TRƯỜNG ĐẠI HỌC KHOA HỌC TỰ NHIÊN
KHTN
ĐHQG-HCM

fit@hcmus

Project 02

HỆ ĐIỀU HÀNH

Các system calls thao tác với files

Giảng viên hướng dẫn

Giảng viên lý thuyết	Lê Quốc Hòa
Giảng viên thực hành	Chung Thùy Linh

Thông tin nhóm

19127292	Nguyễn Thanh Tình
19127339	Phạm Chi Bảo

Thành phố Hồ Chí Minh, tháng 4 năm 2022

Lời cảm ơn

Hệ điều hành là thành phần quan trọng không thể thiếu trong hầu hết tất cả các thiết bị điện tử hiện nay. Nghiên cứu hệ điều hành sẽ giúp chúng ta có cái nhìn chuyên sâu về cách hoạt động của chúng. Tuy nhiên, đối với người mới bắt đầu, việc tìm hiểu những hệ điều hành mới và phổ biến nhất hiện nay sẽ rất khó khăn và tốn nhiều thời gian hơn. Vì lý do đó, **NachOS** ra đời là một hệ điều hành đơn giản để tìm hiểu và nghiên cứu, cung cấp kiến thức cơ bản để xây dựng các thành phần phức tạp hơn của các hệ điều hành sau này.

Chân thành cảm ơn thầy **Lê Quốc Hòa** và cô **Chung Thùy Linh** đã cung cấp cho cả lớp nói chung và nhóm chúng em nói riêng **Project 02 - Các system calls thao tác với files** - một đồ án thật sự rất bổ ích. Và cũng chân thành cảm ơn cô **Chung Thùy Linh** một lần nữa vì những tài liệu và những buổi seminar hướng dẫn chi tiết đầy hữu ích từ cô, giúp chúng em hoàn thành được đồ án này.

Thành phố Hồ Chí Minh, ngày 19 tháng 4 năm 2022

Tập thể nhóm

Mục lục

Phần 1. Cài đặt system calls thao tác với files

Quy trình thêm một system call

Quy trình thêm một system call sẽ bao gồm các bước sau đây:

- Trong tập tin userprog/syscall.h, thêm prototype cho syscall, đồng thời define mã cho syscall này.
- Trong tập tin test/start.s, thêm một đoạn code nạp mã syscall này vào thanh ghi $2 và thực hiện gọi syscall, sau đó nhảy đến thanh ghi $31. Hoặc đơn giản hơn, copy một đoạn lệnh có sẵn cho một system call và chỉnh sửa lại tên.
- Trong tập tin userprog/exception.cc, thêm phần xử lý system call vào switch (type). Cấu trúc của đoạn mã cho system call này sẽ như sau:
  - Thông báo DEBUG là thực hiện system call.
  - Cài đặt xử lý system call.
  - Chuẩn bị kết quả system call (nếu cần).
  - Tăng Program Counter, sau đó return.
  - Đánh dấu ASSERTNOTREADCHED().
  - Break system call.
- Sau khi thực hiện xong, biên dịch lại NachOS.

1. System call: Create

Cài đặt system call **int Create(char* name)**.

System call Create sử dụng NachOS **FileSystemObject** để tạo một file rỗng. Vì filename đang ở trong userspace (buffer mà con trỏ đang trỏ trong user space), nên ta cần chuyển nó vào vùng nhớ kernel.

Người dùng có thể gây sụp hệ điều hành nếu:

- Tên file quá lớn (giải quyết: đặt giới hạn ký tự).
- Tên file rỗng (giải quyết: trả về kết quả là thất bại).

Syscall trả về 0 khi thành công và -1 khi thất bại.

Ta làm các bước như sau:

- Chuyển tên file từ User space sang Kernel Space.
- Nếu tên file không hợp lệ, ghi -1 vào thanh ghi và dừng system call.
- Tạo file bằng **kernel->fileSystem->Create(filename)**.

2. System call: Open

Cài đặt system call **OpenFileID Open(char* name)**.

System call Open sẽ mở một file bằng **kernel->fileSystem->Open(filename)**, và lưu trữ nó vào một mảng (hỗ trợ mở nhiều file) và trả về OpenFileID là index.

Người dùng có thể gây sụp hệ điều hành nếu:

- Tên file quá lớn (giải quyết: đặt giới hạn ký tự).
- Tên file rỗng (giải quyết: trả về kết quả là thất bại).
- Mở quá nhiều file (giải quyết: đặt giới hạn số lượng file mở).

