BÁO CÁO

ĐỒ ÁN 2 HỆ ĐIỀU HÀNH

LẬP TRÌNH ĐA CHƯƠNG VỚI NACHOS

MỤC LỤC

1. THÔNG TIN NHÓM

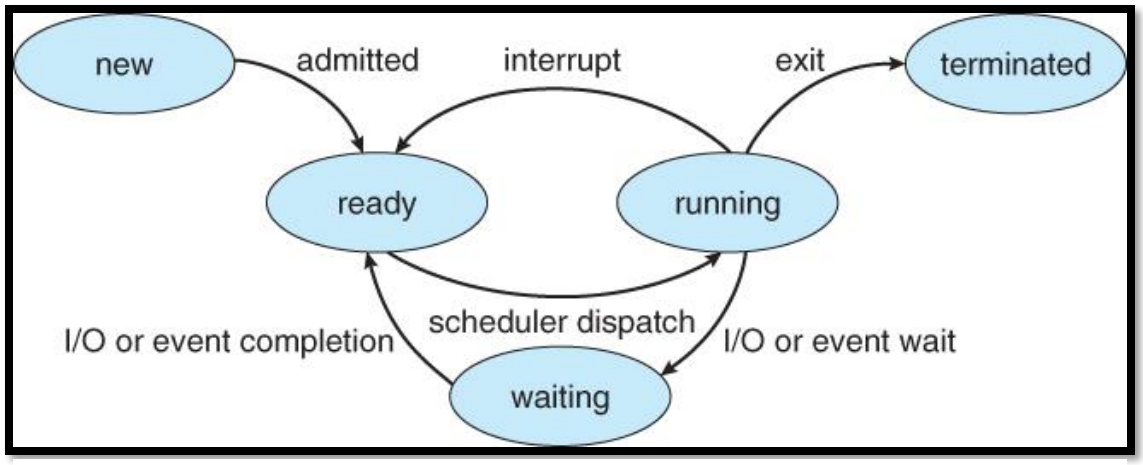
STT	MSSV	Họ tên	Mức độ đóng góp
1	1512034	Nguyễn Đăng Bình	100%
2	1512042	Nguyễn Thành Chung	100%
3	1512123	Hoàng Ngọc Đức	100%

Mức độ hoàn thành đồ án: 100% . Đã cài đặt cho chương trình chạy đa chương như đồ án yêu cầu.

2. THIẾT KẾ VÀ CÀI ĐẶT

2.1. TÌM HIỂU MỘT SỐ KHÁI NIỆM QUAN TRỌNG:

➢ Tìm hiểu về trạng thái của tiến trình:



