

**TRƯỜNG ĐẠI HỌC KHOA HỌC TỰ NHIÊN
KHOA CÔNG NGHỆ THÔNG TIN**



Nguyễn Tạ Bảo - 21120205

Lê Hữu Hưng - 21120463

Nguyễn Thế Phong - 21120527

BÁO CÁO
PROJECT 01: TÌM HIỂU SYSTEM
CALLS - CÁC THAO TÁC VỚI FILE VÀ
NETWORK

GIÁO VIÊN HƯỚNG DẪN

ThS. Lê Giang Thanh

ThS. Nguyễn Thanh Quân

Tp. Hồ Chí Minh, tháng 11/2023

Mục lục

Đề cương chi tiết	i
Mục lục	i
Tóm tắt	v
1 Giới thiệu	1
2 Cài đặt System call	2
2.1 Cài đặt tổng quan	2
2.2 Cài đặt System call Create	2
2.2.1 Yêu cầu	2
2.2.2 Mô tả:	3
2.2.3 Cách cài đặt	3
2.3 Cài đặt system call OpenFileID Open	4
2.3.1 Yêu cầu	4
2.3.2 Mô tả	4
2.3.3 Cách cài đặt	4
2.4 Cài đặt System call OpenFileID id Close	5
2.4.1 Yêu cầu	5
2.4.2 Mô tả	5
2.4.3 Cách cài đặt	6
2.5 Cài đặt System call Read	6
2.5.1 Yêu cầu	6
2.5.2 Mô tả	6
2.5.3 Cách cài đặt	7
2.6 Cài đặt System call Write	8
2.6.1 Yêu cầu	8
2.6.2 Mô tả input output :	8
2.6.3 Cách cài đặt :	8

2.7	Cài đặt System call Seek	9
2.7.1	Yêu cầu:	9
2.7.2	Mô tả cài đặt :	9
2.7.3	Cách cài đặt :	10
2.8	Cài đặt System call Remove	10
2.8.1	Yêu cầu :	10
2.8.2	Mô tả	10
2.8.3	cách cài đặt :	11
2.9	Cài đặt System call SocketTCP	11
2.9.1	Yêu cầu	11
2.9.2	Mô tả	11
2.9.3	Cách cài đặt	12
2.10	Cài đặt System call Connect	12
2.10.1	Yêu cầu	12
2.10.2	Mô tả	12
2.10.3	Cách cài đặt	13
2.11	Cài đặt System call Send	14
2.11.1	Yêu cầu	14
2.11.2	Mô tả	14
2.11.3	Cách cài đặt	15
2.12	Cài đặt System call Receive	16
2.12.1	Yêu cầu	16
2.12.2	Mô tả	16
2.12.3	Cách cài đặt	17
2.13	Cài đặt System call Close_Socket	17
2.13.1	Yêu cầu	18
2.13.2	Mô tả	18
2.13.3	Cách cài đặt	18
3	Cài đặt chương trình Test	19
3.1	Chương trình createfile: tạo ra một file mới	19
3.1.1	Demo minh họa	19
3.2	Chương trình cat: hiện nội dung của file	20
3.2.1	Demo minh họa	20

3.3	Chương trình copy: copy một file nguồn thành một file mới	21
3.3.1	Demo minh họa	22
3.4	Chương trình delete: Xóa file	23
3.4.1	Ý tưởng	23
3.4.2	Demo minh họa	23
3.5	Chương trình concatenate: Nối nội dung 2 file	24
3.5.1	Ý tưởng	24
3.5.2	Demo minh họa	24
3.6	Chương trình echo	25
3.6.1	Ý tưởng	25
3.6.2	Demo minh họa	25
3.7	Chương trình truyền file	27
3.7.1	Ý tưởng	27
3.7.2	Demo minh họa	28
4	Kết quả đạt được	31
	Tài liệu tham khảo	32

Danh sách hình

3.1	Các file trong thư mục (sắp xếp theo bảng chữ cái) trước khi chạy	19
3.2	Nội dung chương trình khi biên dịch tạo file a_test.txt . .	19
3.3	File thư mục sau khi tạo a.txt	20
3.4	Nội dung của file a_test.txt	20
3.5	Chạy chương trình biên dịch để đọc nội dung file	21
3.6	Cây thư mục ban đầu không có file b_test.txt	22
3.7	Chạy chương trình biên dịch để tạo file b_test.txt và chép nội dung file a_test.txt vào	23
3.8	Nội dung của file b_test.txt sau khi được copy	23
3.9	Nội dung biên dịch và kết quả sau khi biên dịch	23
3.10	Nội dung 2 file	24
3.11	Nội dung biên dịch và file sau khi biên dịch	24
3.12	Màn hình bắt đầu khởi chạy chương trình	25
3.13	Nhập nội dung muốn gửi	26
3.14	Kết quả trả về từ chương trình	27
3.15	Nội dung của file cần gửi đi (fileclienttest.txt)	28
3.16	Màn hình bắt đầu khởi chạy chương trình	28
3.17	Thực hiện nhập tên file muốn gửi đi, sau đó nhấn Enter . .	29
3.18	Nhận từ Server thành công. Thực hiện nhập tên file cần xuất. Kết thúc chương trình.	29
3.19	Nội dung của file sau khi được xuất bởi chương trình. . . .	30

Danh sách bảng

4.1	Bảng kết quả đạt được của đề án	31
-----	---	----

Chương 1

Giới thiệu

Nachos là một phần mềm hướng dẫn được thiết kế để cung cấp cho sinh viên cơ hội nghiên cứu và điều chỉnh một hệ điều hành thực sự. Sự khác biệt chính giữa Nachos và hệ điều hành "thực sự" nằm ở việc Nachos hoạt động như một quy trình Unix duy nhất, so với việc chạy trực tiếp trên phần cứng vật lý như các hệ điều hành truyền thống. Tuy nhiên, Nachos mô phỏng những chức năng cấp thấp cơ bản của các máy thông thường, bao gồm ngắt, bộ nhớ ảo và I/O dựa trên ngắt (exception).

Đồ án này bao gồm việc lập trình System call cơ bản và viết các chương trình Test để kiểm tra chức năng của các System call này, hiểu rõ cách mà hệ thống hoạt động thông qua việc tìm hiểu, triển khai và kiểm thử các chức năng hệ điều hành cơ bản.

Bài báo cáo này trình bày cụ thể các nội dung như sau: Chương 1: Giới thiệu - Đây là phần giới thiệu cơ bản về Nachos, mô tả về mục tiêu của đồ án, cũng như các khái niệm cơ bản về hệ thống. Chương 2: Cài đặt System call - Tiến hành triển khai các System call cơ bản, từ việc định nghĩa đến việc thực hiện chúng trong Nachos. Chương 3: Cài đặt chương trình Test - Kiểm tra và đánh giá tính năng của các system call đã triển khai. Chương 4: Kết quả đạt được - Trình bày tổng quan các phần đã đạt được trong đồ án.

