Báo cáo đồ án 1 và 2

System Calls, Exceptions and Files **Nhóm `0xff`**

Hệ điều hành - 15CNTN *Nguyễn Thành Tân - 1512491*

Ngày nộp: 10/12/2017 *Hà Tấn Linh - 1512284*

1 Task Assignment for Project 1

Đồ án 1 được nhóm chia thành bốn phần công việc:

- Module 1: Thêm lớp `SynchConsole` vào `nachos` để sử dụng thao tác đọc/ghi từ màn hình console và xử lý các Run-time error exceptions.
- Module 2: Định nghĩa và cài đặt các System Calls như trong yêu cầu đồ án.
- Module 3: Viết các chương trình ứng dụng như trong yêu cầu đồ án, kiểm thử và sửa lỗi các System Calls nếu có.
- Module 4: Viết báo cáo.

Dựa theo phân chia như trên, nhóm quyết định bạn Nguyễn Thành Tân sẽ làm Module 1 và 2. Còn bạn Hà Tấn Linh làm Module 3 và 4. Cụ thể về cách nhóm đọc hiểu mã nguồn `nachos`, thiết kế và cài đặt các phần của đồ án được trình bày chi tiết trong phần tiếp theo.

2 Task Assignment for Project 2

Đồ án 2 được nhóm chia thành hai phần công việc:

- Module 1: Thêm các lớp cần thiết và cài đặt các System Call như trong yêu cầu đồ án.
- Module 2: Cài đặt các chương trình thử nghiệm các System Calls và viết báo cáo.

Dựa theo phân chia như trên, nhóm quyết định bạn Nguyễn Thành Tân sẽ làm Module 1. Còn bạn Hà Tấn Linh làm Module 2. Thiết kế và cài đặt các phần của đồ án được trình bày chi tiết trong phần tiếp theo.

3 Exceptions

Hàm `ExceptionHandler()` bên trong file `code/userprog/exception.cc` nhận vào một tham số `which` có kiểu `ExceptionType`. Kiểu dữ liệu `ExceptionType` được định nghĩa trong file `code/machine/machine.h` là một kiểu `enum` định nghĩa các loại exceptions khác nhau có thể xảy ra trong quá trình thực thi của hệ điều hành. Mỗi khi được gọi, `ExceptionHandler()` sẽ dựa vào tham số `which` để có thể biết được loại exception nào đã xảy ra và có những cách giải quyết phù hợp.

Run-time errors

Run-time errors là những lỗi xảy ra do người lập trình ứng dụng vi phạm những quy định do người cài đặt hệ điều hành đặt ra. Run-time errors khi đã xảy ra thì việc khắc phục để chạy tiếp chương trình là không khả thi và cách khắc phục tốt nhất là thông báo lỗi cho người dùng và kết thúc chương trình một cách êm đẹp. Run-time errors có thể là Unchecked Run-time errors hoặc Checked Run-time errors.
Những Checked Run-time errors được định nghĩa trong `code/machine/machine.h` gồm: `IllegalInstr`, `ReadOnly`, `BusError`, `AddressError`, `Overflow`, `PageFaultException` và `NumExceptionTypes`.
Với mỗi loại exception được định nghĩa, `ExceptionHandler()` in ra màn hình một thông báo vắn tắt về Run-time error đã xảy ra và sau đó gọi hàm `Interrupt::Halt()` để kết thúc phiên làm việc của hệ điều hành.

No exception

Đối với trường hợp đặc biệt No exception, `ExceptionHandler()` sẽ trả điều khiển lại cho chương trình ứng dụng ngay lập tức. Tuy nhiên, để chương trình không bị lặp vô hạn, `ExceptionHandler()` phải thực hiện tăng giá trị thanh ghi `$PC` trước khi trả về bằng lệnh `return`.

Syscall exceptions

Syscall exception là một dạng exception đặc biệt đặc tả một dịch vụ mà hệ điều hành cung cấp cho người lập trình sử dụng. Nếu giá trị của tham số `which` là `SyscallException` thì thanh ghi `$r2` lúc này chứa giá trị định nghĩa loại System Call mà người lập trình yêu cầu từ hệ điều hành và có thể đọc được thông qua `Machine::ReadRegister(2)`.
Cũng giống như trường hợp No exception ở trên, trước khi hệ điều hành trả luồng điều khiển lại cho chương trình, thì `ExceptionHandler()` cần phải thực hiện thao tác tăng giá trị thanh ghi `$PC` lên để tránh trường hợp exception lặp vô hạn. Cụ thể chi tiết cách nhóm khai báo và cài đặt các System Calls sẽ được trình bày trong phần tiếp theo của báo cáo.

