ĐỒ ÁN 2 – HỆ ĐIỀU HÀNH

1. Mô tả

- Đồ án được làm theo nhóm: 3-5 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)
- 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.

NỘI DUNG ĐỒ ÁN

Phần 1. Hiểu mã chương trình nachos

Việc đầu tiên là đọc để hiểu nachos. Mã hiện nay ở mức 1 người dùng bằng chương trình C tại một thời điểm. Chúng tôi cung cấp các bạn một chương trình **halt** rất đơn giản để thử nghiệm nachos, chương trình halt làm là bắt HĐH tắt máy. Chạy chương trình "**nachos –rs 1234 –x ../test/halt**". Dò tìm khi chương trình người dùng nạp, chạy, và gọi một system call.

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**
- **openfile.h** định nghĩa các hàm trong hệ thống file nachos. Trong đồ án này chúng ta sử dụng lời gọi thao tác với file trực tiếp từ Linux, trong đồ án khác chúng ta sẽ triển khai hệ thống file trên ổ đĩa giả lập. (nếu kịp thời gian)
- **translate.*** Phiên bản nachos chúng tôi gửi các bạn, chúng tôi giả sử mỗi địa chỉ ảo là cũng giống hệt như địa chỉ vật lý, điều này giới hạn chúng ta chỉ chạy 1 chương trình tại một thời điểm. Các bạn có thể viết lại phần này để cho phép nhiều chương trình chạy
- cùng lúc trong đồ án sau.
- **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 l y' nhập xuất I/O theo dòng trong Nachos.
- **../test/*** Các chương trình C sẽ được biên dịch theo MIPS và chạy trong Nachos

Phần 2. Hiểu thiết kế

Để hiểu HĐH làm việc như thế nào, chúng ta cần phải hiểu và phân biệt được **kernel (system space)** và **user space**. Mỗi chương trình trong hệ thống phải có các thông tin cục bộ của nó, bao gồm program counters, registers, stack pointers, và file system handler. Mặc dù user program truy cập các thông tin cục bộ của nó, nhưng HĐH điều khiển các truy cập này, HĐH đảm bảo các yêu cầu từ user program tới kernel không làm cho HĐH sụp đổ. Việc chuyển quyền điều khiển từ user mode thành system mode được thực hiện thông qua system calls, softwave interrupt/trap. Trước khi gọi một lệnh trong hệ thống thì các tham số truyền vào cần thiết phải được nạp vào các thanh ghi của CPU. Để chuyển một biến mang giá trị, tiến trình chỉ việc ghi giá trị vào thanh ghi. Để chuyển một biến tham chiếu, thì giá trị lưu trong thanh ghi đc xem như là "user space pointer". Bởi vì user space pointer không có y nghĩa đối với kernel, mà chúng ta

cần là chuyển nội dung từ user space vào kernel sao cho ta có thể xử ly' dữ liệu này. Khi trả thông tin từ system về user space, thì các giá trị phải đặt trong các thanh ghi của CPU.

Nachos cung cấp một CPU giả lập, thực tế CPU giả lập này giống hệt CPU thật (MIPS chip), nhưng chúng ta không thể chỉ thực thi chương trình như một tiến trình bình thường của UNIX, bởi vị chúng ta muốn kiểm soát có bao nhiêu lệnh được thực hiện trong một đơn vị tgian, không gian địa chỉ làm việc như thế nào, các interrupt và exception(system calls) được xử lý như thế nào.

Nachos cung cấp môi trường giả lập để chạy các chương trình bằng C, xem Makefile trong thư mục test, chương trình biên dịch phải link với vài flag, sao đó chuyển sang định dạng đặc biệt của Nachos, dùng CT "coff2noff"

Phần 3. [85%] Exceptions và system calls

Tham Khảo:

- **[2] Giao tiep giua HDH Nachos va chuong trinh nguoi dung**
- **[3] Cach Viet Mot SystemCall**
- **[4] Cach Them 1 Lop Vao Nachos.**