Chương 2

Cài đặt System call

2.1 Cài đặt tổng quan

Trước khi đi đến cài đặt chi tiết các system call thì ta sẽ cài đặt các hàm cần thiết cho các system call:

1. `char *User2System(int virtAddr, int limit)`: Sao chép buffer từ không gian bộ nhớ người dùng sang không gian bộ nhớ hệ thống.
2. `int System2User(int virtAddr, int len, char *buffer)`: Sao chép buffer từ không gian bộ nhớ hệ thống sang không gian bộ nhớ người dùng.
3. `void move_program_counter()`: Dùng để tăng địa chỉ thanh ghi PC lên 4 byte, nạp lệnh tiếp theo để thực thi.

Ngoài các hàm trên, để quản lí file và network trong một mảng duy nhất thì ta sẽ tạo 1 struct chứa `file_id` và `socket_id` trong `filesys.h`

```
1 struct FD_Table
2 {
3     OpenFile *fileID;
4     int socketID;
5 };
```

2.2 Cài đặt System call Create

2.2.1 Yêu cầu

Cài đặt system call `int Create(char*name)`. Create system call sẽ sử dụng Nachos FileSystem Object để tạo một file rỗng. Chú ý rằng filename

đang ở trong user space, có nghĩa là buffer mà con trỏ trong user space trỏ tới phải được chuyển từ vùng nhớ user space tới vùng nhớ system space. System call Create trả về 0 nếu thành công và -1 nếu có lỗi

2.2.2 Mô tả:

```
int Create(char*name)
```

Tạo ra file rỗng với tham số là tên file.

Parameter

name: `char*`

Tên file bạn muốn tạo.

Return

out: `int`

0 nếu tạo file thành công, 1 nếu tạo file thất bại.

2.2.3 Cách cài đặt

System call `SC_Create` được cài đặt để xử lý tác vụ này. Nó thực hiện các thao tác sau:

1. Đọc địa chỉ của tham số name từ thanh ghi thứ r4.
2. Sao chép name từ vùng nhớ người dùng sang vùng nhớ hệ thống để xử lý.
3. Hàm `SysCreate`(được khởi tạo trong file `ksyscall.h`) được gọi để thực hiện quá trình tạo file.
 - Kiểm tra xem file name có NULL không. Kiểm tra file bạn muốn tạo đã có có trước đó hay chưa bằng hàm `doesFileExist(char *name)`.
 - Nếu thỏa mãn 2 điều kiện trên thì dùng `kernel->fileSystem->Create(filename, 0)` để tạo file.
 - Ta sẽ trả về 0 khi tạo file thành công, -1 là khi ngược lại với các điều kiện trên.

2.3 Cài đặt system call OpenFileID Open

2.3.1 Yêu cầu

System call `OpenFileID Open(char*name, int type)` cho phép người dùng có thể mở 2 kiểu file (type), file chỉ có thể đọc (read only), và file có thể đọc và viết (read and write). Bạn xây dựng một mảng các file descriptor table với kích thước là 20 file descriptors. 2 phần tử đầu tiên 0 và 1 sẽ không được sử dụng, cho mục đích console input và console output tương ứng. System call sẽ chịu trách nhiệm chuyển đổi buffer của user space khi cần thiết và cấp phát tương ứng dưới kernel. Sử dụng file system objects được cung cấp trong filesys folder.

Hàm system call `Open` sẽ trả về file descriptor id (`OpenFileID` là một số nguyên integer), hoặc -1 nếu lỗi.

2.3.2 Mô tả

```
OpenFileID Open(char*name, int type)
```

Dùng để mở file theo 2 kiểu: Read Only và Read-Write.

Parameter:

name: `char*`

Tên file bạn muốn mở

type: `int`

Kiểu file (0: Read Only, 1: Read and Write)

Return:

out: `int`

`OpenFileID` nếu mở file thành công, -1 nếu mở file thất bại.

2.3.3 Cách cài đặt

System call `SC_Open` được cài đặt để xử lý tác vụ này. Nó thực hiện các thao tác sau:

1. Đọc địa chỉ của tham số name từ thanh ghi thứ r4 và type từ thanh ghi r5. Sau đó, sao chép name từ vùng nhớ người dùng sang vùng nhớ hệ thống.
2. Kiểm tra xem liệu FD_Tabel còn trống không bằng FindFreeSlot(). Nếu FD_Tabel còn trống, kiểm tra xem liệu type có bằng 0 hoặc 1 không. Cuối cùng là kiểm tra xem file đã được mở hay chưa, nghĩa là nó có trong FD_Table hay không bằng checkIfFileOpened(char *name).
3. Nếu đã thỏa mãn 3 điều kiện trên thì ta dùng kernel->fileSystem->Open(char *name, int type) để mở file, và dùng addFilenameToOpenset(int freeSlot, char *fileName) để thêm file vừa mở vào danh sách file đã được mở.
4. trả về OpenFileID(chính là index còn trống đầu tiên trong FD_Table) nếu trường hợp file không tồn tại hay ngược lại với các trường hợp trên thì trả về -1.

2.4 Cài đặt System call OpenFileID id Close

2.4.1 Yêu cầu

Hàm system call `int Close(OpenFileID id)` sẽ truyền vào OpenFileID và sẽ trả về -1 nếu lỗi và 0 nếu thành công.

2.4.2 Mô tả

```
int Close(OpenFileID id)
```

Đóng file đang mở.

Parameter:

id: OpenFileID

OpenFileID của file mà bạn muốn xóa

Return:

out: int

0 nếu đóng file thành công, -1 nếu đóng file thất bại.

2.4.3 Cách cài đặt

1. Đọc id từ thanh ghi r4.
2. Kiểm tra id có nằm trong các id có thể có hay không ($0 \leq id \leq 20$), nếu có thì xóa vùng nhớ được dùng để lưu trữ file và gán nó bằng NULL. Ngoài ra, ta còn phải gán `kernel->fileSystem->openedFileName[fid] = NULL` để xóa nó khỏi danh sách nhưng file đã được mở.
3. Trả về 0 khi đã đóng file thành công, 1 nếu đóng file thất bại.

2.5 Cài đặt System call Read

2.5.1 Yêu cầu

Các system call `int Read(char *buffer, int size, OpenFileID id)` đọc file với id cho trước. Bạn cần phải chuyển vùng nhớ giữa user space và system space, và cần phải phân biệt giữa file và console in. Trong trường hợp Console là read console, sử dụng `SynchConsoleInput` class cho việc read, bạn phải đảm bảo trả đúng dữ liệu cho người dùng. Read vào console sẽ trả đúng số lượng ký tự được read, không phải là charcount. Trong trường hợp read bị lỗi, trả về -1.

2.5.2 Mô tả

```
int Read(char *buffer, int size, OpenFileID id)
```

Nếu $id \geq 2$ thì đọc charcount kí tự từ file có OpenFileID là id, sau đó gán vào buffer. Nếu $id = 0$, tức là Console In thì trả về đúng số ký tự được read, không phải charcount.