4 System Calls

Để khai báo một System Call trong `nachos`, đầu tiên ta cần khai báo tên của System Call, gán cho tên vừa đặt một giá trị (không trùng lặp với các System Calls đã được định nghĩa trước đó) và khai báo function prototype (API - Application Programming Interface) cho System Call bên trong file `code/userprog/syscall.h`. Mục đích của việc khai báo này là để người lập trình ứng dụng có thể `#include <syscall.h>` vào trong chương trình của mình và sử dụng các System Call APIs đã được khai báo tại đây.
Sau đó, với mỗi System Call được khai báo, ta cần thêm 2 global entries tương ứng vào các file `code/test/start.c` và `code/test/start.s`. Mỗi entry sẽ thực hiện gán giá trị tương ứng với loại System Call cho thanh ghi `$r2` rồi gọi lệnh `syscall` của MIPS. Mục đích của việc này là cài đặt xử lý cho các System Call APIs mà ta đã khai báo ở trên. Sau lệnh gọi `syscall`, một exception sẽ được phát sinh và luồng điều khiển được chuyển

từ User space về cho hàm `ExceptionHandler()` ở Kernel space để tiếp tục xử lý.
Lúc này, hệ thống đang ở Kernel space và sẵn sàng để xử lý các System Calls. Tuy nhiên, để có thể cài đặt được các System Calls như trong đồ án yêu cầu, ta cần phải thêm lớp `SynchConsole` vào `nachos` nhằm mục đích thực hiện các thao tác đọc/ghi với màn hình Console. Để thêm lớp `SynchConsole`, đầu tiên ta sao chép hai tập tin `synchcons.cc` và `synchcons.h` được cung cấp sẵn vào thư mục `code/threads/`. Một lưu ý đó là để tránh trường hợp quá trình tiền xử lý mã nguồn gặp lỗi khi biên dịch, ta cần phải thêm include guard vào file `synchcons.h`. Sau đó, ta cần khai báo và khởi tạo một đối tượng `gSynchConsole` bên trong các file `code/threads/system.h` và `code/threads/system.cc` để ta có thể dùng đối tượng này thực hiện thao tác đọc/ghi với màn hình console bên trong hàm `ExceptionHandler()`. Cuối cùng, ta cần chỉnh sửa `code/Makefile.common` để `nachos` có thể nhận ra và biên dịch lớp `SynchConsole`.

Syscall: ReadInt

Đầu tiên, System Call `ReadInt` dùng hàm `SynchConsole::Read()` để đọc dữ liệu từ console.
Sau đó `ReadInt` xác định vị trí bắt đầu của dữ liệu người dùng nhập vào (bỏ qua tất cả khoảng trắng và tabs). Nếu kí tự đầu tiên là dấu trừ thì đánh dấu là số âm. Sau đó lặp qua tất cả các kí tự còn lại. Nếu là số thì cộng vào giá trị `number`, ngược lại nếu gặp kí tự không phải số thì đánh dấu người dùng nhập vào không phải là một số hợp lệ và nhảy khỏi vòng lặp ngay lập tức.
Cuối cùng, nếu người dùng nhập vào số hợp lệ thì `ReadInt` ghi giá trị `number` vào thanh ghi `$r2` và trả về. Nếu không phải là số hợp lệ thì ghi vào thanh ghi `$r2` giá trị 0 (nhập lỗi) và trả về.

Syscall: PrintInt

Giá trị cần in ra màn hình được lấy từ thanh ghi `$r4`. System Call `PrintInt` tạo một chuỗi để biểu diễn số cần in và lặp để chuyển giá trị từ thành ghi `$r4` thành chuỗi sau đó gọi hàm `SynchConsole::Write()` để in ra màn hình.

Syscall: ReadChar

System Call `ReadChar` dùng hàm `SynchConsole::Read()` của lớp `SynchConsole` để đọc vào một chuỗi từ màn hình console và trả về kí tự cuối cùng trong chuỗi ấy như là kí tự mà người dùng nhập vào thông qua thanh ghi `$r2`.

Syscall: PrintChar

Tương tự như System Call `PrintInt`, `PrintChar` chỉ cần đọc kí tự cần in ra từ thanh ghi `$r4` và in ra màn hình console thông qua lời gọi hàm `gSynchConsole->Write()` của lớp `SynchConsole`.

