

## Đề án 1. Exceptions và các system calls đơn giản

Hệ điều hành TH106

Ra đề ngày: 07.02.2007

Hạn cuối: 18.04.2007

*Cả hai lớp 2005-01 và 2005-02 đều có thời hạn nộp bài giống nhau, tuy nhiên thời gian chấm vẫn đáp sẽ thông báo sau*

### Download chương trình Nachos và hướng dẫn cài đặt:

ftp vào máy 172.29.64.138

user = hdh, không có password

Vào đường dẫn: /Download/TH106/Nachos

Download toàn bộ các files trong thư mục này.

Đề án đầu tiên giúp các bạn hiểu được mối quan hệ giữa HĐH và chương trình người dùng. Trong bài tập này các bạn phải viết một số system calls cơ bản. Trong nachos, bộ phận xử lý exception sẽ chịu trách nhiệm xử lý các system calls. Bạn phải viết xử lý cho các exception phát sinh lúc thực thi chương trình cũng như xử lý I/O. Chúng tôi sẽ cung cấp các bạn một số đoạn mã cơ bản, công việc của các bạn là lập trình để hoàn thành nó.

### 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 đề 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 đề án này chúng ta sử dụng lời gọi thao tác với file trực tiếp từ Linux, trong đề 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 đề 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 lý' 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, software 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 đó xem như là “user space pointer”. Bởi vì user space pointer không có ý nghĩa đối với kernel, mà chúng ta cần là chuyển nội dung từ user space vào kernel sao cho ta có thể xử lý' 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 vì chúng ta muốn kiểm soát có bao nhiêu lệnh được thực hiện trong một đơn vị thời gian, không gian địa chỉ làm việc như thế nào, các interrupt và exception(system calls) được xử lý' ntn.


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 IO system calls

Cài đặt các xử lý' cho **exceptions** và các **system calls** cơ bản cho việc nhập xuất files. (Tất cả các system calls được liệt kê trong syscall.h). Chúng tôi đã cung cấp các hàm bằng ngôn ngữ assembly để gọi các system call từ hàm của ngôn ngữ C. 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 lý' 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 lý 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 các dùng system call **Halt** để kiểm tra tính đúng đắn của cấu trúc mới
- c. Tất cả các system calls (không 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 lý dựa trên giá trị của program counter, vì vậy bạn phải viết mã để tăng giá 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 lý các system call của bạn. Hiện tại, bạn phải dùng **Halt system call** tại cuối mỗi user program.
-  d. Cài đặt system call **int** XXXXXXXXXX (**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
- e. 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 lý phù hợp trong kernel. Bạn sẽ phải dùng đối tượng filesystem trong thư mục **filesystems**. 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.
- f. 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.**

- g. 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.
-  h. Viết chương trình **createfile** để kiểm tra system call Create. Bạn sẽ dùng tên file cố định, hoặc cho người dùng nhập vào từ console nếu như phần console IO của bạn là chạy được
- i. Viết chương trình **help**, CT **help** là in ra các dòng giới thiệu các lệnh cơ bản (thật ra: chỉ việc gọi system call Write() vào standard output)
- j. Viết chương trình **echo**, mỗi khi nhập một dòng từ console thì console xuất lại dòng đó
- k. Viết chương trình **cat**, yêu cầu nhập vô filename, rồi hiển thị nội dung của file đó
- l. Viết chương trình **copy**, yêu cầu nhập tên file nguồn và file đích và thực hiện copy
- m. Viết chương trình **reverse**, yêu cầu nhập tên file nguồn và file đích, đọc nội dung file nguồn, đảo ngược nội dung và ghi vào file đích.
- n. Viết bất cứ chương trình nào khác mà bạn nghĩ là cần thiết để chứng minh giải pháp của bạn là đúng đắn. (mà chúng tôi chưa kiểm tra trong phần từ h tới n)

Chú ý: phần lớn chấm điểm các bạn là dựa vào tính đúng đắn của chương trình của mình. Khi chúng ta xây dựng một HĐH tốt thì nó phải đảm bảo sao cho khi user gọi system call thì không làm cho HĐH phải “sụp”. Ngữ cảnh xấu nhất là User program có thể phát sinh Nachos runtime exception, bạn cũng đã xử lý trong phần a.

#### **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. Chúng tôi sẽ thông báo cách nộp bài và thời gian vấn đáp cụ thể sau. Bài các bạn nộp sẽ chạy trên máy của chúng tôi, để đảm bảo là chương trình của các bạn chạy đúng ở môi trường thông thường.