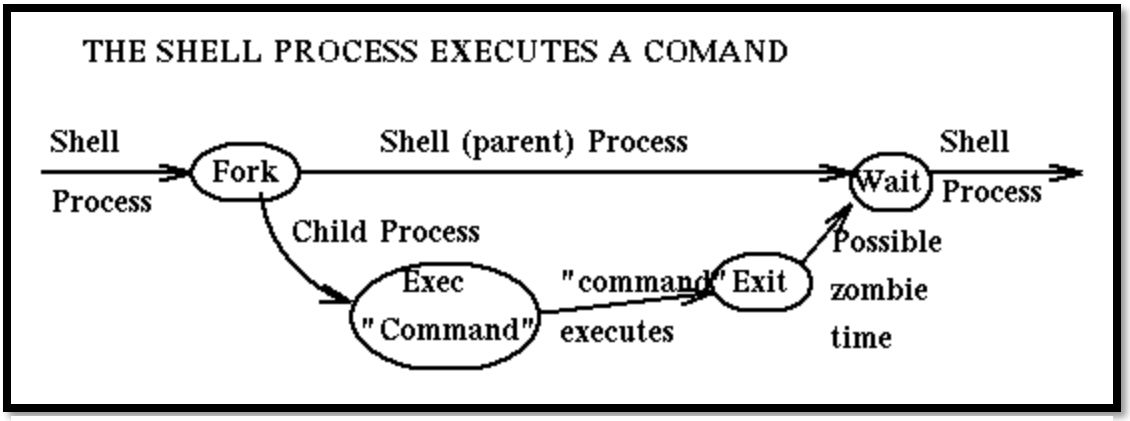
Hình 1. Process State

Mỗi tiến trình sẽ trải qua nhiều trạng thái khác nhau từ lúc hình thành cho đến khi kết thúc. Sự thay đổi trạng thái này là do quyết định của hệ điều hành.

Các trạng thái của tiến trình:

- New: Process được khởi tạo.
- Running: Tiến trình thực this au khi được nạp lên bộ xử lý.
- Waiting: Process đợi bộ điều phối tiến trình nạp nó từ bộ nhớ ngoài (như ổ đĩa cứng, flash drive hoặc CD-ROM,…) lên bộ nhớ chính.
- Ready: Tiến trình đang chờ để giao cho vi xử lý.
- Terminated: Khi tiến trình đã thực thi xong, hoặc bị kết thúc bởi hệ điều hành.

➢ Trong Unix, một tiến trình được tạo ra nhờ gọi System Call fork(). Sau khi gọi fork(), tiến trình sẽ gọi tiếp System Call Exec() để thay đổi vùng nhớ của tiến trình thành chương trình mới.