Parameter:

buffer: `char*`

buffer để chứa nội dung đọc được từ file hay console in.

charcount: `int`

Số kí tự cần đọc trong file

id: `OpenFileID`

`OpenFileID` của file(hoặc console in) bạn muốn đọc.

Return:

out: `int`

Số kí tự đọc được nếu thành công, -1 nếu đọc thất bại.

2.5.3 Cách cài đặt

System call `SC_Read` được cài đặt để xử lý tác vụ này. Nó thực hiện các thao tác sau:

1. Đọc địa chỉ tham số buffer từ thanh ghi r4, charcount từ thanh ghi r5, id từ thanh ghi r6.
2. Kiểm tra id nằm trong `FD_Table` không, file có tồn tại không bằng `kernel->fileSystem->table[id].fileID == NULL` và id khác 1(file không là console output) không.
3. Nếu `id = 0`, tức đây là `consolin`. Đầu tiên ta khởi tạo biến count để đếm số kí tự mà ta đọc được và `_stdinBuffer` dùng để lưu các kí tự đọc được. Tiếp theo, ta dùng `kernel->synchConsoleIn->GetChar()` để đọc từng kí tự trong được mà mình đã nhập cho đến khi gặp kí tự **Enter**.
4. Nếu `id ≥ 2` thì đầu tiên ta lấy vị trí con trỏ(`OldPos`) trước khi đọc file. Sau đó, sao chép buffer từ vùng nhớ người dùng sang vùng nhớ hệ thống với kích thức là charcount. Dùng `kernel->fileSystem->table[id].fileID->Read(buf, charcount)` để đọc file, tính vị trí con trỏ(`NewPos`) sau khi đọc. Tính vị trí con trỏ bằng `GetCurrentPos()`. Số kí tự đọc được là `NewPos - OldPos`.
5. Trả về số kí tự đọc được nếu đọc thành công và -1 nếu đọc thất bại

2.6 Cài đặt System call Write

2.6.1 Yêu cầu

Cài đặt system call `int Write(char *buffer, int size, OpenFileID id)`. Các System call `Write` ghi vào file với id cho trước. Bạn cần phải chuyển vùng nhớ giữa User space và System space, và cần phải phân biệt giữa Console IO (`OpenFileID 0, 1`) và File. Trong trường hợp Console là write console, sử dụng `SynchConsoleOutput` class cho việc write, bạn phải đảm bảo trả đúng dữ liệu cho người dùng. write vào console sẽ trả đúng số lượng ký tự được write, không phải là `charcount`. Trong trường hợp write bị lỗi, trả về -1.

2.6.2 Mô tả input output :

```
int Write(char *buffer, int size, OpenFileID id) |
```

Ghi file với tham số là buffer, số ký tự cho phép và id của file

Parameter

buffer : `char*`

Chuỗi kí tự muốn ghi vào file

size : `int`

Lượng kí tự tối đa muốn ghi vào

OpenFileID : `int`

id của file tính ghi vào.

Return

out : `int`

Trả về -1 nếu lỗi , số byte write thực sự nếu thành công

2.6.3 Cách cài đặt :

1. Lấy địa chỉ của tham số buffer từ thanh ghi số 4.

2. Lấy charcount từ thanh ghi số 5.
3. Lấy id của file từ thanh ghi số 6.
4. Kiểm tra id của file truyền vào có nằm ngoài bảng mô tả file hay không.
5. Xét trường hợp ghi file only read (type quy ước là 1) hoặc file stdin (type quy ước là 2) thì trả về -1
6. Kiểm tra file có tồn tại hay không
7. Kiểm tra thành công thì lấy vị trí OldPos
8. Copy chuỗi từ vùng nhớ User Space sang System Space với bộ đệm buffer dài charcount
9. Xét trường hợp ghi file read và write (type quy ước là 0)

2.7 Cài đặt System call Seek

2.7.1 Yêu cầu:

Cài đặt system call `int Seek(int position, OpenFileID id)`. Seek sẽ phải chuyển con trỏ tới vị trí thích hợp. position lưu vị trí cần chuyển tới, nếu pos = -1 thì di chuyển đến cuối file. Trả về vị trí thực sự trong file nếu thành công và -1 nếu bị lỗi. Gọi Seek trên console phải báo lỗi.

2.7.2 Mô tả cài đặt :

```
int Seek(int position, OpenFileID id)
```

Di chuyển con trỏ đến vị trí thích hợp trong file với tham số là vị trí cần chuyển tới và id của file.

Parameter

position : int

Vị trí cần chuyển con trỏ đến trong file

id : `int`

Id của file

Return

out : `int`

Trả về -1 nếu lỗi , vị trí thực sự nếu thành công

2.7.3 Cách cài đặt :

1. Lấy vị trí cần chuyển con trỏ đến trong file.
2. Lấy id của file.
3. Kiểm tra id của file truyền vào có nằm ngoài bảng mô tả file không.
4. Kiểm tra file có tồn tại không.
5. Kiểm tra có gọi **Seek** trên console không.
6. Nếu $pos = -1$ thì gán $pos = Length$, ngược lại thì giữ nguyên pos .
7. Kiểm tra lại vị trí pos có hợp lệ không.
8. Nếu hợp lệ thì trả về vị trí di chuyển thực sự trong file.

2.8 Cài đặt System call Remove

2.8.1 Yêu cầu :

Cài đặt system call `int Remove(char *name)`. Remove system call sẽ sử dụng Nachos FileSystem Object để xóa file. Chú ý: cần kiểm tra file có đang mở hay không trước khi xóa.

2.8.2 Mô tả

Xóa file với tên file cho trước

```
int Remove(char *name)
```


- Input : tên file
- Output : 0 nếu xóa thành công, -1 nếu thất bại.

`int Remove(char *name)`

Parameter

name : `char*`

Tên File muốn xóa

Return

out : `int`

Trả về -1 nếu lỗi , 0 nếu thành công

2.8.3 cách cài đặt :

1. Kiểm tra id của file truyền vào có nằm ngoài bảng mô tả file không.
2. Tiến hành xóa file bằng hàm `Remove` trong `filesystem.cc`

2.9 Cài đặt System call SocketTCP

2.9.1 Yêu cầu

Cài đặt System call `int SocketTCP()`. Bạn xây dựng một mảng các File Descriptor Table với kích thước là 20 File descriptors.

Hàm System call `SocketTCP` sẽ trả về File descriptor ID (`int` là một số nguyên integer), hoặc -1 nếu lỗi.

2.9.2 Mô tả

```
int SocketTCP()
```

Khởi tạo 1 Socket, sau đó trả về File descriptor ID của Socket đó. File descriptor ID của socket này đóng vai trò là đối số của các hàm Network TCP được nhắc đến tiếp theo.

Return

id : `int`

Trả về id là File descriptor ID của Socket nếu được khởi tạo thành công. Nếu thất bại trả về -1.

