ĐẠI HỌC QUỐC GIA THÀNH PHỐ HỒ CHÍ MINH
TRƯỜNG ĐẠI HỌC KHOA HỌC TỰ NHIÊN


# ĐỒ ÁN 01 NACHOS OS

## HỆ ĐIỀU HÀNH

21CLC08

GIẢNG VIÊN LÊ GIANG THANH
GIẢNG VIÊN LÊ HÀ MINH
GIẢNG VIÊN NGUYỄN THANH QUÂN

1. TRẦN BÌNH KHA 21127065
2. LÊ VŨ NGÂN LAM 21127334
3. TRẦN TÙNG LÂM 21127337

# Contents

# 1 Thông tin nhóm

| STT | MSSV | Họ và tên |
|---|---|---|
| 1 | 21127065 | \|l\| Trần Bình Kha |
| 2 | 21127334 | \|l\| Lê Vũ Ngân Lam |
| 3 | 21127337 | \|l\| Trần Tùng Lâm |

# 2 Cài đặt tổng quan

## 2.1 Cài đặt sơ bộ

- Ở ../code/threads/ trong file system.h và system.cc thực hiện khai báo, cấp phát, và xóa vùng nhớ cấp phát một biến toàn cục thuộc lớp "SynchConsole" để hỗ trợ việc nhập xuất với màn hình console.
- Ở ../code/userprog trong file exception.cc thực hiện cài đặt hai hàm char* User2System(int virtAddr, int limit) để sao chép vùng nhớ từ user sang system và hàm int System2User(int virtAddr, int len, char* buffer) để sao chép vùng nhớ từ system về cho user.

## 2.2 Giá trị thanh ghi

- R2: Lưu mã syscall đồng thời lưu kết quả trả về của mỗi syscall nếu có.
- R4: Lưu tham số thứ nhất.
- R5: Lưu tham số thứ hai.
- R6: Lưu tham số thứ ba.
- R7: Lưu tham số thứ tư.

## 2.3 Các bước cài đặt một system call

- Bước 1: ./code/userprog/syscall.h

```
#define SC_Create 4
int Create(char* name); // function prototype
```

**Listing 1:** Định nghĩa syscall dùng switch-case

- Bước 2: ./code/test/start.c và /code/test/start.s thêm dòng

```
.globl Create
.ent Create
Create:
addiu $2, $0, SC_Create
syscall
j $31
1512034_1512042_1512123 5
.end Create
```

**Listing 2:** start.s

- Bước 3: ./code/userprog/exception.cc sửa điều kiện if thành switch.. case (chỉ sửa một lần duy nhất, lần sau cứ theo format sẵn mà làm cho từng case system call)
- Bước 4: Viết chương trình ở mức người dùng để kiểm tra file .c ./code/test. Sử dụng hàm như đã khai báo prototype ở /code/userprog/syscall.h

- Bước 5: ./code/test/Makefile Thêm tên chương trình (fileName) vào dòng all

```
all: halt shell matmult sort (fileName in ./test)
```

**Listing 3:** Makefile

Thêm đoạn sau phía sau malmult

```
<fileName>o: <fileName>.c
$(CC) $(CFLAGS) -c <fileName>.c
<fileName>: <fileName>.o start.o
$(LD) $(LDFLAGS) start.o <fileName>.o -<fileName>.coff
../bin/coff2noff <fileName>.coff <fileName>
```

**Listing 4:** Makefile

- Bước 6: Biên dịch lại nachos, cd tới ./nachos/code/build.linux và chạy lệnh sau: 'make depend  make all'
- Bước 7: Chạy lệnh 'make all' trong thư mục test và chạy thử chương trình:
  - ../build.linux/nachos -x <executeName> (đang ở thư mục test)
  - ./nachos -x ./test/<executeName> (đang ở thư mục build)

# 3 Cài đặt SystemCall

## 3.1 Cài đặt hàm move_program_counter()

Làm tăng Programming Counter để nạp lệnh tiếp theo để thực hiện. Ta thực hiện lưu giá trị của PC hiện tại cho PC trước, nạp giá trị kế cho PC hiện tại, nạp giá trị kế tiếp nữa cho PC kế.

## 3.2 Cài đặt syscall Create(char *name, int initialSize)

Ta sử dụng hàm có sẵn OpenForWrite(name) để tạo một file với tên được người dùng nhập vào.

