**HỆ ĐIỀU HÀNH**

**ĐỒ ÁN 02:**

**File + Multiprocess**

C:\Users\tdqua_000\Dropbox\SS-Slides\DeCuong-CDIO\Template CDIO v4.2\Templates\Hinh anh\LogoTruong.png

Bộ môn Mạng máy tính

Khoa Công nghệ thông tin

Đại học Khoa học tự nhiên TP HCM

**MỤC LỤC**

[I. Thông tin nhóm 3](#_Toc452318053)

[II. File 4](#_Toc452318054)

[1. Tổng quan 4](#_Toc452318055)

[2. Viết system call int CreateFile(char \*name) 4](#_Toc452318056)

[3. Viết systemcall OpenFileID Open(char \*name, int type) 4](#_Toc452318057)

[4. Viết systemcall int Close(OpenFileID id) 5](#_Toc452318058)

[5. Viết systemcall int Read(char \*buffer, int charcount, OpenFileID id) 5](#_Toc452318059)

[6. Viết systemcall int Write(char \*buffer, int charcount, OpenFileID id) 5](#_Toc452318060)

[7. Viết systemcall int Seek (int pos, OpenFileID id) 5](#_Toc452318061)

[8. Viết chương trình demo 6](#_Toc452318062)

[III. Đa chương – Multiprocess (thực ra là multithread). 6](#_Toc452318063)

[1. Viết system call SpaceID Exec(char\* name) 6](#_Toc452318064)

[2. Viết system call int Join(SpaceID id) 7](#_Toc452318065)

[3. Viết system call int Exit(SpaceID id) 7](#_Toc452318066)

[4. Viết system call int CreateSemaphore(char\* name, int semval) 7](#_Toc452318067)

[5. Viết system call int Up(char\* name), và int Down(char\* name) 7](#_Toc452318068)

[6. Chương trình shell 7](#_Toc452318069)

[IV. Tài liệu tham khảo 8](#_Toc452318070)

# Thông tin nhóm

|  |  |  |  |
| --- | --- | --- | --- |
| **MSSV** | **Họ Tên** | **Email** | **Điện thoại** |
| 1412669 | Ngô Huỳnh Ngọc Khánh | [nhnkhanh@arduino.vn](mailto:nhnkhanh@arduino.vn) | 097 276 8491 |
| 1412127 | Võ Quốc Đạt | [vqdproduction@gmail.com](mailto:vqdproduction@gmail.com) | 094 968 7556 |

# File

## Tổng quan

**Đây là phần do bạn Nguyễn Quốc Đạt làm chính. Bạn Ngô Huỳnh Ngọc Khánh lên ý tưởng.**

Chúng em nhận thấy đề có 2 loại filesystem: một là stub (giả lập) (dùng file trên unix), hai là realtime filesystem (tạo ra một cái file rồi xây dựng cấu trúc folder, file trên đó).

Nhận thấy đề không yêu cầu phải chơi realtime filesystem nên em chúng em chơi filestub cho nó nhanh. Đỡ phải code nhiều ạ.

## 2. Viết system call int CreateFile(char \*name)

Cái này không hề khó. Kịch bản như sau:

Bọn em chuyển từ user vào kernel!

Sau đó dùng filesystem với giao thức Create để tạo file thôi ạ. Cái này thì lại port của unix mà xử lý.

## 3. Viết systemcall OpenFileID Open(char \*name, int type)

Với yêu cầu là đọc / ghi và đọc, ngoài ra còn phải xử lý thêm vụ đọc / ghi cho console.

Chúng em quyết định sẽ dùng type = 0 (Đọc / ghi), 1 chỉ đọc, 2 dành cho console input, 3 dành cho console output. Chúng em làm thế vì cái console là đặc biệt, giờ lại dùng thêm một biến để đánh dấu nó thì rãnh quá, nên việc tận dụng biến type để làm luôn là quyết định của nhóm.

Sau đó thêm vào bảng tra file là xong! (qua index)

## 4. Viết systemcall int Close(OpenFileID id)

Chỉ đơn giải là dựa vào index id rồi xóa, nếu xóa ok thì trả về 0 còn lỗi thì -1 thôi ạ. Đơn giản lắm.

## 5. Viết systemcall int Read(char \*buffer, int charcount, OpenFileID id)

Không khác gì mấy so với ReadString.

Nếu id == 0 thì là ReadString thôi ạ.

Nếu id == 1 thì báo về -1 thôi.

Còn ngươc lại thì bọn em dùng thư viện có sẵn để đọc cho với số lượng ký tự. Cũng không cần code gì nhiều.

## 6. Viết systemcall int Write(char \*buffer, int charcount, OpenFileID id)

Không khác gì mấy so với PrintString.

Nếu id == 1 thì là ReadString thôi ạ.

Nếu id == 0 thì báo về -1 thôi.

Còn ngươc lại thì bọn em dùng thư viện có sẵn để ghi kết quả ra thôi ạ!