Syscall: ReadString

System Call `ReadString` nhận vào địa chỉ vùng nhớ của người dùng từ thanh ghi `$r4` và số kí tự tối đa người dùng cần đọc từ thanh ghi `$r5`. Sau đó, `ReadString` sẽ tạo một vùng nhớ ở Kernel space và dùng hàm `SynchConsole::Read()` để đọc từ console vào vùng nhớ này. Cuối cùng, `ReadString` sẽ gọi hàm `System2User` để sao chép dữ liệu từ Kernel space vào User space trước khi trả luồng điều khiển về cho người dùng.

Syscall: PrintString

Đầu tiên, System Call `PrintString` lấy giá trị vùng nhớ chứa chuỗi cần in ra của người dùng từ thanh ghi `$r4`. Sau đó `PrintString` gọi hàm `User2System`, hàm này tạo ra một vùng nhớ ở Kernel space và sao chép dữ liệu từ User space vào đây. Lúc này `PrintString` sẽ gọi hàm `SynchConsole::Write()` để in chuỗi kí tự từ Kernel space ra console.

Đối với các System Calls của Dồ án 2, nhóm đặt tên System Calls khác biệt đôi chút so với yêu cầu đồ án vì các tên System Calls như trong yêu cầu đồ án đã được sử dụng trước đó trong nachos.

Syscall: CreateFile

Đầu tiên, `CreateFile` sao chép chuỗi tên file cần tạo được trỏ tới bởi giá trị trong thanh ghi `r4` vào một vùng nhớ ở kernel space. Sau đó `CreateFile` sẽ gọi hàm `Create` của đối tượng `fileSystem` để tạo file và trả về `-1` vào thanh ghi `r2` nếu tạo file thất bại hoặc trả về `0` nếu thành công.

Syscall: OpenFileFunc

`OpenFileFunc` đầu tiên chuyển chuỗi biểu diễn tên tập tin từ user space sang kernel space. Sau đó gọi hàm `Open` của đối tượng fileSystem và thêm phần tử nhận được vào mảng `openf`, cuối cùng trả về giá trị tương ứng cho người dùng.

Syscall: CloseFile

System Call `CloseFile` nhận tham số từ thanh ghi `r4` là file descriptor đối với người dùng và là chỉ số của một phần tử của mạng `openf` của đối tượng `fileSystem`. `CloseFile` kiểm tra giá trị của phần tử này, nếu nó trỏ tới `NULL` thì bỏ qua, còn ngược lại thì giải phóng bộ nhớ của phần tử và trỏ nó về `NULL`.

Syscall: ReadFile

Đầu tiên, `ReadFile` sẽ lấy tham số từ người dùng, kiểm tra xem file đã mở hay chưa. Nếu người dùng muốn đọc từ `stdin` thì `ReadFile` sẽ dùng hàm `SynchConsole::Read()` để đọc dữ liệu từ màn hình console rồi dùng hàm `System2User` để chuyển dữ liệu từ kernel space sang vùng nhớ của người dùng, hoặc nếu người dùng yêu cầu đọc từ một file, `ReadFile` sẽ dùng hàm `Read` và `GetCurrentPos` để đọc từ file và tính số bytes đã đọc được để trả về cho người dùng.

Syscall: WriteFile

Ngược lại với `ReadFile`, System Call `WriteFile` sẽ đọc tham số từ người dùng, chuyển vùng nhớ từ user space sang kernel space. Cuối cùng tuỳ theo người dùng muốn viết vào `stdout` hay vào file mà `WriteFile` sẽ dùng hàm của `Write` của `SynchConsole` hoặc tổ hợp 2 hàm `Write` và `GetCurrentPos` của file system.

Syscall: SeekFile

System Call `SeekFile` đầu tiên sẽ lấy tham số từ người dùng từ 2 thanh ghi `r4` và `5`. Sau đó `SeekFile` sẽ kiểm tra file descriptor và vị trí nhảy tới có hợp lệ hay không, nếu không thì trả về `-1`. Cuối cùng `SeekFile` sẽ gọi hàm `Seek` của phần tử tương ứng trong mảng `openf` của đối tượng `fileSystem` để nảy tới vị trị mong muốn và trả về kết quả tương ứng.