Nếu hàm này xảy ra bất kì lỗi nào, fileDescriptor sẽ nhận giá trị là -1. Khi nhận giá trị này, ta thoát hàm ngay và trả về FALSE.

Nếu không lỗi, ta đóng file và trả về TRUE.

## 3.3 Cài đặt syscall OpenFileID Open(char *name, int type), Close(OpenFileID id)

### 3.3.1 OpenFileID Open(char *name, int type)

- Bước 1: Tạo fdFileSocket[20] để chứa 20 descriptor của file và của socket. Trong đó, vị trí số 0 và số 1 tương ứng dành cho console input và output.
- Bước 2: Tạo **OpenFile **fileTable[20]** để chứa các file bình thường (Lưu ý, mảng này mới là để chứa dữ liệu chính của file, chứ không đơn thuần chứa thông tin tóm tắt như fdFileSocket[]). Giả sử tất cả 20 vị trí trong fdFileSocket[] đều dùng để mở file bình thường thì số lượng file tối đa của fileTable[] là 20. Ta khởi tạo 20 vị trí để phòng trường hợp này.
- Bước 3: Tạo openFileMode[20] để chứa các loại mở file hiện tại, mỗi phần tử nhận các giá trị bao gồm:
  - 0: Tương ứng cho file đọc và viết (MODE_READ_AND_WRITE)

  - 1: Tương ứng cho file chỉ đọc (MODE_WRITE)

- Bước 4: Mỗi khi hàm Open trong filesys.h được gọi, ta sẽ tiến hành kiểm tra như sau:

  1. Khởi tạo:
     - Biến freeIndex1 để tìm vị trí còn trống dùng để mở file trong fileTable[], khởi tạo bằng -1.
     - Biến freeIndex2 để tìm vị trí còn trống dùng để mở file trong fdFileSocket[], khởi tạo bằng -1.
     - Biến fileDescriptor khởi tạo bằng -1.
  2. Dùng vòng lặp chạy qua toàn bộ các phần tử trong mảng fileTable[], fdFileSocket[] và kiểm tra xem có tồn tại bất kì chỗ trống nào trong 2 mảng hay không. Nếu có, lưu lại vị trí đó vào biến freeIndex1 và freeIndex2. Nếu không tìm thấy 1 trong 2 vị trí đó, ta thoát hàm ngay, trả về -1.
  3. Kiểm tra xem mode mở có thuộc 2 loại: *đọc và viết*, *chỉ đọc* hay không. Nếu có, ta sẽ thực hiện mở file bằng hàm **OpenForReadWrite** và **OpenForRead**, đồng thời lưu lại fileDescriptor được trả về từ 2 hàm đó. Nếu fileDescriptor nhận giá trị -1, tức là việc mở file đã bị lỗi, ta sẽ thoát chương trình ngay lúc này.
  4. Ta tiến hành mở file bằng hàm **OpenFile(fileDescriptor)** đã có sẵn, lưu kết quả trả về từ hàm này vào fileTable[freeIndex1] (mảng chỉ dùng để lưu các file bình thường của chúng ta)
  5. Ta phải lưu thêm tên file vào **fileName[]** để thuận tiện trong việc kiểm tra file có đang mở ở hàm Remove (ta sẽ đề cập sau).
  6. Ta lưu thêm mode mở file vào mảng **openFileMode[]** để thuận tiện trong việc chạy hàm Read và Write (ta sẽ đề cập sau).
  7. Sau đó, ta cần liên kết file này vào mảng tổng: fdFileSocket[] bằng cách sau:

```
1    fdFileSocket[freeIndex2].type = 0;
2    fdFileSocket[freeIndex2].id = freeIndex1;
```

**Listing 5:** Liên kết file vừa rồi ở trong fileTable vào fdFileSocket

     Ở đây, type = 0 tức là file bình thường, type = 1 tức là file socket, type = -1 tức là vị trí còn trống.

  8. Tất cả bước trên đều không lỗi và thoát chương trình thì ta sẽ đến bước cuối cùng là trả về freeIndex2 đó làm OpenFileId.