2.9.3 Cách cài đặt

System call `SC_SocketTCP` được cài đặt để xử lý tác vụ này. Nó thực hiện các thao tác sau:

1. Tìm vị trí trống trong File descriptor table (`FD_Table`) bằng hàm `FindFreeSocketTableSlot(table)` để lưu trữ Socket ID mới. Nếu không có vị trí trống nào trong Table thì ghi giá trị -1 vào thanh ghi trả về. Kết thúc System call.
2. Socket mới sẽ được tạo bằng cách sử dụng hàm `codesocket(AF_INET, SOCK_STREAM, 0)`. Nếu tạo Socket thất bại thì ghi giá trị -1 vào thanh ghi trả về. Kết thúc System call.
3. Nếu Socket được tạo thành công, ID của Socket được gán vào thanh ghi trả về. Ngược lại, nếu thất bại thì ghi giá trị -1 vào thanh ghi trả về.
4. Kết thúc System call.

2.10 Cài đặt System call Connect

2.10.1 Yêu cầu

Cài đặt System call `int Connect(int socketid, char *ip, int port)`. Connect đến Server theo thông tin IP và Port. Trả về 0 nếu kết nối thành công, và -1 nếu bị lỗi.

2.10.2 Mô tả

```
int Connect(int socketid, char *ip, int port)
```

Kết nối đến Server theo IP và Port, sử dụng Socket đã được khởi tạo trước.

Parameter

`socketid : int`

File descriptor ID của Socket đã được khởi tạo trước đó nhờ hàm `SocketTCP`.

`ip : char *`

Địa chỉ IP của Server muốn kết nối tới.

`port : int`

Port của Server muốn kết nối tới.

Return

`out : int`

Trả về 0 nếu kết nối thành công, nếu kết nối thất bại thì trả về -1.

2.10.3 Cách cài đặt

1. Đọc cách thông số `socketid`, `addIp`, `ServerPort` lần lượt từ các thanh ghi 4, 5, 6:
 - `socketid`: ID của Socket cần kết nối.
 - `addIp`: Địa chỉ IP của Server sẽ kết nối tới.
 - `ServerPort`: Port của Server.
2. Hàm `User2System` được sử dụng để chuyển địa chỉ IP từ User space sang Kernel space.
3. Hàm `SysConnect` (được khởi tạo trong file `ksyscall.h`) được gọi để thực hiện quá trình kết nối đến Server thông qua địa chỉ IP và Port đã cung cấp, dựa trên `socketid`
 - Khởi tạo một biến `remote` có kiểu `sockaddr_in` để lưu thông tin về địa chỉ và Port của Server.
 - Gán địa chỉ IP của Server thông qua hàm `inet_addr`, cổng và family `AF_INET`.

- Sử dụng hàm `connect` đã được khởi tạo sẵn từ API của Linux để kết nối tới Server thông qua `socketid` và thông tin trong cấu trúc `remote`
 - Nếu kết nối không thành công (`iRetVal < 0`), ghi giá trị -1 vào thanh ghi trả về. Ngược lại nếu thành công thì ghi giá trị 0 vào thanh ghi trả về.
4. Giải phóng bộ nhớ đã cấp phát cho các biến đã khởi tạo. Kết thúc System call.

2.11 Cài đặt System call Send

2.11.1 Yêu cầu

Cài đặt System call `int Send(int socketid, char *buffer, int len)`. Gửi dữ liệu từ Socket.

- Trả về số lượng bytes gửi đi nếu thành công.
- Nếu kết nối bị đóng trả về 0.
- Nếu thất bại thì trả về -1.

2.11.2 Mô tả

```
int Send(int socketid, char *buffer, int len)
```

Gửi dữ liệu đến Server Socket, sử dụng Socket đã được khởi tạo trước.

Parameter

`socketid` : `int`

File descriptor ID của Socket đã được khởi tạo trước đó nhờ hàm `SocketTCP`.

`buffer` : `char *`

Dữ liệu cần gửi tới Server.

`len` : `int`

Độ dài của dữ liệu cần gửi tới Server.

Return

out : `int`

- Trả về số lượng bytes gửi đi nếu thành công.
- Nếu kết nối bị đóng thì trả về 0.
- Nếu thất bại thì trả về -1.

2.11.3 Cách cài đặt

1. Đọc các thông số `socketid`, `addContent`, `sizeContent` lần lượt từ các thanh ghi 4, 5, 6.
 - `socketid`: ID của Socket cần gửi dữ liệu.
 - `addContent`: Địa chỉ của buffer cần gửi.
 - `sizeContent`: Kích thước của buffer cần gửi.
2. Chuyển dữ liệu từ User space sang Kernel space bằng hàm `User2System` thông qua con trỏ `contentSend`
3. Thiết lập thời gian chờ gửi dữ liệu bằng hàm `setsockopt`. Nếu thiết lập thời gian chờ thất bại thì ghi giá trị -1 vào thanh ghi trả về. Kết thúc System call.
4. Hàm `send` được sử dụng để gửi dữ liệu thông qua `socketid`, buffer được lưu trong `contentSend` và kích thước nội dung `sizeContent`. Giá trị trả về từ hàm `send` sẽ được lưu vào biến `shortRetval`.
5. Kiểm tra trạng thái kết nối thông qua giá trị mà hàm `send` trả về và `errno`. Nếu Server đóng thì ghi giá trị 0 vào thanh ghi trả về. Kết thúc System call.
6. Xoá các vùng nhớ đã được khởi tạo.
7. Giá trị `shortRetval` được ghi vào thanh ghi trả về. Kết thúc System call.

2.12 Cài đặt System call Receive

2.12.1 Yêu cầu

Cài đặt System call `int Receive(int socketid, char *buffer, int len)`. Nhận dữ liệu từ Socket.

- Trả về số lượng bytes gửi đi nếu thành công.
- Nếu kết nối bị đóng trả về 0.
- Nếu thất bại thì trả về -1.

2.12.2 Mô tả

```
int Receive(int socketid, char *buffer, int len)
```

Nhận dữ liệu từ Server Socket, sử dụng Socket đã được khởi tạo trước.

Parameter

`socketid` : `int`

File descriptor ID của Socket đã được khởi tạo trước đó nhờ hàm `SocketTCP`.

`buffer` : `char *`

Dữ liệu nhận được từ Server.

`len` : `int`

Độ dài của dữ liệu được nhận từ Server.

Return

`out` : `int`

- Trả về số lượng bytes gửi đi nếu thành công.
- Nếu kết nối bị đóng thì trả về 0.
- Nếu thất bại thì trả về -1.

