ĐẠI HỌC QUỐC GIA THÀNH PHỐ HỒ CHÍ MINH
TRƯỜNG ĐẠI HỌC KHOA HỌC TỰ NHIÊN
KHOA CÔNG NGHỆ THÔNG TIN

BÁO CÁO

PROJECT 1: Tìm hiểu Nachos

LỚP: 21_21

Thành viên: Đàm Hồng Hưng – 21120461
Đoàn Đức Hữu – 21120465
Quách Đức Huy – 21120472

Giảng viên hướng dẫn: Lê Giang Thanh
Nguyễn Thanh Quân
Lê Hà Minh

2023-TP Hồ Chí Minh

Nội dung

Thông tin đồ án

Môi trường lập trình

- Công cụ: WSL , Visual Studio Code
- Ngôn ngữ lập trình: C
- Hệ điều hành: Linux (Ubuntu 18.04 LTS)

Phân công

Thành viên	Công việc	Hoàn thành
Đàm Hồng Hưng	Cài đặt system call theo tác với network tcp, viết báo cáo	100%
Đoàn Đức Hữu	Cài đặt system call thao tác với files, viêt báo cáo	100%
Quách Đức Huy	Viết chương trình test và báo cáo	100%

Nội dung

I. Cài đặt tổng quan

1. Biên dịch và cài đặt Nachos

Theo hướng dẫn từ **How to install NachOS on Ubuntu 18.04** của thầy hướng dẫn, kết quả sau khi cài đặt Nachos thành công bằng cách chạy lệnh **../build.linux/nachos -x halt** trong thư mục **./nachos/code/test** như sau:

```
Machine halting!

Ticks: total 22, idle 0, system 10, user 12
Disk I/O: reads 0, writes 0
Console I/O: reads 0, writes 0
Paging: faults 0
Network I/O: packets received 0, sent 0
```

2. Các bước viết một SystemCall mới

Theo hướng dẫn của đồ án, chúng ta sẽ hình thành các bước cài đặt thông qua việc cài đặt system call **SC_Create** nhằm phục vụ yêu cầu của chương trình người dùng muốn tạo một tập tin và cách system call trong hệ điều hành Nachos nhận tham số

Bước 1 : Trong tập tin **./code/userprog/syscall.h,** khai báo syscall mới như sau:

```
#define SC_Create 4 // define syscall để dùng trong switch-case
int Create (char* name); // Khai báo prototype của hàm
```

Bước 2 : ./code/test/start.s thêm dòng

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

- Lưu ý**: thanh ghi r2** dùng để lưu mã system call đồng thời lưu kết quả trả về của mỗi system call nếu có.

Bước 3: Trong tập tin **./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). **Lưu ý về quy định các giá trị thanh ghi :**

r4: Lưu tham số thứ nhất
r5: Lưu tham số thứ hai.
r6: Lưu tham số thứ ba.
r7: Lưu tham số thứ tư

Bước 4 : Viết chương trình ở mức người dùng để kiểm tra file **.c** trong tập tin **./code/test.** Sử dụng hàm như đã khai báo prototype ở **./code/userprog/syscall.h**

Bước 5 : Trong tập tin **./code/test/Makefile** thêm tên chương trình (tên file) vào dòng all

all: halt shell matmult sort createfile

Thêm đoạn sau phía sau **malmult**

```
createfile.o: createfile.c

	$(CC) $(CFLAGS) -c createfile.c

createfile: createfile.o start.o

	$(LD) $(LDFLAGS) start.o createfile.o -o createfile.coff

	../bin/coff2noff createfile.coff createfile
```

Bước 6: Biên dịch lại nachos bằng cách cd tới **./nachos/code** chạy lên "**make**".
Bước 7: Trong thư mục **./nachos/code/test**, gõ "**make**" để biên dịch các file trong **Makefile**
Bước 8: Chạy thử chương trình đứng từ thư mục **./nachos/code/test** :

../build.linux/nachos -x <Tên chương trình>

3. Cài đặt 2 hàm hỗ trợ: User2System và System2User

- Trong tập tin ./nachos/userprog/exception.cc , ta cài đặt hai thủ tục copy vùng nhớ gồm:

- **char* User2System (int virtAddr, int limit)**
  Input: Địa chỉ vùng nhớ User, kích thước giới hạn của buffer
  Output: Chuỗi buffer
  Mục đích: Sao chép buff từ vùng nhớ User sang vùng nhớ System

- **int System2User (int virtAddr, int len, char* buffer)**
  Input: Địa chỉ vùng nhớ User, kích thước giới hạn của chuỗi buffer, chuỗi buffer.

Output: Số byte được sao chép
Mục đích: Sao chép buffer từ vùng nhớ System sang vùng nhớ User.

- **void IncreasePC ()**

Mục đích: Trong MIPS cần sử dụng program counter để chứa vị trí câu lệnh tiếp theo cần xử lý vì vậy sau khi thực hiện mỗi system call ta cần thực hiện thao tác tăng program counter lên thông qua hàm **IncreasePC()**

4. Chỉnh sửa và cài đặt thêm một số thuộc tính, phương thức khác

- **Trong filesys.h**
  - Thêm thuộc tính **OpenFile** fileDescriptorTable** để chứa các file descriptor
  - Các phương thức được thêm mới và chỉnh sửa trong class FileSystem

```
// Tạo 20 file descriptor với 2 file đầu dành cho Console IO
FileSystem() ···
// Giải phóng bộ nhớ 20 file descriptor
~FileSystem() ···
//  ------------------------------  Quản lý phần File ----------------------------------------
// Tạo mới một file và trả về true nếu tạo file thành công
bool Create(char *name) ···
// Xóa file chỉ định và trả về true nếu xóa file thành công
bool Remove(char *name) { return Unlink(name) == 0; }
OpenFile *Open(char *name) ···
// Mở một file với tên và kiểu read-write hoặc read-only
// trả về đối tượng đó, trả về NULL nếu thất bại
OpenFile *Open(char *name, int mode) ···
// Đóng file thành công trả về 0, trả về -1 nếu thất bại
int Close(int fileId) // fileId có thể là openFileId hoặc là Socket id ···
// Trả về số byte đã đọc được, -1 nếu thất bại
int Read(char *buffer, int charCount, int openFileId) ···
// Viết số kí tự (charCount) kí tự vào đối tượng có openFileId chỉ định,
// trả về số kí tự viết được hoặc -1 nếu thất bại
int Write(char *buffer, int charCount, int openFileId) ···
// Xóa một đối tượng OpenFile tại vị trí fileId của fileDescriptorTable, trả về true nếu thành công
// Nếu fileId là một số lớn thì đó là socket id
bool RemoveFileDescriptor(int fileId) ···
// Thêm đối tượng OpenFile có fileName vào index chỉ định trong fileDescriptorTable
OpenFile *AddToFileDescriptorTable(int index, char *filename, int mode) ···
// Kiểm tra xem fileDescriptorTable còn slot trống không
int GetFileDescriptorEmptySlot() ···
// Lấy đối tượng OpenFile thông qua openFileId, trả về đối tượng nếu
// file đang mở(tồn tại trong fileDescriptorTable) , trả về NULL nếu thất bại
OpenFile *GetFileDescriptorByID(int openFileID) ···
```

```
// Lấy đối tượng OpenFile thông qua tên file, trả về đối tượng nếu
// file đang mở(tồn tại trong fileDescriptorTable) , trả về NULL nếu thất bại
OpenFile *GetFileDescriptorByName(char *filename) ···
// Trả về OpenFileId thông qua filename, trả về -1 nếu không tìm thấy
int GetFileDescriptorID(char *filename) ···
// Seek con trỏ tới vị trí pos chỉ định đối với đối tượng OpenFile có id la openFileId
int Seek(int pos, int openFileId) ···
//  ---------------------------- Kết thúc phần Quản lý File ----------------------------------------
//  ----------------------------  Quản lý phần Socket ----------------------------------------
// Mở một socket và trả về vị trí socket nếu list Socket còn slot, nếu hết slot trả về -1
int OpenNewSocket() ···
// Connect socket tới server trả về 0 nếu thành công hoặc -1 nếu lỗi
int Connect(int socketDescriptor, char *ip, int port) ···
// Gửi message tới server thông qua id của nó(file descriptor id), trả về số byte đã gửi
// hoặc -1 nếu lỗi gửi thất bại hoặc không tồn tại id trong list socket
int SendSocketMessage(int socketDescriptor, char *buffer) ···
// Trả về 0 nếu nhận thành công, trả về -1 nếu lỗi hoặc socket id không tồn tại trong list socket,
// hoặc 0 cho End of File - eof
int ReceiveSocketMessage(int socketDescriptor, char *buffer, int length) ···
//  ----------------------------  Kết thúc phần Quản lý Socket ----------------------------------------
```