#### 3.3.2 bool Close(int id)

Phần cài đặt hàm Close(int id) khá đơn giản, ta chỉ cần duy nhất một câu lệnh điều kiệnd để kiểm tra các điều sau:

- id nhập vào phải nằm trong 20 vị trí được cho sẵn trong fdFileSocket[]
- fdFileSocket[id] phải được mở rồi (tức giá trị tại fdFileSocket[id].type != -1)

Nếu 2 điều kiện trên thoả mãn, ta sẽ tiến hành xoá nó khỏi fdFileSocket[]. Lưu ý, ta phải đảm bảo rằng xoá trong fileTable[] trước rồi mới xoá trong fdFileSocket[], và cả 2 vị trí phải được xoá, nếu không sẽ sinh ra lỗi. (Vì chúng ta lưu file trên cả 2 mảng)

## 3.4 Cài đặt syscall Read(char *buffer, int charCount, int id), Write

### 3.4.1 int Read(char *buffer, int charCount, int id)

1. Khi bắt đầu chạy hàm này, ta phải kiểm tra xem rằng file id có hợp lệ hay không (nằm trong 20 vị trí của mảng fdFileSocket). Nếu không, ta thoát chương trình ngay và trả về giá trị lỗi: -1.

2. Kiểm tra tiếp rằng đây có phải là file socket hay không, nếu là file socket, ta sẽ chạy vào hàm xử lý socket (đề cập rõ hơn ở phần sau của report này). Nếu không, ta xét tiếp điều kiện thứ 3.

3. Ta cần kiểm tra rằng tại id được truyền vào thì có tồn tại file đang mở và phải là file bình thường (type = 0) hay không. Nếu không, ta thoát chương trình ngay và trả về giá trị lỗi: -1.

4. Cuối cùng, nếu 3 điều kiện trên đều thoả mãn, ta gọi đến hàm **Read(buffer, charCount)** có sẵn của system để đọc file. Hàm này trả về kí tự đọc được. Ta cần kiểm tra kí tự đọc được thực tế có bằng với kí tự mong muốn hay không. Nếu không, ta trả về giá trị lỗi. Nếu có, trả về số kí tự đó.

### 3.4.2 int Write(char *buffer, int charCount, int id)

Ở hàm Write, ta cũng tương ứng kiểm tra các điều kiện như ở hàm Read, được nhắc lại như sau (có một số điểm khác biệt ở các bước cuối):

1. Khi bắt đầu chạy hàm này, ta phải kiểm tra xem rằng file id có hợp lệ hay không (nằm trong 20 vị trí của mảng fdFileSocket). Nếu không, ta thoát chương trình ngay và trả về giá trị lỗi: -1.

2. Kiểm tra tiếp rằng đây có phải là file socket hay không, nếu là file socket, ta sẽ chạy vào hàm xử lý socket (đề cập rõ hơn ở phần sau của report này). Nếu không, ta xét tiếp điều kiện thứ 3.

3. Ta cần kiểm tra rằng tại id được truyền vào thì có tồn tại file đang mở và phải là file bình thường (type = 0). Nếu không, ta thoát chương trình ngay và trả về giá trị lỗi: -1.

4. Ta kiểm tra mode hiện tại của file không được là MODE_READ hay không. Nếu không, ta thoát chương trình ngay và trả về giá trị lỗi: -1.

5. Cuối cùng, nếu 3 điều kiện trên đều thoả mãn, ta gọi đến hàm **Write(buffer, charCount)** có sẵn của system để đọc file. Hàm này trả về kí tự đọc được. Ta cần kiểm tra kí tự viết được thực tế có bằng với kí tự mong muốn viết hay không. Nếu không, ta trả về giá trị lỗi. Nếu có, trả về số kí tự đó.

## 3.5 Cài đặt syscall Seek(int pos, int id)

1. Khi bắt đầu chạy hàm này, ta phải kiểm tra xem rằng file id có hợp lệ hay không (nằm trong 20 vị trí của mảng fdFileSocket). Nếu không, ta thoát chương trình ngay và trả về giá trị lỗi: -1.

2. Ta cần kiểm tra rằng tại id được truyền vào thì có tồn tại file đang mở (type != -1) hay không. Nếu không, ta thoát chương trình ngay và trả về giá trị lỗi: -1.