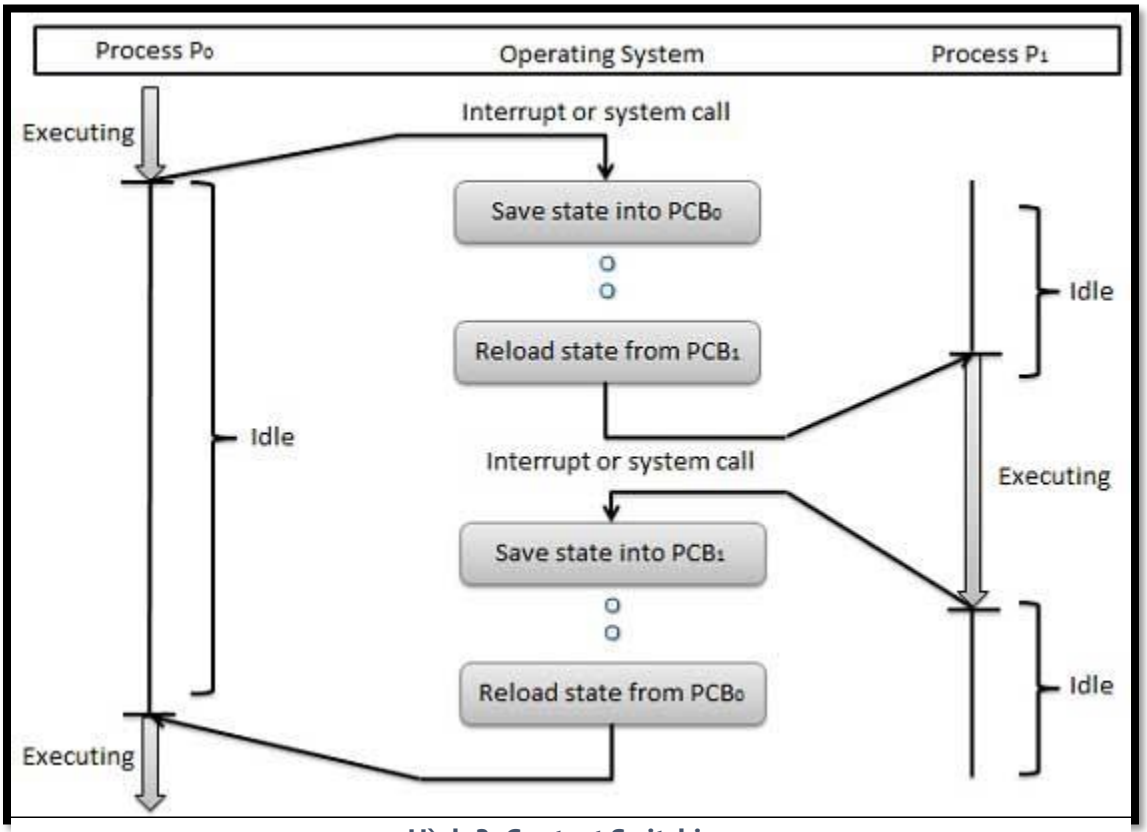


Hình 2. Process creation use fork()

2.2. CONTEXT SWITCHING

Context switching (chuyển đổi ngữ cảnh) là một quá trình lưu lại trạng thái hiện hành và chuyển sang trạng thái khác của CPU giúp cho nhiều tiến trình có thể chia sẻ một CPU duy nhất.

Là đặc tính tiêu biểu của hệ điều hành đa nhiệm (multitasks).



Hình 3. Context Switching

Nachos hiện tại chỉ là môi trường đơn chương. Chúng ta sẽ phải 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.

2.3. THIẾT KẾ GIẢI PHÁP

- Cơ sở đồng bộ hóa:

Cơ sở của đồng bộ hóa chính là đối tượng Semaphore. (./threads/synch.h)

Semaphore(char * debugName, int initialValue): Phương thức khởi tạo mặc định có tham số truyền vào là initialValue với ý nghĩa là Semaphore này sẽ có tối đa initialValue tiến trình được phép thực thi cùng lúc.

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

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.

- Lớp Thread: (./threads/thread.h)

Lớp thread tạo ra các tiểu trình bao gồm việc nạp và cấp phát vùng nhớ Stack, quản lý trạng thái của tiến trình.

Một vài thuộc tính quan trọng:

- char* name: Lưu tên của tiến trình, đường dẫn tương đối tới chương trình thực thi.
- AddrSpace * space: Vùng nhớ của tiến trình trên ram ảo.

Một vài hàm quan trọng:

- void Thread(char * debugName): Khởi tạo một tiến trình với debugName là đường dẫn tương đối tới file chương trình thực thi.

- void Fork(VoidFunctionPtr func, int arg): Cấp phát vùng nhớ Stack cho tiến trình. Trong lúc cấp phát vùng nhớ Stack này, Fork gán con trỏ hàm VoidFunctionPtr (func), và số nguyên int (arg). Số nguyên này là tham số của hàm được trỏ bởi VoidFunctionPtr, ta cài số này là id của tiểu trình phụ chạy trong nachos.

➢ Lớp BitMap: (./userprog/bitmap.h)

Lớp này để lưu vết các tiến trình hiện hành. Gồm 1 mảng cờ hiệu để đánh dấu các khung trang còn trống để nạp vào page tương ứng ở class AddrSpace.

Các hàm quan trọng 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.

➢ Lớp PCB: (./userprog/userprog.h)

PCB (Process Control Block): Lưu thông tin để quản lý process.

Một số thuộc tính quan trọng:

- 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 trìn cha.
- FileName: Lưu tên của tiến trình.
- 3 thuộc tính Semaphore: Để quản lý quá trình Join, Exit và nạp chương trình.

➢ Lớp 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().