Ngoài các System Calls ở trên, ta cũng cần phải định nghĩa thêm System Call `Exit`. Khi bất cứ chương trình nào chạy trên `nachos` kết thúc thì System Call `Exit` đều được gọi. Chính vì thế ta cần cài đặt System Call này để tránh trường hợp System Call `Exit` được xử lý như một Unexpected System Call (in ra thông tin không cần thiết ở cuối mỗi chương trình).

5 User Programs

Các chương trình ứng dụng của `nachos` mặc định được đặt trong thư mục `code/test/`. Sau khi viết chương trình cho `nachos`, để có thể biên dịch và thực thi chương trình, ta cần phải chỉnh sửa kịch bản biên dịch trong file `code/test/Makefile`.
Cụ thể, đầu tiên ta cần phải thay đổi giá trị của `$GCCDIR` và `$CC` để chúng trỏ đến pre-built gcc directory. Có thể sử dụng đường dẫn trực tiếp đối với hai giá trị này, tuy nhiên cần phải sửa chữa Makefile mỗi lần ta di chuyển thư mục `nachos`. Cách tốt hơn là sử dụng đường dẫn tương đối cho `$GCCDIR` và `$CC`. Với thư mục `gnu-decstation-ultrix` và `nachos` cùng nằm trong một thư mục cha, ta có thể sử dụng đường dẫn tương đối dạng `"../../../gnu-decstation-ultrix..."`.
Tiếp theo, với mỗi chương trình (ví dụ tên `abc`) muốn được biên dịch, ta phải tạo hai luật biên dịch (rules) `abc.o` và `abc` vào Makefile. Luật `abc.o` dùng compiler được định nghĩa trong `$CC` để biên dịch file mã nguồn thành object file. Luật `abc` đầu tiên dùng linker được định nghĩa trong `$LD` để liên kết `abc.o` và `start.o` thành một file thực thi định dạng COFF. Sau đó, nó dùng cross-compiler `code/bin/coff2noff` để chuyển file thực thi sang định dạng NOFF chạy được trên `nachos`.
Cuối cùng, ta cần phải thêm tên chương trình vào luật `all` để câu lệnh `make all` có thể hiểu và biên dịch luôn cả chương trình mới được thêm vào.

code/test/help

Chương trình `help` in ra thông tin của nhóm và giới thiệu tóm tắt về chương trình `ascii` và `sort`.
Help dùng Syscall `PrintString()` để in các cuỗi ra màn hình console. `PrintString()`

nhận tham số đầu vào là các string literals được lưu trong bộ nhớ của chương trình ở User space. Sau đó `PrintString()` sao chép chuỗi đầu vào sang Kernel space và dùng hàm `SynchConsole::Write()` để in ra màn hình console.

code/test/ascii

Chương trình `ascii` in ra màn hình console 128 kí tự của bảng mã ASCII tiêu chuẩn.
`ascii` in ra bảng mã ASCII lần lượt theo thứ tự từ nhỏ đến lớn, 2 kí tự trên một dòng. Nếu kí tự có thể in ra được (printable characters) thì trực tiếp in kí tự đó ra màn hình. Ngược lại nếu kí tự đó là mã điều kiển (control codes) hoặc kí tự khác không in ra màn hình được thì in một dòng diễn tả vắn tắt công dụng của kí tự đó. Mỗi kí tự hoặc chú thích (nếu kí tự không in ra màn hình được) chiếm đúng 36 cột của màn hình cosole.
Đầu tiên, `ascii` dùng syscall `PrintString()` để in chuỗi `"ASCII Table"` ra màn hình console.
Lần lượt lặp với kí tự có giá trị lần lượt từ `0` đến `127`. Với mỗi kí tự, in ra số thứ tự của nó (thêm số 0 vào đầu để đảm bảo các giá trị số thứ tự chiếm độ rộng như nhau trên màn hình console). Sau đó, gọi hàm `get_char_description()` với hai tham số là giá trị của kí tự và một con trỏ kiểu `unsigned int *`. Cuối cùng, dựa vào giá trị trả về nhận được để in ra màn hình thông tin kí tự tương ứng.
Cụ thể, hàm `get_char_description()` sẽ kiểm tra giá trị của kí tự nó nhận được. Nếu kí tự là kí tự in được (printable characters) thì `get_char_description()` sẽ trả về NULL. Ngược lại, nếu kí tự là mã điều khiển (control codes) thì nó sẽ gọi hàm `ctrl_code_description()` để lấy chuỗi thông tin về mã điều khiển và trả về cho chương trình chính. Ngoài ra, `get_char_description()` cũng xử lý hai trường hợp đặc biệt là kí tự Space (`0x20`) và kí tự Delete (`0x7f`).