2.12.3 Cách cài đặt

1. Đọc các thông số `socketid`, `addContent`, `sizeContent` lần lượt từ các thanh ghi 4, 5, 6.
 - `socketid`: ID của Socket cần nhận dữ liệu.
 - `addContent`: Địa chỉ của buffer để lưu dữ liệu sau khi nhận được.
 - `sizeContent`: Kích thước của buffer muốn nhận.
2. Chuyển dữ liệu từ User space sang Kernel space bằng hàm `User2System` thông qua con trỏ `contentSend`
3. Thiết lập thời gian chờ nhận dữ liệu bằng hàm `setsockopt`. Nếu thiết lập thời gian chờ thất bại thì ghi giá trị -1 vào thanh ghi trả về. Kết thúc System call.
4. Hàm `recv` được sử dụng để nhận dữ liệu thông qua `socketid`, buffer được lưu trong `contentReceive` và kích thước nội dung `sizeContent`. Giá trị trả về từ hàm `recv` sẽ được lưu vào biến `shortRetval`.
 - Nếu nhận được dữ liệu, hệ thống sẽ chuyển dữ liệu chứa trong `contentReceive` từ Kernel space sang User space bằng hàm `System2User`.
5. Kiểm tra trạng thái kết nối thông qua giá trị mà hàm `recv` trả về và `errno`. Nếu Server đóng thì ghi giá trị 0 vào thanh ghi trả về. Kết thúc System call.
6. Xoá các vùng nhớ đã được khởi tạo.
7. Giá trị `shortRetval` được ghi vào thanh ghi trả về. Kết thúc System call.

2.13 Cài đặt System call `Close_Socket`

Ghi chú: Trong yêu cầu của đề án, tên System call là `Close`, nhưng trong đề án này, tên của System call đã được đổi thành `Close_Socket`

nhằm tránh sự nhập nhằng với System call `Close` đã được tạo trước đó cho tác vụ xử lý file.

2.13.1 Yêu cầu

Cài đặt System call `int Close_Socket(int socketid)`. Đóng socket với 0 là thành công, -1 là lỗi

2.13.2 Mô tả

```
int Close_Socket(int socketid)
```

Đóng Socket đã khởi tạo (nếu tồn tại).

Return

socketid : `int`

ID của Socket cần đóng.

2.13.3 Cách cài đặt

System call `SC_Close_Socket` được cài đặt để xử lý tác vụ này. Nó thực hiện các thao tác sau:

1. Đọc ID của socket cần đóng từ thanh ghi, sau đó lưu vào biến `id`.
2. Hàm `close(id)` được gọi để thực hiện đóng Socket với ID đã cung cấp.
3. Nếu việc đóng thành công (`iRetVal >= 0`), ghi giá trị 0 vào thanh ghi trả về. Ngược lại, nếu không thành công thì ghi giá trị -1 vào thanh ghi trả về.
4. Kết thúc System call.

Chương 3

Cài đặt chương trình Test

3.1 Chương trình createfile: tạo ra một file mới

- Dùng hàm PrintString để in ra các hướng dẫn cho người dùng
- Đầu tiên người dùng sẽ nhập vào tên file
- Tên file sẽ được bỏ vào hàm Create.
- Nếu hàm Create trả về 0, tức là file đã được tạo thành công và ta thông báo người dùng là file đã được tạo thành công.
- Nếu như Create không trả về 0 thì file đã không được tạo và ta sẽ thông báo lại cho người dùng rằng file đã được tạo không thành công

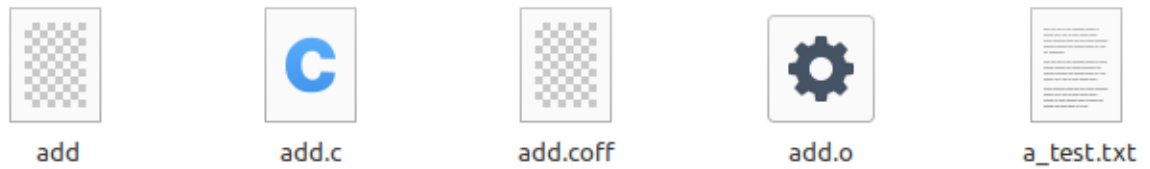
3.1.1 Demo minh họa



Hình 3.1: Các file trong thư mục (sắp xếp theo bảng chữ cái) trước khi chạy

```
• (base) lhh1323@lhh1323-Nitro-AN515-57:~/nachos/NachOS-4.0/code/test$ ../build.linux
/nachos -x createfile
===== CREATE FILE PROGRAM =====
Enter file's name: a_test.txt
File a_test.txt created successfully!
Machine halting!
```

Hình 3.2: Nội dung chương trình khi biên dịch tạo file a_test.txt



Hình 3.3: File thư mục sau khi tạo a.txt

3.2 Chương trình cat: hiện nội dung của file

- Dùng hàm `PrintString` để in ra các hướng dẫn cho người dùng
- Dùng hàm `Read` để đọc tên file.
- Hàm `Open` để mở file với tên file là tham số.
- Nếu như `Open` trả về -1, tức là file mở không thành công.
- Nếu như `Open` trả về khác -1, dùng hàm `Read`, sử dụng biến `char*` buffer để chứa thông tin của file đọc vào.
- Dùng hàm `PrintString` để in ra nội dung của buffer.

