TRƯỜNG ĐẠI HỌC KHOA HỌC TỰ NHIÊN
KHOA CÔNG NGHỆ THÔNG TIN

Nguyễn Tạ Bảo - 21120205
Lê Hữu Hưng - 21120463
Nguyễn Thế Phong - 21120527

BÁO CÁO
PROJECT 02: MULTIPROGRAMING

GIÁO VIÊN HƯỚNG DẪN
ThS. Lê Giang Thanh
ThS. Nguyễn Thanh Quân

Tp. Hồ Chí Minh, tháng 1/2023

Mục lục

Danh sách hình

Danh sách bảng

Chương 1

Giới thiệu

Nachos là một phần mềm hướng dẫn được thiết kế để cung cấp cho sinh viên cơ hội nghiên cứu và điều chỉnh một hệ điều hành thực sự. Sự khác biệt chính giữa Nachos và hệ điều hành "thực sự" nằm ở việc Nachos hoạt động như một quy trình Unix duy nhất, so với việc chạy trực tiếp trên phần cứng vật lý như các hệ điều hành truyền thống. Tuy nhiên, Nachos mô phỏng những chức năng cấp thấp cơ bản của các máy thông thường, bao gồm ngắt, bộ nhớ ảo và I/O dựa trên ngắt (exception).

Đồ án này bao gồm việc lập trình System call cơ bản và viết các chương trình Test để kiểm tra chức năng của các System call này, hiểu rõ cách mà hệ thống hoạt động thông qua việc tìm hiểu, triển khai và kiểm thử các chức năng hệ điều hành cơ bản.

Bài báo cáo này trình bày cụ thể các nội dung như sau: Chương 1: Giới thiệu - Đây là phần giới thiệu cơ bản về Nachos, mô tả về mục tiêu của đồ án, cũng như các khái niệm cơ bản về hệ thống. Chương 2: Cài đặt System call - Tiến hành triển khai các System call cơ bản, từ việc định nghĩa đến việc thực hiện chúng trong Nachos. Chương 3: Cài đặt chương trình Test - Kiểm tra và đánh giá tính năng của các system call đã triển khai. Chương 4: Kết quả đạt được - Trình bày tổng quan các phần đã đạt được trong đồ án.

Chương 2

Multiprogramming

2.1 Thêm exception

Thêm các exception để phục vụ cho multiprogramming. Nó sẽ không làm cho hệ điều hành bị tắt(shut down) và sẽ đồng bộ hóa các tác vụ khi mà tiến trình được hoàn thành. Các exception đã có sẵn trong file `machine.h` và nhóm chỉ thêm các case vào file `exception.cc`.
Dưới đây là các exception đã được thêm vào:

- NoException
- PageFaultException
- ReadOnlyException
- BusErrorException
- AddressErrorException
- OverflowException
- IllegalInstrException
- NumExceptionTypes

2.2 Thực hiện multi-process

2.2.1 Giải quyết vấn đề phân phát frame của bộ nhớ vật lý

Dùng hàm `kernel->gPhysPageBitMap->FindAndSet()` để tìm một frame trống và lưu vào physical page của page table.

2.2.2 Giải quyết việc giải phóng các vùng nhớ khi chương trình người dùng kết thúc

Dùng vòng lặp for để chạy qua từng phần tử trong mảng pagetable và gọi `kernel->gPhysPageBitMap->Clear(pageTable[i].physicalPage)` để giải phóng bộ nhớ, trả lại frame đã đánh dấu cho gPhysPageBitMap khi chương trình người dùng kết thúc.

2.2.3 Giải quyết việc load chương trình người dùng vào bộ nhớ

Chương trình người dùng được chia thành 3 segment: code, initData và readOnlyData. Với mỗi segment, nhóm sẽ nạp vào bộ nhớ bằng cách ánh xạ thông qua pagetable. Tuy nhiên, đối với mỗi segment, các frame cuối có thể vẫn chưa được sử dụng hết bộ nhớ, do đó nhóm sẽ tính toán để đưa phần bộ nhớ này cho các segment liền sau.