code/test/sort

Chương trình `sort` yêu cầu người dùng nhập vào một số nguyên `N` nhỏ hơn hoặc bằng 100. Sau đó, nó sẽ lại tiếp tục yêu cầu người dùng nhập vào `N` số nguyên. Với mỗi thao tác nhập, nếu người dùng nhập sai (syscall `ReadInt()` trả về 0) thì chương trình sẽ yêu cầu người dùng nhập lại số nguyên đó cho đến khi nhập đúng.
Sau khi nhập đủ các số nguyên hợp lệ, `sort` dùng thuật toán Bubble Sort sắp xếp lại dãy số theo thứ tự tăng dần.
Cuối cùng, `sort` dùng kết hợp 2 Syscall `PrintInt()` và `PrintString()` để in ra màn hình console dãy số đã được sắp xếp, mỗi số cách nhau một dấu phẩy.

code/test/createfile

Chương trình `createfile` yêu cầu người dùng nhập vào một tên file từ console bằng cách gọi System Call `PrintString` để in thông điệp và `ReadFile` với tham số thứ 3 bằng `0` để chờ người dùng nhập dữ liệu vào từ `stdin`. Cuối cùng sẽ gọi System Call `CreateFile` để thực hiện tạo file như yêu cầu của người dùng và thông báo kết quả tạo file tương ứng ra console.

code/test/echo

Chương trình `echo` chờ người dùng nhập vào một dòng kí tự từ console, đọc nó bằng System Call `ReadFile` và ghi lại chính nó ra console bằng System Call `WriteFile`.

code/test/cat

Chương trình `cat` yêu cầu người dùng nhập tên tập tin từ console, đọc tên file bằng System Call `ReadFile`. Sau đó chương trình sẽ gọi System Call `OpenFileFunc` để mở tập tin này để đọc. Cuối cùng chương trình đọc lần lượt mỗi lần 256 bytes từ file và ghi ra console dữ liệu tương ứng bằng System Call `WriteFile`.

code/test/copy

Chương trình `copy` yêu cầu người dùng lần lượt nhập tên file nguồn và file đích cần sao chép. Sau đó chương trình sẽ gọi System Call `OpenFileFunc` lần lượt cho 2 file này để mở file, đặc biệt phải gọi System Call `CreateFile` đối với file đích để tạo nó trước khi mở nếu nó chưa tồn tại. Cuối cùng chương trình lần lượt đọc một lần 256 bytes từ file nguồn và ghi đúng số bytes đã đọc được tại lượt đó ra file đích.

code/test/reverse

Chương trình `copy` yêu cầu người dùng lần lượt nhập tên file nguồn và file đích cần đảo ngược. Sau đó chương trình sẽ gọi System Call `OpenFileFunc` lần lượt cho 2 file này để mở file, đặc biệt phải gọi System Call `CreateFile` đối với file đích để tạo nó trước khi mở nếu nó chưa tồn tại. Sau đó, chương trình sẽ gọi System Call `SeekFile` với tham số vị trí `-1` để lấy kích thước file nguồn. Cuối cùng, mỗi lần chương trình sẽ dùng hai System Calls `SeekFile` và `ReadFile` đọc một byte từ cuối file nguồn trở về đầu file và ghi lần lượt ra file đích.

code/test/hexdump

Đối với yêu cầu 10 của Đồ án 2, nhóm quyết định viết chương trình Hex dump thực hiện việc in ra bytecode của một tập tin để minh hoạ cho phần cài đặt các System Calls về File System.
Chương trình `hexdump` đầu tiên yêu cầu người dùng nhập vào tên của tập tin cần dump. Sau đó chương trình gọi System Call `OpenFileFunc` để mở tập tin đó lên, đọc lần lượt mỗi lần 256 bytes. Cuối cùng, với mỗi byte đọc được, chương trình sẽ chuyển sang mã thập lục phân và in ra màn hình console, với số lượng mỗi dòng là 16 bytes.

Summary

Trên đây là báo cáo của nhóm `0xff` về việc phân công, thiết kế cũng như cài đặt các System Calls, Exceptions và các System Calls về hệ thống tập tin và nhập xuất với tập tin dựa theo yêu cầu đồ án. Cảm ơn thầy/cô đã dành thời gian đọc bản báo cáo này của nhóm.