- **Trong openfile.h**
  - Thêm thuộc tính **char* filename**(tên của file), **int filetype** (loại file: 0 cho read-write và 1 cho only-read) và **int socketId** (id của socket)
  - Các phương thức được thêm mới và chỉnh sửa trong class **OpenFile**

```
// Mặc định SocketId có giá trị -1 để xử lí phần File,
// nếu SocketId có giá trị dương lớn hơn 100 , thì đây là file socket
int socketId;
OpenFile(int f)
{
  file = f;
  currentOffset = 0;
  fileName = NULL;
  fileType = 0;
  socketId = -1;
} // open the file
OpenFile(int f, char *fileName, int fileType)
{
  file = f;
  currentOffset = 0;
  this->fileType = fileType;
  this->fileName = new char[strlen(fileName)];
  strncpy(this->fileName, fileName, strlen(fileName));
  socketId = -1;
}
```

```
// Trả về loại file
int GetFileType()
{
  return fileType;
}
// Trả về tên của file
char *GetFileName()
{
  return fileName;
}
// Trả về true nếu đây là file socket, ngược lại trả về false
bool isSocket()
{
  return socketId != -1;
}
```

5. Chương trình server phục vụ kiểm tra socket network

- Tạo thư mục **server** trong thư mục **./nachos/code/test.** Thư mục **server** gồm 2 file **server.c** và **Makefile**. File **Makefile** dùng để biên dịch file **server.c** thành file **server.noff**
  - Trong file **server.c** cài đặt chương trình server được tham khảo từ: https://mohsensy.github.io/programming/2019/09/25/echo-server-and-client-using-sockets-in-c.html

II. Cài đặt System Call

1. System call Create: int Create (char *name).

- Tên system call: **SC_Create**
  - Input: **char* name:** tên file cần tạo

- Output: trả về 0 nếu tạo file thành công, -1 nếu thất bại

★ Cách thức cài đặt:

- Đọc địa chỉ tên file từ thanh ghi R4 và gán vào biến filenameAddr
- Phương thức **User2System** được gán vào biến filename, có chức năng copy chuỗi từ vùng nhớ của user space sang kernel space với tham số là filenameAddr và MAXLENGTH + 1 mặc định là 255 là bộ đếm của buffer dài 255 ký tự.
- Gọi hàm **SysCreate** với tham số truyền vào là biến **fileName.** Ở hàm **SysCreate** được định nghĩa trong tệp tin **ksyscall.h** sẽ thực hiện như sau:
  - Kiểm tra hợp lệ về tên file: nếu tên file rỗng hoặc null thì báo lỗi và trả về -1. Kiểm tra hợp lệ về tạo file: gọi hàm **Create** của class **FileSystem** với tham số **fileName** và trả về kết quả. Kết quả từ hàm **SysCreate** sẽ được gán cho biến **result** ghi vào thanh ghi R2.
- Cuối cùng xóa bộ nhớ địa chỉ tên fileName và gọi hàm **IncreasePC** để tăng progam counter để đi đến vị trí câu lệnh tiếp theo

```
case SC_Create:
{
    // Input: file name
    // Output: trả về 0 nếu tạo file thành công, -1 nếu thất bại
    char *filename;
    int filenameAddr = kernel->machine->ReadRegister(4);
    filename = User2System(filenameAddr, MAXLENGTH + 1);
    int result = SysCreate(filename);
    if (result == -1)
    {
        kernel->machine->WriteRegister(2, -1);
    }
    else
    {
        kernel->machine->WriteRegister(2, 0);
    }
    delete filename;
    IncreasePC();
    break;
}
```

2. System call Open: OpenFileID Open (char *name, int type).

- Tên system call: **SC_Open.**
  - Input: **char* name** : tên file , **int type**: kiểu mở file.
  - Output: trả về OpenFileId của file nếu mở file thành công, -1 nếu thất bại.

★ Cách thức cài đặt:

- Đọc địa chỉ tên file từ thanh ghi R4, đọc kiểu mở file từ thanh ghi R5.
- Quy định type:
  - Type = 0: mở file ở dạng đọc và ghi.
  - Type = 1: mở file ở dạng chỉ đọc.
  - Ngoài 2 giá trị trên thì báo lỗi.
- Thực hiện chép giá trị ở thanh ghi R4 từ vùng nhớ User sang System bằng phương thức User2System và gán vào biến filename. Giá trị chép được chính là tên file người

dùng yêu cầu.

- Kiểm tra nếu type mở file khác 0 và 1 thì ghi giá trị -1 vào thanh ghi R2, gọi hàm **IncreasePC()** và **ASSERTNOTREACH()** để dừng chương trình
- Gọi hàm **SysOpen** được định nghĩa trong tệp tin **ksyscall.h** với tham số tên file và type. Trong hàm này thực hiện các chức năng sau:
  - Kiểm tra tên file rỗng hoặc null thì báo lỗi và trả về -1
  - Gọi hàm **GetFileDescriptorID** được định nghĩa trong **lớp FileSystem** với tham số là tên file để kiểm tra file có tên chỉ định đang được mở không(tức là đã tồn tại trong mảng fileDescriptorTable). Nếu file đang mở thì trả về -1
  - Gọi hàm **getFileDescriptorFreeSlot** được định nghĩa trong lớp **FileSystem** kiểm tra xem còn vị trí để chứa file để mở không, nếu kết quả trả về -1 thì đã hết chỗ mở file, do đó trả về -1. Ngược lại kết quả khác -1 thì gọi hàm **AddFileDescriptorTable,** hàm này có chức năng vừa thêm file vào mảng fileDescriptorTable vừa gọi lệnh **Open** được định nghĩa trong class **FileSystem** để mở file với type chỉ định và trả về OpenFileId. Sau khi nhận được kết quả thì hàm **SysOpen** trả kết quả về.
- Kết quả từ hàm **SysOpen** sẽ được ghi vào thanh ghi R2.
- Giải phóng vùng nhớ của biến filename và gọi hàm **IncreasePC** để tăng progam counter để đi đến vị trí câu lệnh tiếp theo

```
case SC_Open:
{
    // Input: FileName, Type (mode)
    // Output: Trả ve openFileId nếu mở file thành công, -1 nếu lỗi
    int virtAddr = kernel->machine->ReadRegister(4);
    int type = kernel->machine->ReadRegister(5);
    char *filename;
    if (type != 0 && type != 1)
    {
        kernel->machine->WriteRegister(2, -1);
        DEBUG(dbgSys, "\n Type is not accepted.");
        IncreasePC();
        ASSERTNOTREACHED();
    }
    // Copy chuoi tu vung nho User Space sang System Space voi bo dem name
    filename = User2System(virtAddr, 32);
    DEBUG(dbgSys, "Filename requested: " << filename << "!!");
    int openFileID = SysOpen(filename, type);
    if (openFileID == -1)
    {
        kernel->machine->WriteRegister(2, -1);
        DEBUG(dbgSys, "\n Mo file that bai, co the do loi hoac khong con slot");
    }
    else
    {
        kernel->machine->WriteRegister(2, openFileID); // tra ve OpenFileID
        DEBUG(dbgSys, "\n Mo file thanh cong!!! ID File: " << openFileID);
    }
    delete[] filename;
    IncreasePC();
    break;
}
```