2.4. CÀI ĐẶT CHI TIẾT:

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

```
system.h
Semaphore *addrLock;  // semaphore
BitMap *gPhysPageBitMap;       // quan ly cac frame
PTable *pTab;                  // quan ly bang tien trinh


system.cc
addrLock = new Semaphore("addrLock", 1);
gPhysPageBitMap = new BitMap(256);
pTab = new PTable(10);
```

- 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

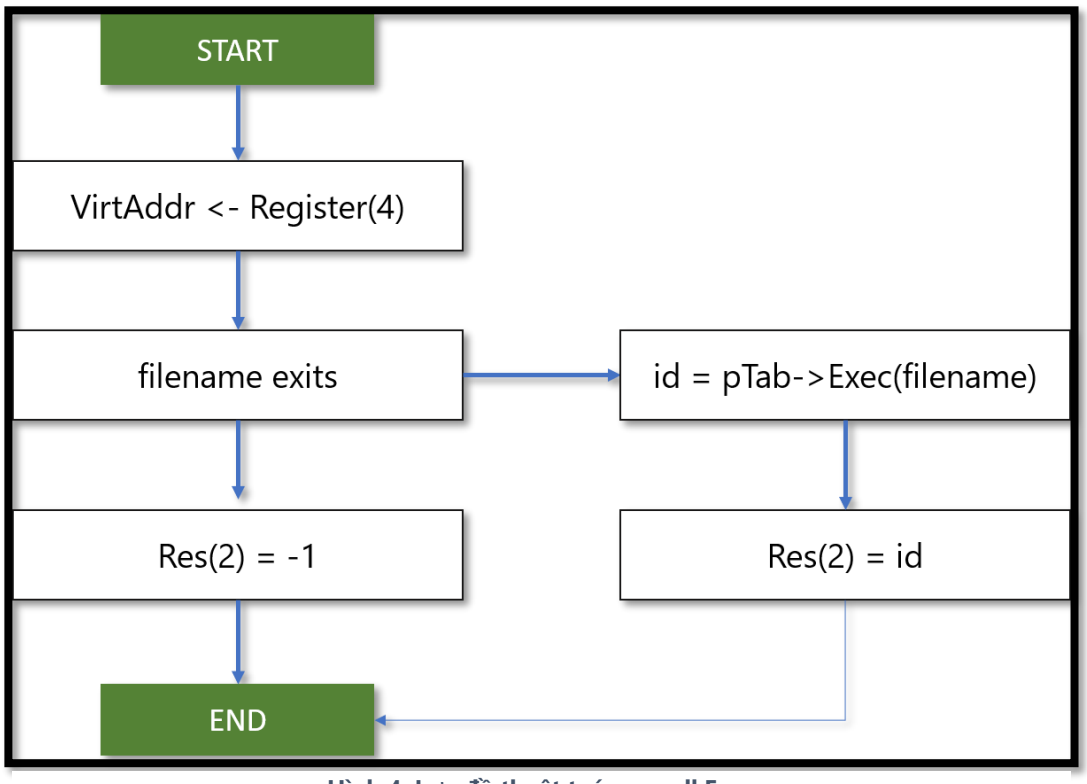
```
#define NumPhysPage 128
(./machine/machine.h)
#define SectorSize 512
(./machine/disk.h)
```

- 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 trỏ hàm này đến vùng nhớ của tiến trình con.
- Bước 6: Cài đặt lớp AddressSpace (./userprog/addrspace.cc và addrespace.h).
  - Chương trình hiện tại giới hạn chỉ thực thi 1 chương trình -> cần phải có vài thay đổi trong file addrspace.h và addrspace.cc để chuyển từ hệ thống đơn chương thành đa chương. Cụ thể:
  - 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 -> sử dụng biến toàn cục Bitmap* gPhysPageBitMap để quản lý các frames.
  - 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.

