (Với MSSV1 < MSSV2 < MSSV3)
- Ví dụ: Nhóm gồm 3 sinh viên: 1512001, 1512002 và 1512003 thì tên file đặt là:

  1512001_1512002_1512003.zip/rar
- Tổ chức thư mục nộp bài gồm:
  1. **Report**: chứa báo cáo về bài làm của mình
     - Font chữ chính là Time-New-Roman size chữ 13, cách dòng 1.5.
     - Ngắn gọn, không chép mã chương trình vào báo cáo.
     - Mô tả chi tiết các thiết kế:
       - Các hình vẽ và mô tả mô hình
       - Các hình vẽ thiết kế và mô tả thiết kế: chỉ nêu một số lớp, hàm quan trọng và ghi chú đầy đủ.
     - Những vấn đề đã làm được và những vấn đề chưa làm được.
     - Hướng dẫn sử dụng chương trình.
     - Tài liệu tham khảo.
  2. **Source**
     - Gồm toàn bộ Project, có ghi chú đầy đủ trong source code
     - Nếu làm không đúng những yêu cầu trên, bài làm sẽ không được chấm.

3. Đánh giá và các mốc thời gian hoàn thành

Mục tiêu của đồ án này tập trung chủ yếu vào 3 phần được mô tả bên dưới với thời gian thực hiện từ ngày 01/12/2017 đến 22/12/2017.

- Các system call nhập/xuất file
- Đa chương, lập lịch và đồng bộ hóa trong Nachos
- Chương trình minh họa

NỘI DUNG ĐỒ ÁN

Gồm 3 phần

❖ **System call nhập xuất file**

1. *Cài đặt system call* ***int CreateFile(char *name).***

CreateFile system call sẽ sử dụng Nachos FileSystem Object để tạo một file rỗng. Chú y' 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 và 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 ly' phù hợp trong kernel. Bạn sẽ phải dùng đối tượng filesystem trong thư mục filesys. System call Open sẽ trả về 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. Tham số 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. 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

❖ **Đa chương, lập lịch và đồng bộ hóa trong Nachos**

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 user program kết thúc.

c) Phần quan trọng là 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 chúng ta hỗ trợ đa chương trình, bộ nhớ sẽ không còn biểu diễn liên tiếp nhau nữa. Nếu chúng ta không lập trình đúng đắn thì khi nạp một chương trình mới có thể làm phá hỏng HĐH của bạn.

4. *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.

5. *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 Joining và Exiting của tiến trình người dùng.

6. *Cài đặt system call* ***int CreateSemaphore(char* name, int semval)***.

Như trong project 1 bạn phải cập nhật file start.s, start.c và syscall.h để thêm system call mới. 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.

7. *Cài đặt system* ***call int Wait(char* name),*** *và* ***int Signal(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. Xem gợi ý khai báo lớp quản lý các semaphore mà Nachos tạo ra để cung cấp cho chương trình người dùng ở phụ lục.

❖ **Chương trình minh họa**

Hãy xây dựng ứng dụng "Thống kê sử dụng máy nóng lạnh" để kiểm tra các system call (**CreateFile, Open, Read, Write, Exec, Join, Exit, CreateSemaphore, Wait, Signal**) được bổ xung của Source code. Bài toán được mô tả như sau:

Trong đại sảnh của toà nhà mới (nhà I) trường ĐH KHTN có đặt một máy cung cấp nước nóng lạnh tự động để phục vụ sinh viên, máy có 2 vòi rót nước. Mỗi khi sinh viên cảm thấy khát nước, họ sẽ tiến đến chiếc máy rót nước tự động để rót đầy chai nước mà họ mang theo để uống. Máy rót nước tự động sẽ rót đầy nước vào chai ngay sau khi chai (đã hết nước) được đặt vào vị trí vòi. Tốc độ rót nước của 2 vòi là như nhau và mỗi vòi rót đầy chai nước 1 lít trong 10 giây. Do chỉ có 2 vòi nên tại một thời điểm chỉ có 2 sinh viên được uống nước, những sinh viên đến sau phải đợi để chờ đến lượt mình.

Hãy tạo một giải pháp cho Sinh Viên/Vòi Nước ở mức người dùng. Giải pháp gồm 3 chương trình: chương trình main khởi đầu, Sinh Viên, Vòi Nước. Chương trình main sẽ đọc sinh viên và dung tích chai nước mà sinh viên mang theo từ file input.txt. Chương trình chính sẽ gọi thực thi Exec các Sinh Viên và Vòi Nước và đợi đến khi tất cả các tiến trình kết thúc. Mỗi Sinh Viên ghi vào file cho biết sinh viên nào uống nước và mỗi Vòi Nước ghi vào file cho biết vòi nước nào rót nước cho sinh viên uống. Kết quả ghi chung vào 1 file output.txt