3. System call Close: int Close (OpenFileID id).

- Tên system call: **SC_Close**

- Input: **OpenFileId** : openFileId của file đang mở hoặc là id của file socket.
- Output: trả về OpenFileId của file nếu đóng file hoặc đóng socket thành công, trả về -1 nếu đóng thất bại.

★ Cách thức cài đặt:

- Lưu giá trị đọc được từ thanh ghi r4 vào biến FileId
- Gọi hàm **SysClose** trong tệp tin **ksyscall.h.** Hàm này có chức năng đóng file bằng cách gọi hàm **RemoveFileDescịptor** được định nghĩa trong class **FileSystem.** Trong hàm **RemoveFileDescịptor** sẽ thực hiện các chức năng sau:
  - Kiểm tra fileId có thuộc [2,20) , nếu thỏa mản thì giải phóng bộ nhớ file và trả về true.
  - Duyệt mảng fileDescriptorTable và gọi hàm **isSocket** để tra fileDescriptor có phải là file Socket không. Nếu là file socket và có id trùng với fileId thì gọi hàm **CloseSocket** của tệp tin sysdep.h để đóng socket và trả về kết quả true
- Hàm **SysClose** nhận kết quả từ **RemoveFileDescriptor** và trả về kết quả tương ứng
- Kết quả nhận được từ **SysClose** sẽ được ghi vào thanh ghi R2. Sau đó, gọi hàm **IncreasePC** để tăng progam counter để đi đến vị trí câu lệnh tiếp theo

```
case SC_Close:
{
    // Input : openFileId hoặc SocketId
    //  Output: Đóng file trả về 0 nếu thành công, -1 thất bại
    int fileID = kernel->machine->ReadRegister(4);
    int result = SysClose(fileID);
    if (result == -1)
    {
        kernel->machine->WriteRegister(2, -1);
    }
    else
    {
        kernel->machine->WriteRegister(2, 0);
    }
    IncreasePC();
    break;
}
```

4. System call Read: int Read (char *buffer, int size, OpenFileID id).

- Tên system call: **SC_Read**
  - Input: **char* buffer** mảng lưu nội dung cần đọc**,** kích thước file**,** OpenFileId của file
  - Output: trả đúng số lượng ký tự được read thành công, trả về -1 nếu read bị lỗi

★ Cách thức cài đặt:

- Lưu địa chỉ của buffer của ip đọc được từ thanh ghi R4 vào biến virtAddr
- Lưu giá trị đọc được từ thanh ghi R5 vào biến size
- Lưu giá trị đọc được từ thanh ghi R6 vào biến OpenFileId
- Thực hiện chép giá trị ở thanh ghi R4 từ vùng nhớ User sang System bằng phương thức **User2System** và gán vào biến filename. Giá trị chép được chính là tên file người dùng yêu cầu và lưu vào biến buffer

- Gọi hàm **SysRead** của tệp tin **ksyscall.h**. Hàm này thực hiện các chức năng sau:
  - Kiểm tra OpenFileId có nằm trong phạm vi [0,20) hoặc OpenFileId có khác 1(file **ConsoleOutput** không thể đọc), trả về -1 nếu nằm ngoài phạm vi hoặc bằng 1
  - Gọi hàm **GetFileDescriptorID** để kiểm tra file có đang open không, nếu kết quả bằng -1(tức là file chưa open) thì trả về -1
  - Kiểm tra nếu OpenFileId:
    - Nếu bằng 0(tức là file ConsoleInput) thì gọi hàm **GetString** được định nghĩa trong class **SynchConsoleInput** để đọc input từ màn hình console
    - Nếu khác 0(tức là file bình thường) thì gọi hàm **Read** của class **FileSystem** để đọc file. Hàm này đọc file và trả về số byte đã đọc được, nếu đọc thất bại thì sẽ trả về -1
- Kết quả từ hàm **SysRead** sẽ được ghi vào thanh ghi R2. Cuối cùng, giải phóng vùng nhớ của biến buffer và gọi hàm **IncreasePC** để tăng program counter để đi đến vị trí câu lệnh tiếp theo

```
case SC_Read:
{
    // Input: buffer(char*), so ky tu(size), openFileId
    // Output: -1 nếu lỗi, trả về đã đọc nếu thành công
    int virtAddr = kernel->machine->ReadRegister(4);
    int size = kernel->machine->ReadRegister(5);
    int openFileID = kernel->machine->ReadRegister(6);
    // *buf này để lưu input từ người dùng
    char *buffer = User2System(virtAddr, size);

    int result = SysRead(buffer, size, openFileID);
    if (result > 0)
    {
        kernel->machine->WriteRegister(2, result);
        // Trả về dữ liệu input từ buf(của hệ thống) về cho người dùng -> truyền vào virtAddr
        System2User(virtAddr, size + 1, buffer);
    }
    else
    {
        kernel->machine->WriteRegister(2, result);
    }
    delete buffer;
    IncreasePC();
    break;
}
```

5. System call Write: int Write (char *buffer, int size, OpenFileID id).

- Tên system call: **SC_Write**
  - Input: **char* buffer** mảng chứa nội dung cần ghi**,** kích thước file**,** OpenFileId của file
  - Output: trả đúng số lượng ký tự được write thành công, trả về -1 nếu write bị lỗi

★ Cách thức cài đặt:

- Lưu địa chỉ của buffer của ip đọc được từ thanh ghi R4 vào biến virtAddr
- Lưu giá trị đọc được từ thanh ghi R5 vào biến size
- Lưu giá trị đọc được từ thanh ghi R6 vào biến OpenFileId
- Thực hiện chép giá trị ở thanh ghi R4 từ vùng nhớ User sang System bằng phương thức **User2System** và gán vào biến filename. Giá trị chép được chính là tên file người

dùng yêu cầu lưu vào biến buffer

- Gọi hàm **SysWrite** của tệp tin **ksyscall.h**. Hàm này thực hiện các chức năng sau:
  - Kiểm tra OpenFileId có nằm trong phạm vi [0,20) hoặc OpenFileId có khác 0(file **ConsoleInput** không thể ghi), trả về -1 nếu nằm ngoài phạm vi hoặc bằng 0
  - Gọi hàm **GetFileDescriptorID** để kiểm tra file có đang open không, nếu kết quả bằng -1(tức là file chưa open) thì trả về -1
  - Kiểm tra nếu OpenFileId:
    - Nếu bằng 1(tức là file ConsoleOutput) thì gọi hàm **PutString** được định nghĩa trong class **SynchConsoleOutput** để xuất input ra màn hình console
    - Nếu khác 1(tức là file bình thường) thì gọi hàm **Write** của class **FileSystem** để ghi file. Hàm này ghi file và trả về số byte đã ghi được, nếu ghi thất bại thì sẽ trả về -1
- Kết quả từ hàm **SysWrite** sẽ được ghi vào thanh ghi R2. Cuối cùng, giải phóng vùng nhớ của biến buffer và gọi hàm **IncreasePC** để tăng program counter để đi đến vị trí câu lệnh tiếp theo

```
case SC_Write:
{
    // Input: buffer(char*), so ky tu(size), openFileId
    // Output: -1 nếu lỗi, trả về đã viết được nếu thành công
    int virtAddr = kernel->machine->ReadRegister(4);
    int size = kernel->machine->ReadRegister(5);
    int openFileId = kernel->machine->ReadRegister(6);

    char *buffer = User2System(virtAddr, size);
    int result = SysWrite(buffer, size, openFileId);
    if (result == -1)
    {
        kernel->machine->WriteRegister(2, -1);
    }
    else
    {
        kernel->machine->WriteRegister(2, result);
    }
    IncreasePC();
    delete buffer;
    break;
}
```

6. System call Seek: int Seek (int position, OpenFileID id).

- Tên system call: **SC_Seek**
  - Input: vị trí cần seek, OpenFileId của file chỉ định
  - Output: trả về vị trí nếu seek thành công, ngược lại trả về -1

★ Cách thức cài đặt:

- Lưu giá trị đọc được từ thanh ghi R4 vào biến pos
- Lưu giá trị đọc được từ thanh ghi R5 vào biến openFileID

- Gọi hàm SysSeek trong tệp tin ksyscall.h. Hàm này đầu tiên kiểm tra OpenFileId nếu thuộc 0 hoặc 1 (file ConsoleInput và ConsoleOuput) hoặc OpenFileId nằm ngoài phạm vi [0,20) thì trả về -1. Tiếp theo hàm sẽ gọi **GetFileDescriptorByID** để kiểm tra file đang mở không, nếu file không mở thì trả về -1. Cuối cùng gọi hàm **Seek** của class **FileSystem** để dịch chuyển con trỏ file tời vị trí pos chỉ định. Hàm **Seek** sẽ kiểm tra nếu pos bằng -1 thì dịch đến cuối file, nếu ngoài phạm vi thì trả về -1, ngược lại trả về vị trí dich chuyển thành công. Kết quả nhận được từ hàm **Seek** sẽ được hàm **SysSeek** nhận và trả về tương ứng
- Kết quả từ hàm **SysSeek** sẽ được ghi vào thanh ghi **R2.** Sau đó gọi hàm **IncreasePC** để tăng progam counter để đi đến vị trí câu lệnh tiếp theo

```
case SC_Seek:
{
    // Input: Vị trí(position), openFileId
    // Output: -1 nếu lỗi, trả về vị trí(seek) con trỏ trong file nếu thành công
    int pos = kernel->machine->ReadRegister(4);
    int openFileID = kernel->machine->ReadRegister(5);

    int result = SysSeek(pos, openFileID);
    if (result == -1) // Kiem tra lai vi tri pos co hop le khong
    {
        kernel->machine->WriteRegister(2, -1);
    }
    else
    {
        kernel->machine->WriteRegister(2, result);
    }
    IncreasePC();
    break;
}
```

7. System call Remove: int Remove (char *name).

- Tên system call: **SC_Remove**
  - Input: Tên file chỉ định
  - Output: trả về 0 nếu xóa file thành công, ngược lại trả về -1

★ Cách thức cài đặt:

```
case SC_Remove:
{
    int virtAddr;
    char *filename;
    virtAddr = kernel->machine->ReadRegister(4);
    filename = User2System(virtAddr, 32);
    int result = SysRemove(filename);
    if (result == -1)
    {
        kernel->machine->WriteRegister(2, -1);
    }
    else
    {
        kernel->machine->WriteRegister(2, 0);
    }
    delete filename;
    IncreasePC();
    break;
}
```

- Lưu giá trị socket id đọc được từ thanh ghi r4 vào biến virtAddr
- Thực hiện chép giá trị ở thanh ghi R4 từ vùng nhớ User sang System bằng phương thức **User2System** và gán vào biến filename. Giá trị chép được chính là tên file người dùng yêu cầu
- Thực hiện gọi hàm **SysRemove** trong tệp tin **ksyscall.h.** Hàm này có chức năng kiểm tra tên file nếu rỗng hoặc null thì trả về -1. Sau đó hàm sẽ gọi phương thức **GetFileDescriptorByName** để kiểm tra file có tên chỉ định có đang open không, nếu kết quả trả về khác -1 tức là file đang mở thì trả về -1. Cuối cùng hàm sẽ gọi hàm **Remove** được định nghĩa trong class **FileSystem** để đóng file. Kiểm tra nếu kết quả trả về từ hàm **Remove** nếu true thì trả về 0 ngược lại trả về -1.
- Kết quả nhận được từ hàm **SysRemove** sẽ được ghi vào thanh ghi R2
- Giải phóng vùng nhớ của biến filename và gọi hàm **IncreasePC** để tăng program counter đi đến vị trí câu lệnh tiếp theo.

8. System call SocketTCP: int SocketTCP ().

- Tên system call: **SC_SocketTCP**
  - Input: Không có tham số
  - Output: Tạo một socket mới trả về file descriptor id , hoặc -1 nếu lỗi.

★ Cách thức cài đặt:

- Gọi hàm **OpenNewSocket** của class **FileSystem**. Hàm này gọi phương thức **OpenNewSocketTCP** của tệp tin **sysdep.h** có chức năng mở một socket và trả về id socket nếu mở thành công, nếu hết slot hoặc bị lỗi thì trả về -1
- Trong hàm **OpenNewSoket**, trả về -1 nếu nhận kết quả -1 từ **OpenNewSocketTCP**, ngược sau khi nhận được id của Socket (khác -1) thì sẽ cộng thêm 100 để tránh bị trùng với OpenFileId của file, sau đó trả về socket id.
- Kết quả của hàm **OpenNewSocket** được lưu vào biến result và ghi vào thanh ghi r2
- Gọi hàm **IncreasePC** để tăng program counter đi đến vị trí câu lệnh tiếp theo

```
case SC_SocketTCP:
{
    // Input: không
    // Output: 0 nếu thành công , -1 nếu thất bại
    // Tạo socket và trả về file descriptor id hoặc -1 nếu lỗi.
    int result = kernel->fileSystem->OpenNewSocket();
    if (result != -1)
    {
        DEBUG(dbgSys, "Ket qua socket tcp: " << result);
        kernel->machine->WriteRegister(2, result);
    }
    else
    {
        kernel->machine->WriteRegister(2, -1);
        DEBUG(dbgSys, "Loi tao socket do tao that bao hoac het slot");
    }
    IncreasePC();
    break;
}
```

9. System call Connect: int Connect (int socketid, char *ip, int port).

- Tên system call: **SC_Connect**
  - Input: Socket id, ip chứa địa chỉ và port kết nối
  - Output: Trả về 0 nếu kết nối thành công, và -1 nếu bị lỗi

★ Cách thức cài đặt:

- Lưu giá trị socket id đọc được từ thanh ghi r4 vào biến socketId
- Lưu địa chỉ của buffer của ip đọc được từ thanh ghi r5 vào biến virtAddr
- Lưu giá trị port kết nối đọc được từ thanh ghi r6 vào biến port
- Phương thức **User2System** được gán vào biến ipAddress, có chức năng copy chuỗi từ vùng nhớ của user space sang kernel space với tham số là virtAddr và MAXLENGTH mặc định là 255 là bộ đếm của buffer dài 255 ký tự.
- Gọi hàm **Connect** của class **FileSystem**. Hàm này gọi phương thức **ConnectSocket** của tệp tin **sysdep.h** có chức năng kết nối socket và trả về -1 nếu kết nối thất bại, ngược lại trả về 0 nếu thành công. Sau khi nhận kết quả từ **ConnectSocket**, hàm **Connect** sẽ trả kết quả tương ứng

- Kết quả từ hàm **Connect** được lưu vào biến result và sau đó ghi kết quả result vào thanh ghi r2. Cuối cùng, giải phóng vùng nhớ của biến ipAddress và gọi hàm **IncreasePC** để tăng program counter đi đến vị trí câu lệnh tiếp theo

```
case SC_Connect:
{
    // Input: socketId, ipAddess, port
    // Output: 0 nếu kết nối đến server thành công, -1 nếu thất bại
    int socketId = kernel->machine->ReadRegister(4);
    int virtAddr = kernel->machine->ReadRegister(5);
    int port = kernel->machine->ReadRegister(6);
    char *ipAddress = User2System(virtAddr, 255);
    int result = kernel->fileSystem->Connect(socketId, ipAddress, port);
    if (result == -1)
    {
        kernel->machine->WriteRegister(2, -1);
        DEBUG(dbgSys, "Loi ket noi den Server.");
    }
    else
    {
        kernel->machine->WriteRegister(2, 0);
    }
    delete ipAddress;
    IncreasePC();
    break;
}
```

10. System call Send: int Send (int socketid, char *buffer, int len).

- Tên system call: **SC_Send**
  - Input: Socket id, buffer chứa message cần gửi đi và độ dài của buffer
  - Output: nếu thành công thì trả về số lượng bytes gửi đi , nếu kết nối bị đóng trả về 0, nếu thất bại trả về -1

★ Cách thức cài đặt:

- Lưu giá trị socket id đọc được từ thanh ghi r4 vào biến socketId
- Lưu địa chỉ của buffer đọc được từ thanh ghi r5 vào biến virtAddr
- Lưu giá trị len đọc được từ thanh ghi r6 vào biến size
- Phương thức **User2System** được gán vào biến buffer, có chức năng copy chuỗi từ vùng nhớ của user space sang kernel space với tham số là virtAddr và MAXLENGTH mặc định là 255 là bộ đếm của buffer dài 255 ký tự.
- Gọi hàm **SendSocketMessage** của class **FileSystem**. Hàm này kiểm tra xem có file descriptor nào có phải là file socket bằng cách gọi hàm **isSocket** của class **OpenFile** và có giá trị socketId trùng nhau thì gọi phương thức **SendMessageToSocket** của tệp tin **sysdep.h** có chức năng gửi message tới socket thông qua ip và port và trả về số byte đã gửi nếu thành công hoặc -1 nếu gửi thất bại. Nếu không trùng nhau thì duyệt đến khi hết file thì trả về -1
- Kết quả nhận được từ hàm **SendSocketMessage** được lưu vào biết result và ghi vào thanh ghi r2. Cuối cùng, giải phóng vùng nhớ của biến buffer và gọi hàm **IncreasePC**

để tăng program counter đi đến vị trí câu lệnh tiếp theo

```
case SC_Send:
{
    // Input: socketId, message cần gửi
    // Output: Trả về số byte gửi thành công đến server hoặc -1 nếu lỗi
    int socketId = kernel->machine->ReadRegister(4);
    int virtAddr = kernel->machine->ReadRegister(5);
    int length = kernel->machine->ReadRegister(6);
    char *buffer = User2System(virtAddr, MAXLENGTH);

    int result = kernel->fileSystem->SendSocketMessage(socketId, buffer, length);
    kernel->machine->WriteRegister(2, result);
    delete buffer;
    IncreasePC();
    break;
}
```

11. System call Receive: int Receive (int socketid, char *buffer, int len).

- Tên system call: **SC_Receive**
  - Input: Socket id, buffer là mảng để chứa dữ liệu nhận được và độ dài của buffer
  - Output: nếu thành công thì trả về số lượng bytes nhận được, nếu kết nối bị đóng trả về 0, nếu thất bại trả về -1

★ Cách thức cài đặt:

- Lưu giá trị socket id đọc được từ thanh ghi r4 vào biến socketId
- Lưu địa chỉ của buffer đọc được từ thanh ghi r5 vào biến virtAddr
- Lưu giá trị len đọc được từ thanh ghi r6 vào biến size
- Phương thức **User2System** được gán vào biến buffer, có chức năng copy chuỗi từ vùng nhớ của user space sang kernel space với tham số là virtAddr và MAXLENGTH mặc định là 255 là bộ đếm của buffer dài 255 ký tự.
- Gọi hàm **ReceiveSocketMessage** của class **FileSystem**. Hàm này kiểm tra xem có file descriptor nào có phải là file socket bằng cách gọi hàm **isSocket** của class **OpenFile** và có giá trị socketId trùng nhau thì gọi phương thức **ReceiveMessageFromSocket** của tệp tin **sysdep.h** có chức năng nhận message từ socket thông qua ip và port và trả về số byte đã nhận được nếu thành công hoặc -1 nếu nhận thất bại. Nếu không trùng nhau thì duyệt đến khi hết file thì trả về -1
- Kết quả nhận được từ hàm **ReceiveSocketMessage** được lưu vào biết result và ghi vào thanh ghi r2. Cuối cùng, giải phóng vùng nhớ của biến buffer và gọi hàm **IncreasePC** để tăng program counter đi đến vị trí câu lệnh tiếp theo

```
case SC_Receive:
{
    // Input: socketId , message từ server
    // Output: Trả về số byte nhận được từ server hoặc trả về -1 nếu thất bại
    int socketId = kernel->machine->ReadRegister(4);
    int virtAddr = kernel->machine->ReadRegister(5);
    int length = kernel->machine->ReadRegister(6);
    char *buffer = User2System(virtAddr, 255);

    int result = kernel->fileSystem->ReceiveSocketMessage(socketId, buffer, length);
    if (result == -1)
    {
        kernel->machine->WriteRegister(2, -1);
        DEBUG(dbgSys, "Nhan that bai: ");
    }
    else
    {
        kernel->machine->WriteRegister(2, result);
        System2User(virtAddr, 255, buffer);
    }
    delete buffer;
    IncreasePC();
    break;
}
```

12. System call ReadString: void ReadString (char* buffer).

- Tên system call: **SC_ReadString**
  - Input: Chuỗi buffer dạng mảng ký tự đại diện cho User Space
  - Output: Trả về độ dài chuỗi ký tự do người dùng nhập và lưu nội dung vào buffer

★ Cách thức cài đặt:

```
case SC_ReadString:
{
    // Đọc chuỗi từ input người dùng và lưu vào user space (virtAddr)
    int virtAddr = kernel->machine->ReadRegister(4);
    char *buffer = SysReadString();
    int length = strlen(buffer);
    System2User(virtAddr, length + 1, buffer);
    printf("So byte doc duoc: %d", length);
    kernel->machine->WriteRegister(2, length);
    delete[] buffer;
    buffer = NULL;
    IncreasePC();
    break;
}
```