3.2.1 Demo minh họa

```
code > test > ≡ a_test.txt
1  Chuc thay mot ngay tot lanh.
```

Hình 3.4: Nội dung của file a_test.txt

```
● (base) lhh1323@lhh1323-Nitro-AN515-57:~/nachos/NachOS-4.0/code/test$ ../build.linux
/nachos -x cat
===== CAT PROGRAM FILE NAME =====

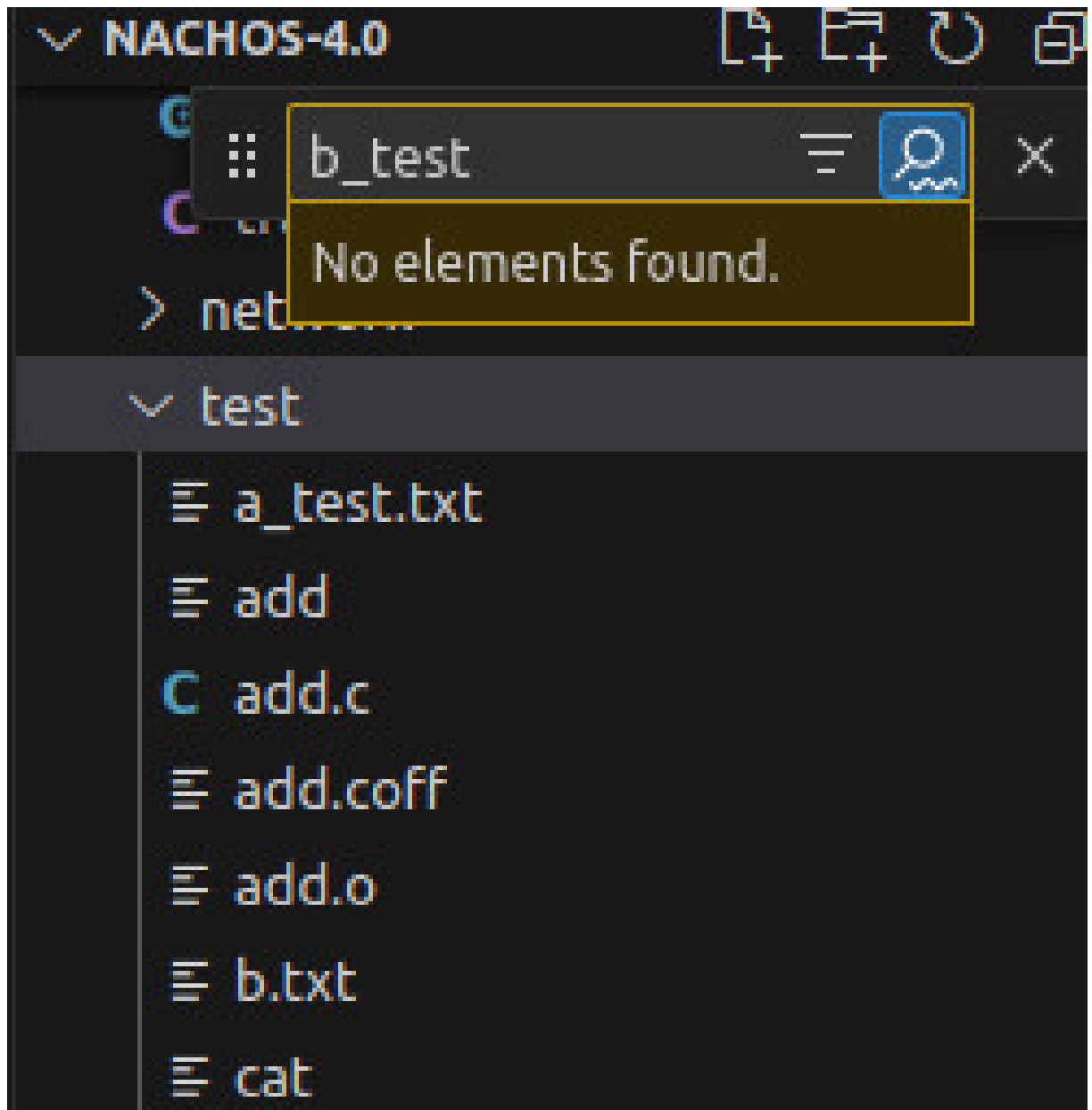
Enter file name: a_test.txt
File content:
<<<< BEGIN >>>>
Chuc thay mot ngay tot lanh.
<<<< END OF FILE >>>>
===== EXIT PROGRAM =====
Machine halting!
```

Hình 3.5: Chạy chương trình biên dịch để đọc nội dung file

3.3 Chương trình copy: copy một file nguồn thành một file mới

- Đầu tiên ta sẽ yêu cầu người dùng nhập vào tên file mà ta muốn copy.
- Sử dụng hàm Open để kiểm tra xem file có tồn tại hay không, cũng như là để copy.
- Nếu mở thành công, dùng hàm Read để ghi giá trị vào buffer.
- Tiếp tục yêu cầu người dùng nhập file muốn tạo và copy vào.
- Dùng hàm Create để tạo và nếu như hàm trả về -1 (lỗi không tạo được), thông báo cho người dùng lỗi và hàm có lẽ đã được tạo rồi.
- Xong khi tạo xong, tiếp tục dùng hàm Open để mở, nếu như hàm Open trả về -1 thì chương trình sẽ báo lỗi, lúc này có thể là do file đã được mở rồi.
- Khi mở thành công, dùng hàm Write để ghi giá trị buffer vào file.

3.3.1 Demo minh họa



Hình 3.6: Cây thư mục ban đầu không có file b_test.txt

```
..bss", filepos 0x0, mempos 0x430, size 0x0
(base) lhh1323@lhh1323-Nitro-AN515-57:~/nachos/NachOS-4.0/code/test$ ../build.linux
/nachos -x copy
===== COPY PROGRAM =====

Enter file name to copy: a_test.txt
Enter file name to paste: b_test.txt
Copy file successfully.

===== EXIT PROGRAM =====
Machine halting!
```

Hình 3.7: Chạy chương trình biên dịch để tạo file b_test.txt và chép nội dung file a_test.txt vào

```
code > test > ≡ b_test.txt
1 Chuc thay mot ngay tot lanh.
```

Hình 3.8: Nội dung của file b_test.txt sau khi được copy

3.4 Chương trình delete: Xóa file

3.4.1 Ý tưởng

- Đầu tiên, ta cho người dùng nhập vào tên file mà ta muốn xóa.
- Dùng hàm Remove(filename) để tiến hành xóa file.
- Nếu như giá trị trả về -1 thì in ra "xóa file thất bại"(do file đang được mở).
- Nếu giá trị trả về là 0 thì in ra "xóa file thành công"

3.4.2 Demo minh họa

```
===== DELETE PROGRAM =====
Enter file name: test2.txt
Remove test2.txt successfully.
===== EXIT PROGRAM =====
```

```
≡ test2.txt
```

Hình 3.9: Nội dung biên dịch và kết quả sau khi biên dịch

3.5 Chương trình concatenate: Nối nội dung 2 file

3.5.1 Ý tưởng

- Đầu tiên, ta sẽ yêu cầu người dùng nhập vào tên file được nối thêm(file 1).
- Sau đó ta mở file 1 lên, nếu giá trị trả về là -1 thì xuất ra thông báo "mở file thất bại". Ngược lại, tức đã mở file thành công, thì ta dùng hàm Seek để chuyển con trỏ về cuối file và in ra thông báo "Di chuyển con trỏ về cuối file thành công".
- Tiếp tục yêu cầu người dùng nhập vào tên file sẽ nối vào file trên(file 2).
- Sau đó ta mở file 2 lên, nếu giá trị trả về là -1 thì xuất ra thông báo "mở file thất bại". Ngược lại, dùng Read để đọc nội dung file 2 và dùng hàm Write để viết nội dung vừa đọc được vào file 1. Nếu hàm Write trả về -1 thì việc nối file thất bại, trả về 0 thì việc nối file thành công

3.5.2 Demo minh họa

```
code > test > ≡ test1.txt
1  Le Huu Hung hoc
```

```
code > test > ≡ test2.txt
1  lop 21TNT1
```

Hình 3.10: Nội dung 2 file

```
===== CONCATENATE PROGRAM =====
Enter file name to append: test1.txt
Move cursor to end of file 1 successfully

Enter file name to copy: test2.txt
Concatenate successfully.
===== EXIT PROGRAM =====
```

```
code > test > ≡ test1.txt
1  Le Huu Hung hoc lop 21TNT1
```