Chương 3

Cài đặt System call

3.1 Cài đặt System call Exec

Trước khi cài đặt system call Exec, nhóm sẽ cài đặt những phương thức sau:

1. `Exec(char* name, int pid)` trong PCB class:
    - Sử dụng một biến Semaphore là mutex để tránh việc nạp 2 tiến trình cùng 1 thời điểm.
    - Cứ mỗi lần gọi hàm là ta sẽ tạo một thread mới, với processID của thread là pid.
    - Đồng thời nhóm cũng đặt parentID của thread này là processID của thread gọi thực thi Exec.
    - Cuối cùng gọi hàm Fork để thực thi thread vừa chạy.
2. `ExecUpdate(char* name)` trong Ptable class:
    - Sử dụng biến semaphore bmsem để quản lí việc truy cập đồng bộ Ptable.
    - Tìm index còn trống trong Ptable bằng hàm `GetFreeSlot()`
    - Dùng `PCB(index)` để tạo PCB cho process và gán nó vào vị trí ta vừa tìm được ở trên
    - Tiếp theo nhóm đặt parentID của process này là processID của process gọi thực thi ExecUpdate.
    - Sau đó gọi hàm Exec ở trên.

Exec() syscall: Gọi `pTab->ExecUpdate(name)`

3.2 Cài đặt System call Join, Exit

1. `JoinWait()` ở lớp PCB: Dùng biến semaphore joinsem. Gọi joinsem->P()
2. `JoinRelease()` ở lớp PCB: Dùng biến semaphore joinsem. Gọi joinsem->V()
3. `ExitWait()` ở lớp PCB: Dùng biến semaphore exitsem. Gọi exitsem->P()
4. `ExitRelease()` ở lớp PCB: Dùng biến semaphore exitsem. Gọi exitsem->V()
5. `JoinUpdate()` ở lớp PCB: Gọi JoinWait để dừng trạng thái và chờ JoinRelease để chạy tiếp. Sau khi chờ xong thì ta chạy tiếp ExitRelease để giải phóng process đó.
6. `ExitUpdate(int id)` ở lớp PTable: Dùng JoinRelease để trả lại những process đang chờ(process đã gọi hàm JoinWait). Sau đó dùng ExitWait để có thể tiến hành ExitRelease bên hàm JoinUpdate.

Join() syscall: Gọi `pTab->JoinUpdate(name)` ở trên
Exit() syscall: Gọi `pTab->ExitUpdate(name)` ở trên

3.3 Cài đặt System call CreateSemaphore

- Kiểm tra Semaphore có tên đó đã được tạo trong bảng bitmap chưa, nếu như rồi trả về -1. Còn không tiếp tục bước 2.
- Khởi tạo Semaphore bằng cách khởi tạo đối tượng Semaphore được cài đặt sẵn trong thư viện hệ đều hành NachOS (trong file synch.cc)

3.4 Cài đặt System call Wait , Signal

- Đầu tiên ta sẽ kiểm tra tên file có hợp lệ hay không (có là NULL không) nếu có trả về -1, không thì ta tiếp tuc

- Ta sẽ dùng `kernel->semTab->Wait(name)`hoặc `kernel->semTab->Signal(name)`(tùy thuộc vào việc gọi System call Wait hoặc Signal)để kiếm trong semTable có Semaphore tên name đã tồn tại chưa. Nếu có trả về 1. Còn nếu như không có Semaphore nào hợp lệ thì trả về -1

- Tiếp đó dựa vào việc ta muốn gọi System call Wait hay Signal, ta sẽ gọi hàm wait() hoặc signal () (Đây là 2 hàm được cài đặt sẵn trong stable.cc sẽ gọi hàm P() và V() trong file synch.cc)

3.5 Cài đặt System call Exec with argument

Ta sẽ cài đặt giống với lại System call Exec ở trên nhưng ta sẽ tạo một object arg từ struct ProcessArg để lưu lại các tham số của chương trình vào struct ProcessArg có cấu trúc như sau :

```
1       struct ProcessArg
2       {
3           int pid;
4           int argc;
5           char **argv;
6       };
7
```

Chương 4

Cài đặt chương trình Test

Chương trình thử nghiệm các System call, và các thuật toán để chạy đa chương đã được triển khai ở Chương 2 và Chương 3: Chương trình Ping-Pong.

4.1 Chương trình multiproc, thử nghiệm chạy đa chương

4.1.1 Giải thuật chương trình multiproc

- Khởi tạo `SpaceID newProc1, newProc2` để lưu lại ProcessID của 2 tiến trình chuẩn bị chạy.
- Gọi System call `Exec("proc01")`, `Exec("proc02")` để thực hiện tạo ra 2 tiến trình con, lần lượt là `proc01` và `proc02`. System call `Exec` khởi chạy một tiến trình mới và trả về ProcessID của tiến trình đó. Các ProcessID này sau đó được gán lần lượt cho các biến `newProc1` và `newProc2`.
- Chương trình sau đó gọi System call `Join` với `newProc1` và `newProc2` làm tham số. System call `Join` cho phép tiến trình chính chờ đợi tiến trình con kết thúc. Trong trường hợp này, tiến trình hiện tại sẽ chờ đợi cho đến khi `proc01` và `proc02` kết thúc.
- Chương trình gọi System call `Halt` để thực hiện tắt hệ điều hành.