- Biến virAddr lưu địa chỉ của buffer là tham số của hàm **ReadString** trong vùng nhớ của user space đã được ghi vào thanh ghi R4
- Tạo buffer kiểu char* là buffer của kernel space để đọc chuỗi được nhập từ bàn phím thông qua việc gọi hàm **SysReadString** của tập tin **ksyscall.h**
- Hàm **SysReadString** sẽ đọc chuỗi người dùng nhập cho đến khi đọc kí tự "\n"(Enter) bằng phương thức **GetChar** của class **SynchConsoleInput** được triển khai trong hàm **SysReadChar** trong cùng tệp tin **ksyscall.h**
- Sau đó, sử dụng phương thức **System2User** của class machine, với tham số truyền vào lần lượt là virtAddr, length + 1, buffer với length là kích thước của chuỗi cộng thêm 1 để tránh tràn bộ đệm. Chuỗi buffer ban đầu là của kernel space, phương thức này có chức năng copy chuỗi này qua User space

- Cuối cùng là hủy buffer kernel giúp giải phóng vùng nhớ và gọi hàm **IncreasePC** để tăng program counter đi đến ví trí câu lệnh tiếp theo

13. System call PrintString: void PrintString (char* buffer).

- Tên system call: **SC_PrintString**
  - Input: Chuỗi kí tự được lưu trữ trong mảng buffer
  - Output: In chuỗi kí tự đọc được và in ra màn hình Console

★ Cách thức cài đặt:

- Biến memPtr lưu địa chỉ của buffer là tham số của hàm **ReadString** trong vùng nhớ của user space đã được ghi vào thanh ghi R4
- Phương thức **User2System** được gán vào biến buffer, có chức năng copy chuỗi từ vùng nhớ của user space sang kernel space với tham số là memPtr và MAXLENGTH mặc định là 255 là bộ đếm của buffer dài 255 ký tự
- Gọi hàm **SysPrintString** trong tập tin **ksyscall.h** với tham số là buffer chứa chuỗi kí tự được copy trong kernel space và độ dài chuỗi copy. Trong hàm này sẽ gọi phương thức **SysPrintChar**, hàm mà gọi phương thức **PutChar** của class **SynchConsoleOuput** để in kí tự ra màn hình. Do đó trong hàm **SysPrintString** sẽ duyệt vòng lặp length lần để gọi hàm **SysPrintChar** để in từng kí tự ra màn hình
- Cuối cùng là hủy buffer kernel giúp giải phóng vùng nhớ và gọi hàm **IncreasePC** để tăng program counter đi đến ví trí câu lệnh tiếp theo

```
case SC_PrintString:
{
    // Xuất chuỗi ra màn hình console
    int memPtr = kernel->machine->ReadRegister(4);
    char *buffer = User2System(memPtr, MAXLENGTH);
    SysPrintString(buffer, strlen(buffer));
    delete[] buffer;
    IncreasePC();
    break;
}
```

```
// Xuất kí tự chỉ định ra màn hình console
void SysPrintChar(char character)
{
  kernel->synchConsoleOut->PutChar(character);
}
// In chuỗi kí tự hoặc mảng chuỗi ra màn hình console
void SysPrintString(char *buffer, int length)
{
  // Xuất length kí tự ra màn hình console bằng cách xuất từng kí tự
  int i = 0;
  while (i < length)
  {
    SysPrintChar(buffer[i++]);
  }
}
```

14. System call Halt: void Halt ().

- Tên system call: **SC_Halt**
  - Input: không có tham số
  - Output: Tắt Nachos và in ra thông tin thống kê về hiệu suất.

★ Cách thức cài đặt:

- Gọi hàm **SysHalt** của tệp tin **ksyscall.h**, hàm này sẽ gọi hàm **Halt** của class **Interrupt** để tắt Nachos và in ra thống kê hiệu suất
- Nếu **SysHalt** không hoạt động sẽ gọi hàm **ASSERTNOTREACHED** để gọi hàm **Abort**() trong **debug.h** tạo một tệp tin code dump

```
case SC_Halt:
{
    DEBUG(dbgSys, "Shutdown, initiated by user program.\n");
    SysHalt();
    ASSERTNOTREACHED();
    break;
}
```

```
void SysHalt()
{
  kernel->interrupt->Halt();
}
```

III.Chương trình Test

1. Viết chương trình createfile

Mục đích:

- Tạo ra một file mới từ tên file do người dùng nhập vào từ console từ **ReadString.**

Cách thức hoạt động:

- Sử dụng system call **PrintString** để thông báo người dùng nhập tên file cần tạo. Sau đó, dùng system call **ReadString** để lưu tên file do người dùng nhập từ console.

- Kiểm tra hợp lệ: Gọi system call **CreateFile** để tạo file với tham số truyền vào là tên file đọc được và lưu lại kết quả. Kiểm tra kết quả nhận được và dùng system call **PrintString** Nếu kết quả nhận được bằng 0 thì xuất thông báo tạo file thành công, ngược lại nếu bằng -1 thì xuất thông báo tạo thất bại.
  Hình ảnh demo chương trình

```
ddhuu@Admin:~/nachos/NachOS-4.0/code/test$ ./../build.linux/nachos -rs 1023 -x createfile
* * * Nhap vao ten file can tao: Nachos.txt
Tao file thanh cong
Machine halting!

Ticks: total 1141080533, idle 1141078273, system 2210, user 50
Disk I/O: reads 0, writes 0
Console I/O: reads 11, writes 53
Paging: faults 0
Network I/O: packets received 0, sent 0
```

Chương trình createfile khi người dùng tạo file thành công

2. Viết chương trình cat

Mục đích:

- Hiển thị nội dung của file.

Cách thức hoạt động:

- Nhập vào tên file cần đọc từ console từ **ReadString** và gọi System call **Open** để mở file với type bằng 1(Read-only)
- Kiểm tra hợp lệ: **OpenFileId** khác -1 thì sẽ lấy độ dài nội dung trong file bằng cách gọi system call **Seek** truyền tham số -1 và **OpenFileId** để đưa con trỏ đến cuối file, kết quả khi seek đến cuối file cũng là kích thước của file. Sau đó gọi lại **Seek** với tham số 0 để đưa con trò file về đầu file. Trong quá trình gọi **Seek** kiểm tra kết quả **Seek** có khác 0, nếu kết quả bằng -1 thì thông báo lỗi seek thất bại
- Với kích thước file đọc được, Dòng vòng lặp để đọc từ byte bằng cách gọi system call **Read** kết hợp với in kí tự đọc được bằng system call **PrintString. Trong quá trình** gọi system call **Read** nếu kết quả trả về -1 thì xuất thông báo lỗi và dừng chương trình.
- Sau khi đọc thành công sẽ tiến hành đóng file bằng system call **Close** nếu kết quả trả về -1 và in ra màn hình "đóng file thất bại", ngược lại sẽ in ra "đóng file thành công".