Cài đặt các xử lý cho **exceptions** và các **system calls** cơ bản cho việc nhập xuất (số nguyên, ký tự …). Chú ý các bạn không nên thay đổi mã chương trình trong thư mục **machine**, chỉ được thay đổi mã CT bên trong thư mục **userprog**

a. Viết lại file exception.cc để xử ly' tất cả các exceptions được liệt kê trong *machine/machine.h*. **Hầu hết các exception trong này là run-time errors**, **khi các exception này xảy ra thì user program không thể được phục hồi.** Trường hợp đặc biệt duy nhất là ***no exception*** **sẽ trả quyền điều khiển về HĐH**, còn ***syscall exceptions*** **sẽ được xử ly' bởi các hàm chúng ta viết cho user system calls**. **Với tất cả exceptions khác, HĐH hiển thị ra một thông báo lỗi và Halt hệ thống**.

b. Viết lại cấu trúc điều khiển của chương trình để nhận các Nachos system calls. Kiểm tra cấu trúc mới bằng cách dùng system call **Halt** để kiểm tra tính đúng đắn của cấu trúc mới.

c. Tất cả các system calls (ko phải Halt) sẽ yêu cầu Nachos tăng program counter trước khi system call trả kết quả về. Nếu không lập trình đúng phần này thì Nachos sẽ bị vòng lặp gọi thực hiện system call này mãi mãi. Cũng như các hệ thống khác, MIPS xử ly' dựa trên giá trị của program counter, vì vậy bạn phải viết mã để tăng gia trị biến program counter, tìm đoạn mã này trong thư mục **machine.** Bạn phải copy mã này vào vị trí thích hợp trong phần xử ly' các system call của bạn. Hiện tại, bạn phải dùng **Halt system call** tại cuối mỗi user program.

d. Cài đặt system call **int ReadInt()**. **ReadInt** system call sẽ sử dụng lớp SynchConsole để đọc một số nguyên do người dùng nhập vào. Nếu giá trị người dùng nhập không phải là số nguyên thì trả về zero (0)

e. Cài đặt system call **void PrintInt(int number)**. **PrintInt** system call sẽ sử dụng lớp SynchConsole để xuất một số nguyên ra màn hình

f. Cài đặt system call **char ReadChar()**. **ReadChar** system call sẽ sử dụng lớp SynchConsole để đọc một ký tự do người dùng nhập vào.

g. Cài đặt system call **void PrintChar(char character)**. **PrintChar** system call sẽ sử dụng lớp SynchConsole để xuất một ký tự ra màn hình

h. Cài đặt đặt system call **void ReadString (char[] buffer, int length)**, **ReadString** system call sử dụng lớp SynchConsole dùng để đọc một chuỗi ký tự vào trong buffer (chuỗi sẽ kết thúc khi người dùng nhấn **enter,** hoặc có chiều dài lớn hơn hoặc bằng giá trị length (>= length)). Chú ý rằng buffer là vùng nhớ thuộc userspace, khi người dùng nhập chuỗi thì nội dung được lưu trữ ở kernel space bạn cần viết một hàm tương ứng để chuyển dữ liệu từ kernelspcae qua userspace

i. Cài đặt đặt system call **void PrintString (char[] buffer)**, **PrintString** system call dùng để in chuỗi ký tự trong buffer ra màn hình . Chú ý: tương tự như ReadString bạn cần phải có hàm để chuyển dữ liệu từ userspace qua kernelspace.

Chú ý

- 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)).
- Sau khi xử lý một system call bạn cần tăng PC lên, nếu không sẽ xảy ra hiện tượng loop (Chi tiết coi trong file HDTH: **[3]**).

j. Viết chương trình **help**, CT help dùng để in ra các dòng giới thiệu cơ bản về nhóm và mô tả vắn tắt về chương trình **sort** và **ascii**. (thật ra: chỉ việc gọi system call PrintString(char[])).
h.Viết chương trình **ascii** để in ra bảng mã ascii.
i. Viết chương trình **sort** với yêu cầu sau

- Cho phép người dùng nhập vào một mảng **n** số nguyên với **n** là số do người dùng nhập vào (n <=100)
- Sử dụng thuật toán bubble sort để sắp xếp mảng trên.

Phần 4. [15 %] Viết báo cáo
Bao gồm các comments bên trong chương trình, ngắn nhưng đầy đủ, và báo cáo trên giấy, mô tả bạn đã thiết kế và cài đặt như thế nào, tại sao làm như vậy. Mỗi nhóm cần phải demo đồ án của nhóm mình. Nhớ in phần báo cáo mang theo khi demo, 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.