➢ **File input.txt:**

- Dòng đầu là số nguyên dương N cho biết số thời điểm được xét.
- N dòng tiếp theo, mỗi dòng chứa n số nguyên dương cho biết có n Sinh viên đến uống nước và giá trị của mỗi số nguyên dương cho biết dung tích của chai nước mà sinh viên mang theo.

```
3                   //Có 3 thời điểm được xét
4   2   1           //Có 3 SV, sv1 chai 4lít, sv2 chai 2lít, sv3 chai 1lít
5   3   4   1
8   5   2
```

- **File output.txt**

```
1   2   2          //sv1 sử dụng vòi 1, sv2 vòi 2, sv3 vòi
1   2   2   1
1   2   2
```

PHỤ LỤC

Gợi y lớp Stable để cài đặt cho phần quản lý các lock mà Nachos cung cấp cho chương trình người dùng.

```
//stable.h
#ifndef STABLE_H
#define STABLE_H
#include "synch.h"
#include "bitmap.h"
#define MAX_LOCK 10
class STable {
private:
        BitMap* bm;
        Lock* lockTab[MAX_SEMAPHORE];
    public:
        STable();    ~STable();
      int create(char* name);
      int wait(char *name);
      int signal(char *name);
};
#endif
```

```
#ifndef PTABLE_H
#define PTABLE_H
#include "bitmap.h"
#include "pcb.h"
#include "schandle.h"
#include "semaphore.h"

#define MAXPROCESS 10
```

```
class PTable {
private:
  BitMap *bm;
  PCB    *pcb[MAXPROCESS];
  int     psize;
  Semaphore *bmsem; // dùng để ngăn chặn trường hợp nạp 2 tiến trình cùng lúc
public:
  PTable(int size);
// Khởi tạo  size đối tượng pcb để lưu size process. Gán giá trị ban đầu là null. Nhớ khởi tạo *bm và
*bmsem để sử dụng
```

~PTable();
// hủy các đối tượng đã tạo

int ExecUpdate();// return PID
//Thực thi cho system call SC_EXEC, Kiểm tra chương trình được gọi có tồn tại trong máy không. Kiểm tra thử xem chương trình được gọi là chính nó không? Chúng ta không cho phép điều này. Kiểm tra còn slot trống để lưu tiến trình mới không (max là 10 process). Nếu thỏa các điều kiện trên thì ta lấy index của slot trống là ProcessID của tiến trình mới tạo này, giả sử là ID. Và gọi phương thức Exec của lớp PCB với đối tượng tương ứng quản ly process này, nghĩa là gọi pcb[ID]->Exec(…). Xem chi tiết mô tả lớp PCB ở bên dưới.
Phương thức này được gọi trong hàm xử ly system call SC_Exec

int ExitUpdate(int ec);
//Được thủ tục sử ly cho system call SC_Exit sử dụng. Trong system call này chúng ta kiểm tra thử PID có tồn tại không, sau đó chúng ta mới xử ly kết thúc tiến trình, phải gọi JoinRelease() để giải phóng sự chờ đợi của tiến trình cha và ExitWait() để xin phép tiến trình cha cho kết thúc.

Chú y là sau khi gọi thủ tục này, chúng ta phải giải phóng tiểu trình hiện tại bằng các dòng lênh như sau, nếu là main process thì chúng ta gọi hàm Halt() luôn

```
int pid = currentThread->processID;

currentThread->FreeSpace();
if(pid == 0){//exit main process
  interrupt->Halt();
}
currentThread->Finish();
```

int JoinUpdate(int id);
// Được sử dụng khi mà chúng ta viết thủ tục cho system call Join(), trong thủ tục này thì tiến trình cha gọi pcb[id]->JoinWait() để chờ cho tới khi tiến trình con kết thúc (trước khi tiến trình con kết thúc thì

tiến trình con gọi JoinRelease()). Sau đó tiến trình cha lại gọi ExitRelease() để cho phép tiến trình con được kết thúc.
Chú y là chúng ta không cho phép tiến trình join vào chính nó, hoặc là join vào tiến trình khác không phải là cha của nó.

```
  int GetFreeSlot();
//Tìm free slot để lưu thông tin cho tiến trình mới

bool IsExist(int pid);
//kiểm tra có tồn tại process ID này không

  void Remove(int pid);
//Xóa một processID ra khỏi mãng quản ly nó, khi mà tiến trình này đã kết thúc
};

#endif
```

```
#ifndef PCB_H
#define PCB_H
#include "thread.h"
#include "synch.h"
class PCB{  private:
  Semaphore *joinsem;//semaphore cho quá trình join
  Semaphore *exitsem;//semaphore cho quá trình exit
Semaphore *mutex;   int        exitcode;    Thread
*thread;   int        pid;   int        numwait ;// số tiến trình
đã join

 public:
  int        parentID; //ID của tiến trình cha
```

PCB(int id); ~PCB(); int Exec(char *filename,int pid);// nạp chương trình có tên lưu trong biến **filename** và //processID sẽ là **pid** int GetID() {return pid;} int GetNumWait(); void JoinWait(); void ExitWait(); void JoinRelease(); void ExitRelease(); void IncNumWait(); void DecNumWait(); void SetExitCode(int ec){exitcode = ec;} int GetExitCode(){return exitcode;}

};

#endif