Vậy, để giải quyết bài toán này, ta cần làm các bước sau đây:

- Đọc địa chỉ ảo của filename, dùng User2System để copy filename.
- Tìm slot trống trong bảng OpenFile.
- Nếu không còn slot trống, trả về kết quả là thất bại.
- Nếu còn slot trống, thử mở file.
- Nếu mở thành công, trả về vị trí của file đã mở trong bảng thông tin file.
- Nếu mở thất bại, trả về kết quả -1.
- Xóa buffer filename.

3. System call: Close

Cài đặt system call **int Close(OpenFileID id)**.

System call Close sẽ đóng một file có OpenFileID id bằng cách xóa khỏi bảng lưu trữ file đang mở của hệ điều hành.

Người dùng có thể gây sụp hệ điều hành nếu:

- Truyền vào OpenFileID nằm ngoài bảng thông tin lưu trữ file (giải quyết: kiểm tra điều kiện về OpenFileID trước khi thực hiện).
- Đóng một file chưa mở (giải quyết: kiểm tra đã mở chưa).

Vậy, để cài đặt được system call này, ta thực hiện như sau:

- Đọc OpenFileID từ thanh ghi.
- Kiểm tra điều kiện OpenFileID có nằm trong ngưỡng của bảng lưu trữ thông tin file hay không?
- Kiểm tra điều kiện file đã mở hay chưa.
- Delete đối tượng file.
- Gán NULL cho vị trí OpenFileID id.
- Trả về kết quả là thành công.

4. System call: Read

Cài đặt system call **int Read(char *buffer, int size, OpenFileID id)**.

System call Read sẽ sử dụng **kernel->fileSystem->openFile[id]->Read()** để đọc dữ liệu từ file, và trả về là số ký tự thực sự đã đọc (-1 cho việc đọc thất bại).

Người dùng có thể gây sụp hệ điều hành nếu:

- Truyền OpenFileID không hợp lệ (giải quyết: kiểm tra hợp lệ OpenFileID).
- Đọc trên Console Out (giải quyết: thông báo thất bại).

Vậy, để giải quyết bài toán này, ta cần làm các bước sau đây:

- Đọc vị trí của buffer, số lượng ký tự cần đọc và OpenFileID từ thanh ghi.
- Kiểm tra tính hợp lệ của OpenFileID id.
- Kiểm tra xem có đang đọc trên Console Out không, nếu có, thông báo thất bại và dừng việc đọc.
- Nếu đọc trên Console In, sử dụng lớp **SynchConsoleIn** để đọc.
- Nếu là file thường, sử dụng **kernel->fileSystem->openFile[id]->Read()** để đọc dữ liệu.
- Dùng System2User để đưa kết quả cho người dùng, kết thúc syscall.

5. System call: Write

Cài đặt system call **int Write(char *buffer, int size, OpenFileID id)**.

System call Write sẽ sử dụng **kernel->fileSystem->openFile[id]->Write()** để ghi dữ liệu ra file, và trả về là số ký tự thực sự đã ghi (-1 cho việc ghi thất bại).

Người dùng có thể gây sụp hệ điều hành nếu:

- Truyền OpenFileID không hợp lệ (giải quyết: kiểm tra hợp lệ OpenFileID).
- Ghi trên Console In (giải quyết: thông báo thất bại).

Vậy, để giải quyết bài toán này, ta cần làm các bước sau đây:

- Đọc vị trí của buffer, số lượng ký tự cần ghi và OpenFileID từ thanh ghi.
- Kiểm tra tính hợp lệ của OpenFileID id.
- Kiểm tra xem có đang ghi trên Console In không, nếu có, thông báo thất bại và dừng việc ghi.
- Nếu ghi trên Console Out, sử dụng lớp **SynchConsoleOut** để ghi.
- Dùng User2System để lấy chuỗi cần ghi.
- Nếu là file thường, sử dụng **kernel->fileSystem->openFile[id]->Write()** để ghi dữ liệu.

6. System call: Seek

Cài đặt system call **int Seek(int position, OpenFileID id)**.

System call Seek sẽ chuyển con trỏ tới vị trí thích hợp (vị trí đó là position, nếu position = -1 thì chuyển đến cuối file) và trả về vị trí thật sự trong file nếu thành công và -1 nếu bị lỗi. Gọi Seek trên console cũng sẽ báo lỗi.