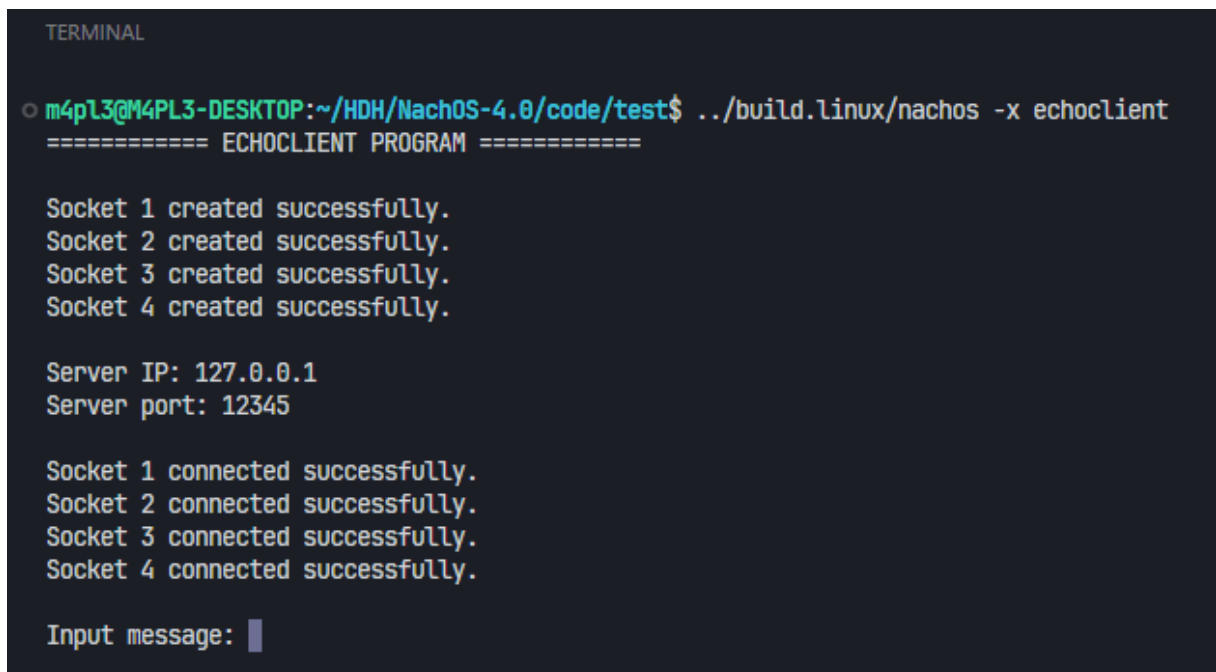
Hình 3.11: Nội dung biên dịch và file sau khi biên dịch

3.6 Chương trình echo

3.6.1 Ý tưởng

1. Khi chạy chương trình, trên màn hình console sẽ xuất hiện ra các thông tin về Socket (4 Socket khởi tạo thành công hay thất bại), địa chỉ IP và Port của Server muốn kết nối tới, và tình trạng kết nối của 4 Socket.
2. Ta nhập nội dung muốn gửi đi vào Console.
3. Chương trình sẽ gửi nội dung đến Server, ngay lập tức Server sẽ trả về nội dung (được in hoa) lại cho chương trình.
4. Kết thúc chương trình.

3.6.2 Demo minh họa



```
TERMINAL

m4p13@M4PL3-DESKTOP:~/HDH/NachOS-4.0/code/test$ ../build.linux/nachos -x echoclient
===== ECHOCLIENT PROGRAM =====

Socket 1 created successfully.
Socket 2 created successfully.
Socket 3 created successfully.
Socket 4 created successfully.

Server IP: 127.0.0.1
Server port: 12345

Socket 1 connected successfully.
Socket 2 connected successfully.
Socket 3 connected successfully.
Socket 4 connected successfully.

Input message: █
```

Hình 3.12: Màn hình bắt đầu khởi chạy chương trình

```
m4pl3@m4pl3-DESKTOP:~/HDH/NachOS-4.0/code/test$ ../build.linux/nachos -x echoclient
===== ECHOCLIENT PROGRAM =====

Socket 1 created successfully.
Socket 2 created successfully.
Socket 3 created successfully.
Socket 4 created successfully.

Server IP: 127.0.0.1
Server port: 12345

Socket 1 connected successfully.
Socket 2 connected successfully.
Socket 3 connected successfully.
Socket 4 connected successfully.

Input message: Day la do an mon he dieu hanh. Demo chuong trinh echoclient.█
```

Hình 3.13: Nhập nội dung muốn gửi


```

● m4p13@m4pl3-DESKTOP:~/HDH/NachOS-4.0/code/test$ ../build.linux/nachos -x echoclient
===== ECHOCLIENT PROGRAM =====

Socket 1 created successfully.
Socket 2 created successfully.
Socket 3 created successfully.
Socket 4 created successfully.

Server IP: 127.0.0.1
Server port: 12345

Socket 1 connected successfully.
Socket 2 connected successfully.
Socket 3 connected successfully.
Socket 4 connected successfully.

Input message: Day la do an mon he dieu hanh. Demo chuong trinh echoclient.

Socket 1 sent message: OK
Socket 2 sent message: OK
Socket 3 sent message: OK
Socket 4 sent message: OK

Socket 1 received: DAY LA DO AN MON HE DIEU HANH. DEMO CHUONG TRINH ECHOCLIENT.
Socket 2 received: DAY LA DO AN MON HE DIEU HANH. DEMO CHUONG TRINH ECHOCLIENT.
Socket 3 received: DAY LA DO AN MON HE DIEU HANH. DEMO CHUONG TRINH ECHOCLIENT.
Socket 4 received: DAY LA DO AN MON HE DIEU HANH. DEMO CHUONG TRINH ECHOCLIENT.

Socket 1 closed successfully.
Socket 2 closed successfully.
Socket 3 closed successfully.
Socket 4 closed successfully.

===== EXIT PROGRAM =====
Machine halting!

Ticks: total 613480360, idle 613445860, system 33850, user 650
Disk I/O: reads 0, writes 0
Console I/O: reads 61, writes 953
Paging: faults 0
Network I/O: packets received 0, sent 0
○ m4p13@m4pl3-DESKTOP:~/HDH/NachOS-4.0/code/test$

```

Hình 3.14: Kết quả trả về từ chương trình

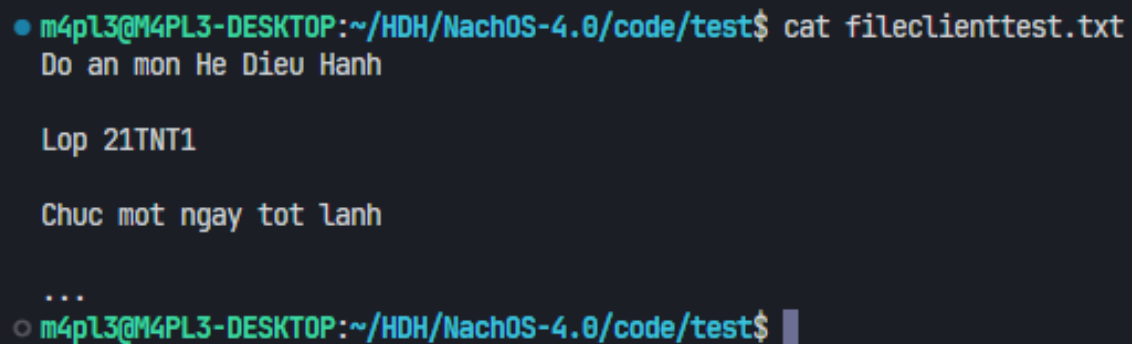
3.7 Chương trình truyền file

3.7.1 Ý tưởng

1. Khi chạy chương trình, trên màn hình console sẽ xuất hiện ra các thông tin về Socket (khởi tạo thành công hay thất bại), địa chỉ IP và Port của Server muốn kết nối tới, và tình trạng kết nối.