3. Nếu thoả 2 điều kiện trên, ta cần kiểm tra pos có bằng -1 hay không (pos = -1 tức là ta cần di chuyển đến cuối file). Nếu có, ta sẽ cập nhật pos bằng độ dài của file bằng cách dùng hàm có sẵn Length(). Nếu không, ta chạy đến bước tiếp theo.

4. Nếu pos là một giá trị không hợp lệ (<0 hoặc > độ dài của file) thì ta thoát chương trình ngay và trả về giá trị lỗi: -1.

5. Đến đây, ta gọi hàm Seek được cung cấp sẵn để hoàn thành nhiệm vụ của hàm này.

### 3.6 Cài đặt syscall Remove(char *name)

Để xoá file, trước hết ta cần phải kiểm tra file hiện có đang trong trạng thái mở không.

Để kiểm tra điều này, ta chạy một vòng lặp for qua tất cả các vị trí của fdFileTable[], nếu tồn tại 1 file đang mở và file đó có tên bằng với tên file cần remove (ta dùng mảng fileName[] đã tạo ở phần Open() để kiểm tra, ta thoát chương trình ngay và trả về giá trị lỗi: 0.

Ngược lại, ta gọi đến hàm Unlink(name) có sẵn để tiến hành xoá file.

### 3.7 Cài đặt syscall socketTCP

Các bước cài đặt:

- Bước 1: Tạo fdTable[20] để chứa 20 file descriptor được trả về từ hàm socket().
- Bước 2: Sau khi có được một cái socketid, ta sẽ thêm vào fdTable tạo ở trên. Đồng thời, phải kiểm tra xem fdTable có trống hay không cũng như có cái socketid nào khả thi để mà trả về không.

Hàm socketTCP() sẽ trả về file descriptor id khả thi từ fdTable, ngược lại nếu không có id nào khả thi thì trả về -1.

### 3.8 Cài đặt syscall Connect

Sau khi nhận được socketid khả thi để tạo kết nối với server thì client sẽ kết nối tới server thông qua địa chỉ ip: '127.0.0.1' và port '9000'. Nếu kết nối thành công thì trả về 0, ngược lại trả về -1.

### 3.9 Cài đặt syscall Send và Receive

Gửi và nhận data từ server

- Nếu thành công thì trả về số bytes của data đã gửi hay đã nhận.
- Nếu ngắt kết nối thì trả về 0
- Nếu thực hiện thao tác thất bại thì trả về -1

### 3.10 Cài đặt syscall closeSocketTCP

- Khi user dừng kết nối thì sẽ đóng socket lại, trả về 0 nếu đóng thành công, ngược lại trả về -1.

### 3.11 Advanced

Trong part 1, ta lưu các file bình thường chính vào OpenFile **fileTable[]. Tuy nhiên, để ghép mảng này vào mảng ở part 2, ta sẽ cần phải tạo ra một mảng mới mang tên fdFileSocket[] (mảng này chúng ta đã đề cập một chút ở các phần trên nhưng chưa đi vào chi tiết cấu trúc và cách hoạt động của mảng).

Mảng này sẽ mang kiểu dữ liệu tự định nghĩa mang tên FileSocket. Cấu trúc đó như sau:

```
struct FileSocket
{
  int type;
  int id;
};
```

**Listing 6:** Cấu trúc FileSocket

Trong đó, trường type sẽ chỉ chứa 3 giá trị tương ứng sau:

1. -1: vị trí này đang trống, chưa chứa bất kì file hay socket nào.
2. 0: đang mở một file bình thường.
3. 1: đang mở một socket.

Trường id mang ý nghĩa sau:

1. Đối với file bình thường: id này sẽ là vị trí của file trong fileTable[]. Ta cần lưu lại trường này để sau dễ dàng truy xuất file ở các vị trí trong fileTable[].
2. Đối với socket: id ở đây sẽ là socketid khả thi để sử dụng được trả về từ vị trí freeIndex có trong bảng.

Như vậy, ta đã có thể ghép 2 mảng lại với nhau, phân biệt file bình thường và socket qua trường type. Mỗi lần cần thao tác trên fileTable[] thuộc file hoặc các hàm của socket, ta cần phải check type trước. Nếu type thoả mãn thì ta mới tiếp tục thực hiện.