Người dùng có thể gây sụp hệ điều hành nếu:

- OpenFileID id không hợp lệ (giải quyết: kiểm tra tính hợp lệ của id).
- Seek trên Console (giải quyết: thông báo thất bại).
- Seek tới vị trí không phù hợp (position <= -2 hoặc position lớn hơn vị trí cuối cùng của file) (giải quyết: kiểm tra tính hợp lệ của position).

Vậy, để giải quyết bài toán này, ta cần làm các bước sau đây:

- Đọc vị trí cần seek và OpenFileID id.
- Kiểm tra tính hợp lệ của OpenFileID id.
- Kiểm tra xem có đang seek trên console không.
- Seek đến vị trí position bằng **kernel->fileSystem->openFile[id]->Seek()**.
- Lưu vị trí thực vào thanh ghi, kết thúc syscall.

7. System call: Remove

Cài đặt system call **int Remove(char* name)**.

System call Remove sẽ sử dụng NachOS FileSystemObject để xóa file. Nếu file đang mở, hệ thống sẽ không được phép xóa.

Người dùng có thể gây sụp hệ điều hành nếu:

- Tên file quá dài (giải quyết: đặt giới hạn số lượng ký tự).
- Tên file rỗng (giải quyết: thông báo thất bại).
- Xóa một file đang mở (giải quyết: vì các file mở bằng **kernel->fileSystem ->Open()** đều mở mặc định dưới dạng đọc và ghi, nên ta có thể kiểm tra rằng file có mở hay không bằng cách mở lại lần nữa. Nếu mở lại không thành công, file đang được mở bởi tiến trình nào đó).

Vậy, để cài đặt được system call này, ta thực hiện như sau:

- Đọc vị trí của filename, lấy filename bằng User2System.
- Kiểm tra hợp lệ của tên file.
- Kiểm tra file đang mở hay không.
- Xóa file bằng **kernel->fileSystem->Remove()**, thông báo thành công.

8. Sao chép vùng nhớ kernel space và user space

Kernel space to User space

Buffer là vùng nhớ thuộc **kernel space**, khi người dùng nhập chuỗi thì nội dung được lưu trữ ở **kernel space**, chúng ta cần viết một hàm tương ứng để chuyển dữ liệu từ **kernel space** qua **user space**.

Công cụ đắc lực để hỗ trợ thao tác này là: **kernel->machine->WriteMem()**.

Để thực hiện được công việc này, nhóm đã tham khảo đoạn code cài đặt hàm **System2User** trong **[2] Giao tiep giua Nachos và chuong trinh nguoi dung.pdf**, và ý tưởng của hàm này là:

- Kiểm tra độ dài buffer, nếu buffer có độ dài nhỏ hơn hoặc bằng 0 thì không cần thực hiện tiếp.
- Tạo một vòng lặp duyệt buffer (vòng lặp dừng khi buffer được đọc hết hoặc đọc tới ký tự '\0')
  - Lấy ký tự từ trong buffer ra.
  - Dùng **kernel->machine->WriteMem()** để ghi vào user space.
- Trả về kết quả là số byte thực tế đã copy.

User space to Kernel space

Chuỗi là vùng nhớ thuộc **user space**, khi người dùng muốn in chuỗi thì nội dung được lưu trữ ở buffer trong **kernel space**, chúng ta cần viết một hàm tương ứng để chuyển dữ liệu từ **kernel space** qua **user space**.

Công cụ đắc lực để hỗ trợ thao tác này là: **kernel->machine->ReadMem()**.

Để thực hiện được công việc này, nhóm đã tham khảo đoạn code cài đặt hàm **User2System** trong **[2] Giao tiep giua Nachos và chuong trinh nguoi dung.pdf**, và ý tưởng của hàm này là:

- Kiểm tra độ dài giới hạn buffer, nếu giới hạn buffer có độ dài nhỏ hơn hoặc bằng 0 thì không cần thực hiện tiếp.
- Tạo một vòng lặp duyệt chuỗi ở mức người dùng (vòng lặp dừng khi chuỗi được đọc hết hoặc đọc tới ký tự '\0')
  - Lấy ký tự từ trong chuỗi ra.
  - Dùng **kernel->machine->WriteMem()** ghi vào buffer system space.
- Trả về địa chỉ của buffer cho system.