4.1.2 Chương trình proc01

- Chương trình đơn giản tạo ra file `proc01.txt`, sử dụng System call `Create` đã được xây dựng ở đồ án 1.

4.1.3 Chương trình proc02

- Tương tự như chương trình `proc01`, tạo ra file `proc02.txt`, sử dụng System call `Create` đã được xây dựng ở đồ án 1.

4.2 Demo chương trình

```
m4pl3@M4PL3-DESKTOP:~/HDH/FinalLab/NachOS-4.0/code/test$ ../build.linux/nachos -x multiproc
Machine halting!

Ticks: total 484, idle 0, system 390, user 94
Disk I/O: reads 0, writes 0
Console I/O: reads 0, writes 0
Paging: faults 0
Network I/O: packets received 0, sent 0
m4pl3@M4PL3-DESKTOP:~/HDH/FinalLab/NachOS-4.0/code/test$ ls
DISK_0             concatenate.c     fileclienttest.txt      multiproc.coff  proc02.coff     sort.o
DayLaFileTest.txt  concatenate.coff  finalgame.txt           multiproc.o     proc02.o        start.S
Makefile           concatenate.o     finaltest.txt           open_file       proc02.txt      start.o
Makefile.dep       copy              halt                    open_file.c     read_file       stdarg.h
add                copy.c            halt.c                  open_file.coff  read_file.c     stdin
add.c              copy.coff         halt.coff               open_file.o     read_file.coff  stdio.c
add.coff           copy.o            halt.o                  outfile.txt     read_file.o     stdio.h
add.o              createfile        input.txt               output.txt      scheduler       stdlib.c
add1               createfile.c      lallalalalla.txt        ping            scheduler.c     stdlib.h
add1.c             createfile.coff   llllll.txt              ping.c          scheduler.coff  stdout
add1.coff          createfile.o      lllllllllllllllll.txt   ping.coff       scheduler.o     student
add1.o             delete            main                    ping.o          script          student.c
add2               delete.c          main.c                  pong            segments        student.coff
add2.c             delete.coff       main.coff               pong.c          segments.c      student.o
add2.coff          delete.o          main.o                  pong.coff       segments.coff   tcp
add2.o             echoclient        mamals.txt              pong.o          segments.o      tcp.coff
cat                echoclient.c      matmult                 proc01          shell           tcp.o
cat.c              echoclient.coff   matmult.c               proc01.c        shell.c         va-mips.h
cat.coff           echoclient.o      matmult.coff            proc01.coff     shell.coff      write_file
cat.o              fileclient        matmult.o               proc01.o        shell.o         write_file.c
cclc               fileclient.c      mmmmmmmmmm.txt          proc01.txt      sort            write_file.coff
clear              fileclient.coff   multiproc               proc02          sort.c          write_file.o
concatenate        fileclient.o      multiproc.c             proc02.c        sort.coff
m4pl3@M4PL3-DESKTOP:~/HDH/FinalLab/NachOS-4.0/code/test$
```

Hình 4.1: Kết quả sau khi chạy chương trình `multiproc`

Chương 5

Kết quả đạt được

Các yêu cầu của đồ án và kết quả đạt được của nhóm được tóm tắt trong bảng dưới đây:

Phần		**Mô tả**	**Mức độ hoàn thành**
1	1	Thay đổi mã nguồn cho các loại lỗi exceptions khác (không phải system call exceptions) để tiến trình hoàn thàn	**100%**
	2	Giải pháp đa chương: quản lý cấp phát và giải phóng vùng nhớ, quản lý cấp phát vùng dữ liệu, đồng bộ hóa.	**100%**
2	1	syscall Exec	**100%**
	2	syscall Join, Exit	**100%**
	3	syscall CreateSemaphore	**100%**
	4	syscall Wait, Signal	**100%**
	5	syscall Exec với tham số	**100%**
3		Kiểm thử chương trình	**100%**

Tài liệu tham khảo

[1] Github Repo: nachos - TrinhLongVu, https://github.com/TrinhLongVu/nachos.

[2] Github Repo: nachos-project - leduythuccs, https://github.com/leduythuccs/nachos-project.

[3] Nachos Project Guide - Jeff Chase, https://users.cs.duke.edu/~chase/nachos-guide/guide/nachos.htm#_Toc535602528.

[4] Echo server and client using sockets in c, https://mohsensy.github.io/programming/2019/09/25/echo-server-and-client-using-sockets-in-c.html.