2. Ta nhập tên file muốn gửi nội dung.
3. Chương trình sẽ gửi nội dung đến Server, ngay lập tức Server sẽ trả về nội dung (được in hoa) lại cho chương trình. Nếu thành công, cho người dùng nhập file cần xuất nội dung ra.
4. Nếu file chưa tồn tại, chương trình sẽ tự động tạo ra file mới và ghi nội dung vào file đó. Ngược lại chương trình sẽ thông báo file đã tồn tại.
5. Kết thúc chương trình.

3.7.2 Demo minh họa



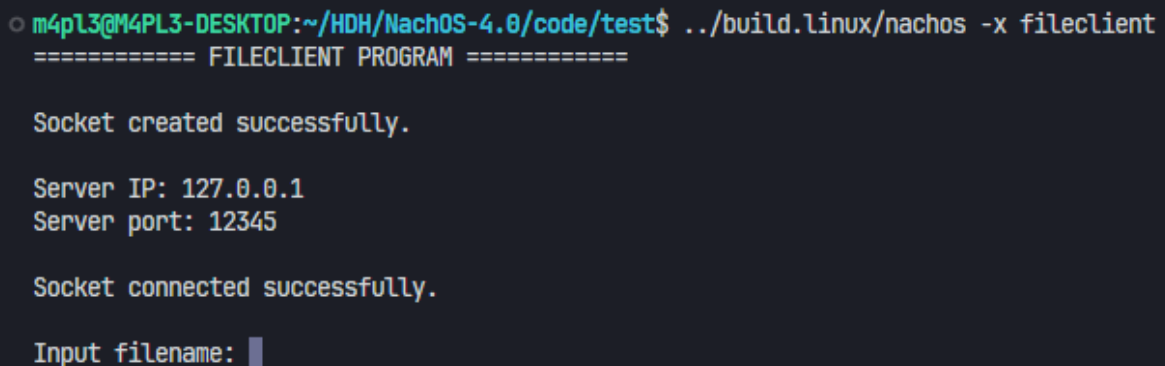
```
m4p13@M4PL3-DESKTOP:~/HDH/NachOS-4.0/code/test$ cat fileclienttest.txt
Do an mon He Dieu Hanh

Lop 21TNT1

Chuc mot ngay tot lanh

...
m4p13@M4PL3-DESKTOP:~/HDH/NachOS-4.0/code/test$
```

Hình 3.15: Nội dung của file cần gửi đi (fileclienttest.txt)



```
m4p13@M4PL3-DESKTOP:~/HDH/NachOS-4.0/code/test$ ../build.linux/nachos -x fileclient
===== FILECLIENT PROGRAM =====

Socket created successfully.

Server IP: 127.0.0.1
Server port: 12345

Socket connected successfully.

Input filename: 
```

Hình 3.16: Màn hình bắt đầu khởi chạy chương trình

```
m4p13@m4pl3-DESKTOP:~/HDH/NachOS-4.0/code/test$ ../build.linux/nachos -x fileclient
===== FILECLIENT PROGRAM =====

Socket created successfully.

Server IP: 127.0.0.1
Server port: 12345

Socket connected successfully.

Input filename: fileclienttest.txt
Socket sent message: OK

Input filename to save content: 
```

Hình 3.17: Thực hiện nhập tên file muốn gửi đi, sau đó nhấn Enter

```
m4p13@m4pl3-DESKTOP:~/HDH/NachOS-4.0/code/test$ ../build.linux/nachos -x fileclient
===== FILECLIENT PROGRAM =====

Socket created successfully.

Server IP: 127.0.0.1
Server port: 12345

Socket connected successfully.

Input filename: fileclienttest.txt
Socket sent message: OK

Input filename to save content: outfile.txt
Write to file outfile.txt successfully.
Socket closed successfully.

===== EXIT PROGRAM =====
Machine halting!

Ticks: total 1041516332, idle 1041503742, system 12260, user 330
Disk I/O: reads 0, writes 0
Console I/O: reads 31, writes 336
Paging: faults 0
Network I/O: packets received 0, sent 0
m4p13@m4pl3-DESKTOP:~/HDH/NachOS-4.0/code/test$ 
```

Hình 3.18: Nhận từ Server thành công. Thực hiện nhập tên file cần xuất. Kết thúc chương trình.

```
● m4p13@m4pl3-DESKTOP:~/HDH/NachOS-4.0/code/test$ cat outfile.txt
DO AN MON HE DIEU HANH

LOP 21TNT1

CHUC MOT NGAY TOT LANH

...
○ m4p13@m4pl3-DESKTOP:~/HDH/NachOS-4.0/code/test$
```

Hình 3.19: Nội dung của file sau khi được xuất bởi chương trình.

Chương 4

Kết quả đạt được

Các yêu cầu của đề án và kết quả đạt được của nhóm được tóm tắt trong bảng dưới đây:

Bảng 4.1: Bảng kết quả đạt được của đề án

Phần	Câu	Ghi chú	Kết quả
1	1	syscall Create	Đã hoàn thành
	2	syscall OpenFileID	Đã hoàn thành
	3	syscall Read	Đã hoàn thành
	4	syscall Seek	Đã hoàn thành
	5	syscall Remove	Đã hoàn thành
2	1	syscall Remove	Đã hoàn thành
	2	syscall Connect	Đã hoàn thành
	3	syscall Send	Đã hoàn thành
		syscall Receive	Đã hoàn thành
3		Nâng cao	Đã hoàn thành
4		Chương trình Test	Đã hoàn thành
	1,2,3,4,5	create, copy, cat, delete, concatenate	Đã hoàn thành
	6	echo	Đã hoàn thành
	7	Truyền file	Đã hoàn thành
		Báo cáo	Đã hoàn thành

Tài liệu tham khảo

- [1] Github Repo: nachos - TrinhLongVu, <https://github.com/TrinhLongVu/nachos>.
- [2] Github Repo: nachos-project - leduythuucs, <https://github.com/leduythuucs/nachos-project>.
- [3] Nachos Project Guide - Jeff Chase, https://users.cs.duke.edu/~chase/nachos-guide/guide/nachos.htm#_Toc535602528.
- [4] Echo server and client using sockets in c, <https://mohsensy.github.io/programming/2019/09/25/echo-server-and-client-using-sockets-in-c.html>.