## 7. Viết systemcall int Seek (int pos, OpenFileID id)

Do lớp OpenFile có hai phần khai báo và cài đặt như là hai lớp riêng, tùy vào chỉ thị tiền xử lý mà sẽ biên dịch phần khai báo và cài đặt tương ứng. Do đó để hỗ trợ cho syscall Seek, nhóm đã chỉnh sửa chút ít trong cải hai lớp kia. Đó là thêm hàm OpenFile::GetCurrentPos() để lấy vị trí hiện tại của con trỏ và OpenFile::Seek(…) cho lớp chưa có. Hàm OpenFile::Seek(…) cũng được sửa lại giá trị trả về là int cho biết vị trí sau khi dịch, điều này rất hữu ích cho chương trình người dùng.

## 8. Viết chương trình demo

Các chương trình mà đồ án yêu cầu hầu hết chỉ cần thao tác bằng cách gọi syscall tương ứng và thuật toán đơn giản. Phần này được thể hiễn rõ trong mã nguồn ngắn gọn của các chương trình đó và không có gì đặc sắc nên nhóm không trình bày chi tiết phần này.

Chúng em sửa Halt thành Exit(0) hết để tiện cho việc xử lý Thread sau này.

# III. Đa chương – Multiprocess (thực ra là multithread).

Như đã nói ở trên, nhóm đã thay Halt và Exit(0) để đảm bảo chương trình sẽ không shutdown hệ điều hành!

Ở mỗi exception nhóm xóa đi Halt thay vào đó là increasePC

Để quản lý các frames bộ nhớ vật lý hệ điều hành Nachos cung cấp một biến toàn cục BitMap\* gPhysPageBitMap. Biến này cần được khai báo trong system.h, system.cc và chỉ thị biên dịch trong Makefile.common. Ngoài ra trong đồ án này, ta còn phải sử dụng nhiều biến toàn cục khác như cô đã hướng dẫn trong file hướng dẫn đa chương!

Các biến toàn cục khác như pTab, semTab,… được xây dựng theo hướng đối tượng như cô hướng dẫn, chúng em có thêm một chút xíu thủ thuật nhỏ tuy nhiên không đáng kế. Ví dụ như: thay vì viết class Sem theo cách của cô, chúng em viết theo kiểu kế thừa IS-A để nhanh chóng đạt được mục đích.

Phần này do bạn Ngô Huỳnh Ngọc Khánh viết, lên ý tưởng, sửa lỗi và viết report!

## 1. Viết system call SpaceID Exec(char\* name)

Chuyển vô kernel là ok, sau đó gọi pTab->ExecUpdate để thêm lệnh đó vào, không có gì khó khi đã viết lệnh như cô hướng dẫn.

Chúng em đã chỉnh lại cái mutex thay vì là multex (chắc cô viết sai chính tả chỗ này). Chúng em đảm bảo FreeSlot chỉ được cấp duy nhất cho một tiến trình vì đã có biến mutex cho phần này.

## 2. Viết system call int Join(SpaceID id)

Syscall Join(…) sử dụng tham số đầu vào là SpaceID id, thực chất nó chính là chỉnh số của đối tượng PCB chứa đặc tả của tiến trình trong mảng lưu trữ của đối tượng pTable.

Sau đó gọi phương thức PTable::JoinUpdate(…). Hoạt động trong phương thức này là tiến trình hiện tại sẽ chờ tiến trinh con kết thúc rồi cho phép nó kết thúc. Code như cô viết, bọn em không chỉnh gì nhiều, chỉ cài đặt lại thôi ạ.

## 3. Viết system call int Exit(SpaceID id)

Syscall Exit(…) tương tự như syscall join, exit được thực hiện nhờ gọi phương thức của lớp PTable::ExitUpdate(…).

## 4. Viết system call int CreateSemaphore(char\* name, int semval)

Syscall CreateSemaphore nhận hai giá trị đầu vào là tên và giá trị của semaphore. Quá trình tạo semphore này được biến toàn cục STable \* semTab xử lý, nó sẽ tạo một đối tượng Sem mới dựa trên thông tin truyền vào và lưu trữ vào mảng để xử lý về sau.

## 5. Viết system call int Up(char\* name), và int Down(char\* name)

Syscall Up và Down đều nhận vào tên của semaphore. Quá trình này thực chất là đối tượng semTab sẽ tìm trong bảng lưu trữ của nó xem Sem nào có tên như vậy để gọi phương thức Semaphore::Wait() (cho Down) và Semaphore::Signal (cho Up) (mà thực chất cũng chỉ là gọi phương thức Semaphore::P() và Semaphore::V()).

## 6. Chương trình shell

Chương trình dùng để nhận một lệnh tại một thời điểm và thực thi chương trình tương ứng.

Quá trình xử lý của chương trình shell:

Chạy vòng lặp do..while(1) với điều kiện dừng khi người dùng nhập lệnh “exit”.

Yêu cầu người dùng nhập vào tên chương trình cần chạy.

Kiểm tra chương trình có tồn tại ở trên máy không, nếu không thì yêu cầu nhập lại.

Kiểm tra chương trình kí tự đầu trong chuỗi buffer người dùng nhập vào.

# IV. Tài liệu tham khảo

* Hướng dẫn đa chương
* <https://github.com/kuroyakumo95/nachoshotga>
* <https://github.com/dangkhoasdc/nachos>