Hình ảnh Demo chương trình

```
ddhuu@Admin:~/nachos/NachOS-4.0/code/test$ ./../build.linux/nachos -rs 1023 -x cat
* * * Nhap vao ten file can doc: Nachos1.txt
* * * Mo file That bai * * *
Machine halting!

Ticks: total 239332858, idle 239330253, system 2560, user 45
Disk I/O: reads 0, writes 0
Console I/O: reads 12, writes 63
Paging: faults 0
Network I/O: packets received 0, sent 0
```

Cat, đọc nội dung của file Nachos.txt

```
ddhuu@Admin:~/nachos/NachOS-4.0/code/test$ ./../build.linux/nachos -rs 1023 -x cat
* * * Nhap vao ten file can doc: Nachos.txt
Using for Testing cat.c
Dong file thanh cong
Machine halting!

Ticks: total 184237041, idle 184233193, system 3060, user 788
Disk I/O: reads 0, writes 0
Console I/O: reads 11, writes 78
Paging: faults 0
Network I/O: packets received 0, sent 0
```

Cat, đọc nội dung của file không tồn tại

3. Viết chương trình delete

Mục đích:

- Xóa một file chỉ định do người dùng nhập từ console từ **ReadString**.

Cách thức hoạt động:

- Nhập vào tên file từ console từ thông qua system call **ReadString**
- Xóa file bằng system call **Remove** nếu kết quả trả về -1 và in ra màn hình "xóa file thất bại", ngược lại remove = 0 in ra màn hình "xóa file thành công"

Hình ảnh demo chương trình

```
ddhuu@Admin:~/nachos/NachOS-4.0/code/test$ ./../build.linux/nachos -rs 1023 -x delete
Nhap ten file can delete: Nachos.txt

Xoa file thanh cong.
Machine halting!

Ticks: total 235313433, idle 235311312, system 2070, user 51
Disk I/O: reads 0, writes 0
Console I/O: reads 11, writes 48
Paging: faults 0
Network I/O: packets received 0, sent 0
```

Delete file Nachos.txt

```
ddhuu@Admin:~/nachos/NachOS-4.0/code/test$ ./../build.linux/nachos -rs 1023 -x delete
Nhap ten file can delete: Nachos1.txt

Xoa file khong thanh cong.
Machine halting!

Ticks: total 261770513, idle 261768192, system 2270, user 51
Disk I/O: reads 0, writes 0
Console I/O: reads 12, writes 54
Paging: faults 0
Network I/O: packets received 0, sent 0
```

Delte File không tồn tại

4. Viết chương trình copy

Mục đích:

- Copy nội dung từ file nguồn sang file đích

Cách thức hoạt động:

- Nhập tên file nguồn và file đích.
- Mở file nguồn bằng system call **Open** nếu mở thành công (**OpenFileId** khác -1) tiến hành lấy kích thước của nội dung file bằng cách gọi system call **Seek** với tham số **-1 và openFileID** sau đó gọi lại **Seek** với tham số 0 để đưa con trỏ về đầu file. Trong quá trình gọi **Seek** nếu kết quả trả về -1 thì xuất thông báo lỗi và dừng chương trình
- Tiếp theo sử dụng system call **Read** để đọc file với độ dài là len đã được lấy ở bước 2. Kiểm tra nếu kết quả đọc file bằng -1 sẽ in ra màn hình lỗi đọc file và kết thúc chương trình. Sau đó đóng file nguồn bằng system call **Close**.
- Mở file đích thông qua system call **Open**. Nếu OpenFileId bằng -1 (tức là file không tồn tại) thì gọi system call **Create** để tạo mới file và tiến hành mở lại file với system call **Open.**
- Cuối cùng sử dụng system call **Write** để ghi dữ liệu từ buffer đã lưu nội dung file nguồn

ở bước 3 vào file đích dựa vào OpenFileId chỉ định. Kiểm tra ghi thất bại (kết quả trả về -1)sẽ in ra màn hình lỗi ghi dữ liệu. Sau đó, gọi **Close** đóng file đích.

Hình ảnh Demo chương trình

```
ddhuu@Admin:~/nachos/NachOS-4.0/code/test$ ./../build.linux/nachos -rs 1023 -x copy
Nhap ten file nguon: Src.txt
Nhap ten file dich: Dest.txt
Dong file nguon thanh cong

Copy du lieu thanh cong !!
Dong file thanh cong
Machine halting!

Ticks: total 716538553, idle 716533896, system 4500, user 157
Disk I/O: reads 0, writes 0
Console I/O: reads 17, writes 117
Paging: faults 0
Network I/O: packets received 0, sent 0
```

Copy từ file nguồn vào file đích thành công

```
ddhuu@Admin:~/nachos/NachOS-4.0/code/test$ ./../build.linux/nachos -rs 1023 -x copy
Nhap ten file nguon: Src.txt
Nhap ten file dich: Dest1.txt
Dong file nguon thanh cong

File khong ton tai. Tao file thanh cong !!

Copy du lieu thanh cong !!
Dong file thanh cong
Machine halting!

Ticks: total 509470893, idle 509464776, system 5930, user 187
Disk I/O: reads 0, writes 0
Console I/O: reads 18, writes 161
Paging: faults 0
Network I/O: packets received 0, sent 0
```

Copy từ file nguồn vào một file đích không tồn tại

5. Viết chương trình concatenate

Mục đích:

- Nối nội dung của 2 file với 2 tên file do người dùng nhập từ console từ **ReadString**.

Cách thức hoạt động:

- Nhập tên 2 file nguồn.
- Gọi system call **Open** với type bằng 1 (read-only). Kiểm tra xem OpenFileId trả về, nếu bằng -1 thì xuất thông báo lỗi và kết thúc chương trình

- Gọi system call **Seek** với tham số -1 để lấy kích thước file, sau đó gọi **Seek** với tham số 0 để đưa con trỏ file về đầu file. Kiểm tra nếu kết quả **Seek** trả về bằng-1 thì xuất lỗi
- Tiến hành đọc file bằng system call **Read** file nguồn kiểm tra lỗi nếu kết quả trả về bằng -1. Nếu đọc thành công thì nội dung đọc được sẽ được lưu vào mảng chỉ định. Sau đó gọi system call **Close để** đóng file
- Việc đọc file và đóng file nguồn 2 cũng tương tự file nguồn 1 như trên, nội dung đọc sẽ được lưu vào mảng chỉ định
- Tạo file chứa việc nối nội dung từ 2 file nguồn với tên file mặc định là "resultConcatenate.txt". Kiểm tra hợp lệ khi tạo file bằng system call **Create** nếu thất bại thì xuất lỗi tạo file và kết thúc chương trình.
- Tiến hành ghi file bằng cách gọi system call **Write** với tham số gồm OpenFileId của file nguồn 1, kết quả đọc byte của file nguồn 1, và mảng chứa nội dung file nguồn 1. Nếu kết quả trả về -1 thì xuất thông báo lỗi và kết thúc .
- Sau khi ghi nội dung file nguồn 1 vào file. Tiến hành gọi **Seek** để đưa con trỏ tới cuối file của file đích. Sau đó gọi system **Write** để ghi nội dung file nguồn 2 vào file với tham số tương ứng. Nếu kết quả ghi nhận được bằng -1 thì xuất thông báo lỗi
- Nếu nối file thành công (kết quả **Write** khác -1) sẽ gọi system call **Close** đóng file đích,

Hình ảnh demo chương trình

```
ddhuu@Admin:~/nachos/NachOS-4.0/code/test$ ./../build.linux/nachos -rs 1023 -x concatenate
Nhap ten file nguon 1: Src.txt
Nhap ten file nguon 2: Dest.txt
File khong ton tai. Tao file thanh cong

Noi file thanh cong
Machine halting!

Ticks: total 384109533, idle 384104497, system 4260, user 776
Disk I/O: reads 0, writes 0
Console I/O: reads 17, writes 107
Paging: faults 0
Network I/O: packets received 0, sent 0
```