- Lưu đồ thuật toán:

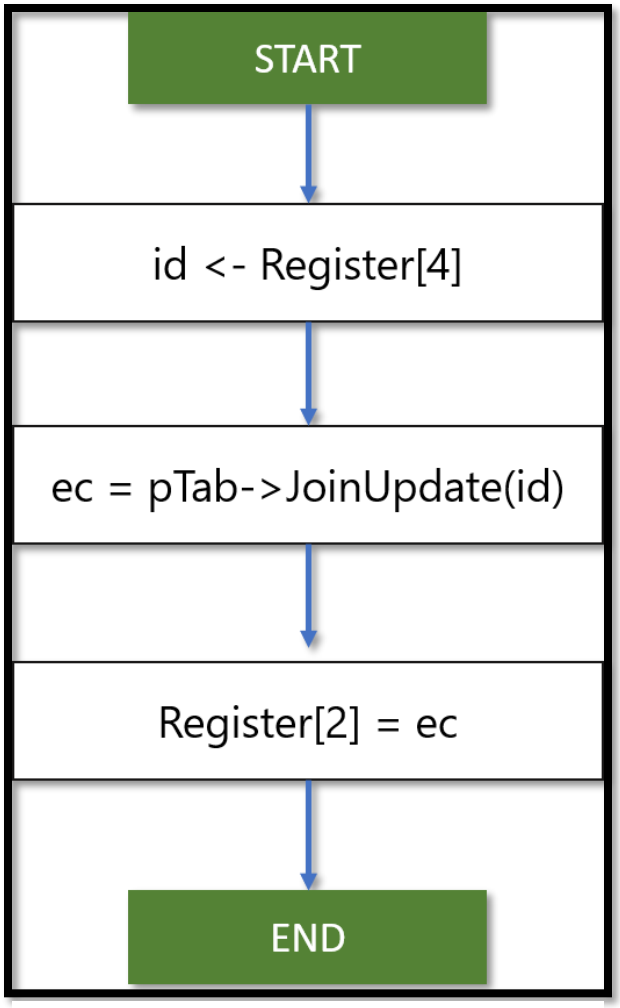


Hình 4. Lưu đồ thuật toán syscall Exec

➢ System call Join:

- Khai báo ở ./userprog/syscall.h:

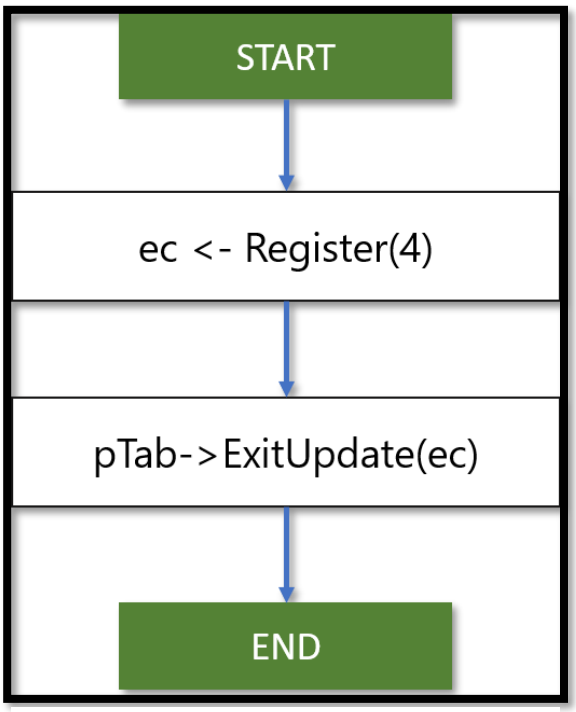
  int Join(SpaceID id).
- Cài đặt: JointWait(); ExitRelease() ở lớp PCB
- Cài đặt: JoinUpdate(int id) ở lớp PTable
- Lưu đồ:



Hình 5. Lưu đồ thuật toán của syscall Update

- 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
  - Lưu đồ



Hình 6. Lưu đồ thuật toán của syscall Exit

3. CHƯƠNG TRÌNH NGƯỜI DÙNG.

- Chương trình Ping

```
#include "syscall.h"

void main()
{
        int i;
        for(i =0; i< 1000; i++)
        {
                PrintChar('A');
        }
        Halt();
}
```