## 3.12 Test program

### 3.12.1 Cài đặt create

- Bước 1: Cho người dùng nhập tên file
- Bước 2: Gọi hàm Create và tiến hành tạo file

### 3.12.2 Cài đặt copy

- Bước 1: Mở file nguồn và kiểm tra. Nếu không mở được thì báo lỗi ra console
- Bước 2: Gọi hàm Create và tiến hành tạo file đích. Kiểm tra nếu không mở được file đích thì báo lỗi ra console
- Bước 3: Gọi hàm Read để lấy nội dung của file nguồn, sau đó gọi hàm Write để lưu nội dung vào file đích.
- Bước 4: Đóng file nguồn và file đích

### 3.12.3 Cài đặt cat

- Bước 1: Mở file nguồn và kiểm tra. Nếu không mở được thì báo lỗi ra console
- Bước 2: Gọi hàm Read để lấy nội dung của file nguồn, sau đó gọi hàm PrintString để in nội dung ra console
- Bước 4: Đóng file nguồn

### 3.12.4 Cài đặt delete

- Bước 1: Cho người dùng nhập tên file
- Bước 2: Gọi hàm Remove và tiến hành xóa file

#### 3.12.5 Cài đặt concatenate

- Bước 1: Mở file nguồn và kiểm tra. Nếu không mở được thì báo lỗi ra console
- Bước 2: Mở file đích và kiểm tra. Nếu không mở được thì báo lỗi ra console
- Bước 3: Gọi hàm Read để lấy nội dung của file nguồn, sau đó gọi hàm Seek để đưa con trỏ xuống cuối của file đích. Cuối cùng gọi hàm Write để ghi nội dung vào file đích.
- Bước 4: Đóng file nguồn và file đích

#### 3.12.6 Cài đặt echo socket

- Bước 1: Thực hiện kết nối 4 clients vào server
- Bước 2: Người dùng nhập data từ bàn phím và gửi data đến server. Server nhận được data sẽ thực hiện hàm viết hoa data lên và sau đó gửi ngược lại về phía người dùng. Người dùng nhận được data sẽ in ra màn hình data.
- Bước 3: Ngắt kết nối đến server và clear file descriptor table

#### 3.12.7 Cài đặt file transfer socket

- Bước 1: Thực hiện mở socket và kết nối vào server
- Bước 2: Mở file nguồn, kiểm tra nếu mở thành công thì tạo mới và mở file đích. Nếu không thì báo lỗi vào console
- Bước 3: Sử dùng hàm Read để đọc nội dung từ file nguồn. Gọi hàm Send để chuyển data sang server. Sau khi server xử lý xong sẽ dùng hàm Receive để nhận lại kết quả. Cuối cùng dùng hàm Write để ghi nội dung vào file đích
- Bước 4: Thực hiện đóng file nguồn và file đích. Ngắt kết nối socket

## 4 Đánh giá mức độ hoàn thành

| Bài | Câu | Ghi chú | Mức độ hoàn thành |
|---|---|---|---|
| 1 | 1 | syscall Create | 100% |
| | 2 | syscall OpenFileID, Close | 100% |
| | 3 | syscall Read,Write | 100% |
| | 4 | syscall Seek | 100% |
| | 5 | syscall Remove | 100% |
| 2 | 1 | syscall socketTCP | 100% |
| | 2 | syscall Connect | 100% |
| | 3 | syscall Send | 100% |
| | 4 | syscall Receive | 100% |
| | 5 | syscall CloseSocketTCP | 100% |
| 3 | | Advanced | 100% |
| 4 | | Test program | 100% |
| | 1,2,3,4,5 | create, copy, cat, delete, concatenate | 100% |
| | 6 | echo | 100% |
| | 7 | file transfer | 100% |

# 5 Bảng phân công công việc

| MSSV | Họ và tên | Công việc | Độ hoàn thiện |
|---|---|---|---|
| 21127065 | Trần Bình Kha | Part 4: 1, 2, 3, 4, 5, 7<br>Report | 100% |
| 21127334 | Lê Vũ Ngân Lam | Part 1, Part 3<br>Report | 100% |
| 21127337 | Trần Tùng Lâm | Part 2, Part 4: 6, Part 3<br>Report | 100% |

# 6 Tài liệu tham khảo


- File trong tài liệu hướng dẫn thực hành
- Lập trình Nachos HCMUS
- Github Lê Duy Thức
- Github nguyenthanhchungfit