Nối 2 file tồn tại

```
ddhuu@Admin:~/nachos/NachOS-4.0/code/test$ ./../build.linux/nachos -rs 1023 -x concatenate
Nhap ten file nguon 1: Src.txt
Nhap ten file nguon 2: Dest3.txt

Mo file nguon 2 that bai !!
Machine halting!

Ticks: total 770486751, idle 770483407, system 3250, user 94
Disk I/O: reads 0, writes 0
Console I/O: reads 18, writes 75
Paging: faults 0
Network I/O: packets received 0, sent 0
```

Nối file không tồn tại

6. Viết chương trình echoclient

- **Mục đích:** Client sẽ gửi một message, sau đó Server nhận message và biến chuỗi message đó thành chữ hoa sau đó gửi lại cho client. Cuối cùng Client nhận lại chuỗi message (đã in hoa), sau đó xuất ra màn hình

- **Cách thức hoạt động:**
  - Tạo một mảng gồm chứa 4 socket id
  - Dùng vòng lặp duyệt 4 lần, mỗi lần lặp:
    - Tạo một socket mới bằng cách gọi system call **SocketTCP** và thêm vào mảng socket**,** nếu kiểm kết quả nhận được từ **SocketTCP** bằng -1 thì dùng system call **PrintString** in ra màn hình lỗi tạo socket.
    - Gọi system call **Connect** với socket id vừa được tạo và lưu trong mảng socket, port và ip chỉ định, nếu kiểm tra kết quả nhận được từ **Connect** bằng -1 thì dùng system call **PrintString** in ra lỗi kết nối.
    - Gọi system call **Send** với tham số gồm socket id, message cần gửi đi và độ dài message. Nếu kiểm tra kết quả bằng -1 thì dùng system call **PrintString** xuất ra màn hình lỗi gửi message, nếu bằng 0 thì xuất ra màn hình lỗi đóng kết nối, ngược lại lưu lại kết quả nhận được
    - Gọi system call **Receive** với tham số gồm socket id, mảng chứa message nhận được và độ dài bytes gửi đi. Nếu kiểm tra kết quả bằng -1 thì dùng system call **PrintString** xuất ra màn hình lỗi nhận message, nếu bằng 0 thì xuất ra màn hình lỗi đóng kết nối, ngược lại xuất ra màn hình message nhận được.
  - Dùng lại vòng lặp duyệt 4 lần, mỗi lần lặp sẽ đóng từng socket bằng cách gọi system call **Close** với socket id đã được lưu trong mảng socket. Kiểm tra nếu kết quả nhận được bằng 0 thì in ra thông báo đóng socket thành công, ngược lại, in ra thông báo lỗi đóng socket thất bại

Hình ảnh Demo chương trình

```
ddhuu@Admin:~/nachos/NachOS-4.0/code/test$ ../build.linux/nachos -rs 1023 -x
echoclient
Nhap message can gui toi server: ConnectToSocket
CONNECTTOSOCKET
CONNECTTOSOCKET
CONNECTTOSOCKET
CONNECTTOSOCKET
Dong Socket thanh cong
Dong Socket thanh cong
Dong Socket thanh cong
Dong Socket thanh cong
Machine halting!

Ticks: total 1346250000, idle 1346242463, system 6920, user 617
Disk I/O: reads 0, writes 0
Console I/O: reads 16, writes 189
Paging: faults 0
Network I/O: packets received 0, sent 0
ddhuu@Admin:~/nachos/NachOS-4.0/code/test$
```

Client

```
ddhuu@Admin:~/nachos/NachOS-4.0/code/test/server$ ./server
waiting for clients
Accepted new connection from a client 127.0.0.1:34530
received ConnectToSocket from client
waiting for clients
Accepted new connection from a client 127.0.0.1:34532
received ConnectToSocket from client
waiting for clients
Accepted new connection from a client 127.0.0.1:34548
received ConnectToSocket from client
waiting for clients
Accepted new connection from a client 127.0.0.1:34564
received ConnectToSocket from client
waiting for clients
```

Server

7. Viết chương trình fileclient

- **Mục đích:** Client sẽ đọc và gửi nội dung file (file text) cho Server, sau đó Server nhận message và biến thành chữ hoa, sau đó gửi lại cho client. Cuối cùng, Client nhận lại chuỗi message (đã in hoa), sau đó ghi lại vào file
- **Cách thức hoạt động:**
  - Mở một file txt chỉ định do người dùng nhập. Kiểm tra nếu kết quả trả về -1 thì dùng

system call **PrintString** xuất màn hình lỗi mở file và gọi system call **Halt** để kết thúc.

- Tạo một socket mới bằng cách gọi system call **SocketTCP,** nếu kiểm kết quả nhận được từ **SocketTCP** bằng -1 thì dùng system call **PrintString** in ra màn hình lỗi tạo socket.
- Gọi system call **Connect** với socket id vừa được tạo, port và ip chỉ định, nếu kiểm tra kết quả nhận được bằng -1 thì dùng system call **PrintString** in ra lỗi kết nối.
- Gọi system call **Send** với tham số gồm socket id, message cần gửi đi và độ dài message. Nếu kiểm tra kết quả bằng -1 thì xuất ra màn hình lỗi gửi message, nếu bằng 0 thì xuất ra màn hình lỗi đóng kết nối, ngược lại lưu lại kết quả nhận được
- Gọi system call **Receive** với tham số gồm socket id, mảng chứa message nhận được và độ dài bytes gửi đi. Nếu kiểm tra kết quả bằng -1 thì dùng system call **PrintString** xuất ra màn hình lỗi nhận message, nếu bằng 0 thì xuất ra màn hình lỗi đóng kết nối, ngược lại message nhận được sẽ được lưu vào mảng và số byte nhận được sẽ lưu vào biến chỉ định.
- Nhập tên file từ màn hình console, gọi system call **Open** để mở file nếu file không tồn tại thì gọi system call **Create** để tạo mới file, nếu tạo thất bại thì xuất ra màn hình lỗi tạo file
- Gọi system call **Write** để ghi message nhận được vào file chỉ định
- Gọi system call **Close** để đóng file, xuất ra màn hình nếu đóng thành công hoặc thất bại

Hình ảnh Demo chương trình

```
ddhuu@Admin:~/nachos/NachOS-4.0/code/test$ ./../build.linux/nachos -rs 1023 -x fileclient
Nhap ten file can doc: Src.txt
Gui message thanh cong den server.
Nhan message thanh cong tu server. Noi dung nhan:
WORKING WITH NACHOS 4.0
 * * * Nhap ten file de ghi message tu server: Dest.txt
Da ghi messsage vao file: Dest.txt
Dong file thanh cong
Machine halting!

Ticks: total 701968192, idle 701958766, system 8460, user 966
Disk I/O: reads 0, writes 0
Console I/O: reads 17, writes 236
Paging: faults 0
Network I/O: packets received 0, sent 0
```

Client

```
ddhuu@Admin:~/nachos/NachOS-4.0/code/test/server$ ./server
waiting for clients
Accepted new connection from a client 127.0.0.1:49964
received Working with Nachos 4.0 from client
waiting for clients
```

Server

IV. Nguồn tham khảo

- Các tài liệu trên Moodle
- Tìm hiểu về các System call GIẢI THÍCH VÀ CÀI ĐẶT CÁC SYSTEM CALL TRÊN NACHOS