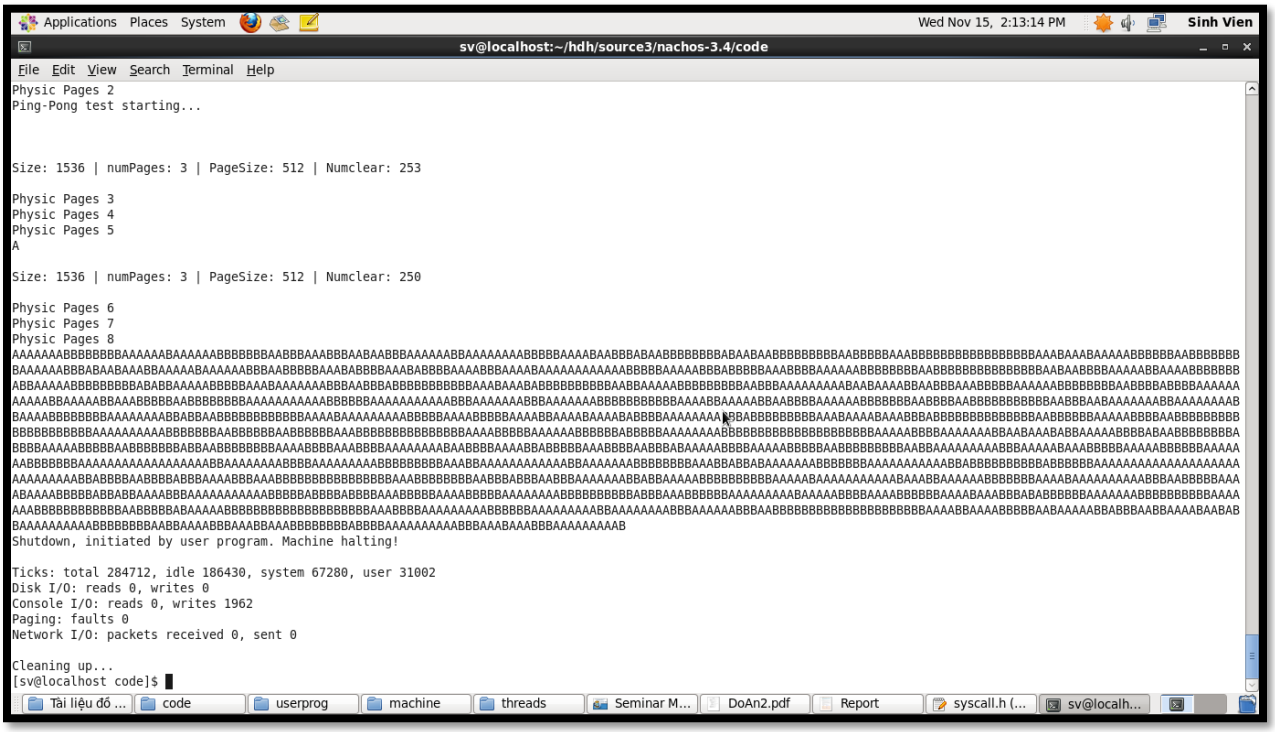
- Chương trình Pong

```
#include "syscall.h"

void main()
{
        int i;
        for(i =0; i< 1000; i++)
        {
                PrintChar('B');
        }
        Halt();
}
```

- Chương trình Scheduler

```
#include "syscall.h"


void main()
{
        int pingPID, pongPID;
        PrintString("Ping-Pong test starting...\n\n");
        pingPID = Exec("./test/ping");
        pongPID = Exec("./test/pong");
        Join(pingPID);
        Join(pongPID);
}
```

Kết quả: ./userprog/nachos -rs 1023 -x ./test/scheduler



TÀI LIỆU THAM KHẢO:

nachos_canban.pdf

nachos_study_book.pdf

[5] Da Chuong Dong Bo Hoa (1).pdf

Constructor_Cua_AddrSpace_2

Huong Dan Cac Syscall Ve Da Chuong.pdf