Phần 2. Viết chương trình người dùng

Quy trình viết chương trình người dùng

Quy trình viết chương trình người dùng bao gồm các bước sau:

- Viết chương trình người dùng bằng ngôn ngữ C trong thư mục test. Lưu ý: cần include thư viện syscall.h, đồng thời, ở cuối mỗi chương trình cần Halt() hệ thống.
- Chỉnh sửa Makefile:
  - Thêm chỉ thị trong tab **all**.
  - Thêm chỉ thị biên dịch mã nguồn từ ngôn ngữ C thành file object, sau đó kết hợp với file start.o để tạo thành file coff, cuối cùng dùng tiện ích coff2noff để biên dịch thành file noff.
- Trong thư mục test, gõ **make all**.
- Chạy chương trình ở mức người dùng.

1. Chương trình người dùng: createfile

Viết chương trình **createfile** để kiểm tra system call Create. Chương trình sẽ cho người dùng nhập vào tên file và thực hiện việc tạo file, và cũng thông báo nếu tạo file thất bại.

Chương trình này cài đặt rất dễ dàng, bao gồm các bước sau đây:

- Khai báo mảng tên file, nhập tên file.
- Tạo file. Thành công hay thất bại đều thông báo ra màn hình.

2. Chương trình người dùng: cat

Viết chương trình **cat** với để hiển thị nội dung file với tên file được nhập trước.

Chương trình này cài đặt rất dễ dàng, bao gồm các bước sau đây:

- Khai báo mảng tên file, biến kích thước, biến chạy, biến OpenFileID.
- Mở file.
- Nếu mở file thất bại, thông báo cho người dùng.
- Seek đến cuối file để lấy độ dài file (sau đó seek lại để không ảnh hưởng việc đọc).
- In từng ký tự ra màn hình.
- Đóng file.

3. Chương trình người dùng: copy

Viết chương trình **copy** với yêu cầu người dùng nhập tên file nguồn và đích, sau đó thực hiện copy.

Chương trình này cài đặt rất dễ dàng, bao gồm các bước sau đây:

- Khai báo mảng tên file_1, file_2, openFileId_1, openFileId_2, biến chạy, biến tạm.
- Nhập tên file_1, thực hiện mở thử file. Nếu thất bại thì thông báo.
- Nhập tên file_2, tạo file mới có tên file_2.
- Thực hiện copy từng ký tự.
- Kết thúc chương trình.

4. Chương trình người dùng: delete

Viết chương trình **delete** để kiểm tra syscall Remove.

Chương trình này cài đặt rất dễ dàng, bao gồm các bước sau đây:

- Khai báo mảng tên file, nhập tên file.
- Xóa file. Thành công hay thất bại đều thông báo ra màn hình.

5. Chương trình người dùng: concatenate

Viết chương trình **concatenate** để nối nội dung của 2 file, và chương trình sẽ yêu cầu nhập tên của hai file cần nối.

Chương trình này cài đặt rất dễ dàng, bao gồm các bước sau đây:

- Khai báo mảng tên file_1, file_2, openFileId_1, openFileId_2, biến chạy, biến tạm.
- Đọc tên file_1, mở file_1.
- Đọc tên file_2, mở file_2.
- Nếu mở thất bại, thông báo thất bại và kết thúc chương trình.
- Seek tới vị trí cuối cùng của file_1 để biết kích thước của file_1 (Seek lại vị trí 0 để không ảnh hưởng việc đọc ghi).
- Seek tới vị trí cuối cùng của file_2.
- Ghi từng ký tự của file_1 vào trong file_2.
- Đóng file_1.
- Đóng file_2.

Tài liệu tham khảo

[1] How to install NachOS
https://www.fit.hcmus.edu.vn/~ntquan/os/setup_nachos.html

[2] Project 01 - Exceptions và các system calls đơn giản
Chi tiết hướng dẫn thực hiện đồ án 01 của cô Chung Thùy Linh

[3] Giao tiếp giữa hệ điều hành NachOS và chương trình
Tài liệu hướng dẫn thực hiện đồ án 01

[4] Cách viết một system call
Tài liệu hướng dẫn thực hiện đồ án 01

[5] Đồ án NachOS Nguyễn Thành Chung HCMUS
https://github.com/nguyenthanhchungfit/Nachos-Programing-HCMUS

Cảm ơn thầy cô đã đọc tài liệu này.