ĐẠI HỌC QUỐC GIA THÀNH PHỐ HỒ CHÍ MINH

TRƯỜNG ĐẠI HỌC KHOA HỌC TỰ NHIÊN

KHOA CÔNG NGHỆ THÔNG TIN

BÁO CÁO

ĐỒ ÁN 02

Thành phố Hồ Chí Minh - 2022

MỤC LỤC

THÔNG TIN THÀNH VIÊN NHÓM

MSSV	Họ Tên	Email	Công việc
20120184	Phạm Quang Tân	20120184@student.hcmus.edu.vn	**1,2,6,7,8,9**
20120201	Phạm Gia Thông	20120201@student.hcmus.edu.vn	**1,2,3,4,5,7,8,9**

Danh sách công việc

1. **Tìm hiểu thông tin và cài đặt Nachos**
2. **Cài đặt ssh và live share Vscode, hỗ trợ làm việc**
3. **Tìm hiểu thiết kế của hệ điều hành**
4. **Cài đặt Exceptions và System Calls**
5. **Cách khai báo trong makefile với start.s**
6. **Cài đặt ngôn ngữ C**
7. **Kiểm tra và hoàn thiện lại code**
8. **Viết báo cáo đồ án**
9. **Cài đặt hàm thao tác file**

1. Tổng quan về Nachos

NachOS là viết tắt của Not Another Completely Heuristic Operating System, một phần mềm giả lập hệ điều hành với kiến trúc MIPS (Million Instructions Per Second) chạy trên nền tảng Linux.

Nachos là phần mềm chứa mã nguồn mở (open-source), chạy trên máy ảo được giả lập theo kiến trúc MIPS kèm với cross-compiler (đã biên dịch sẵn) để biên dịch các chương trình C thành các file thực thi có thể chạy được trên hệ điều hành này.

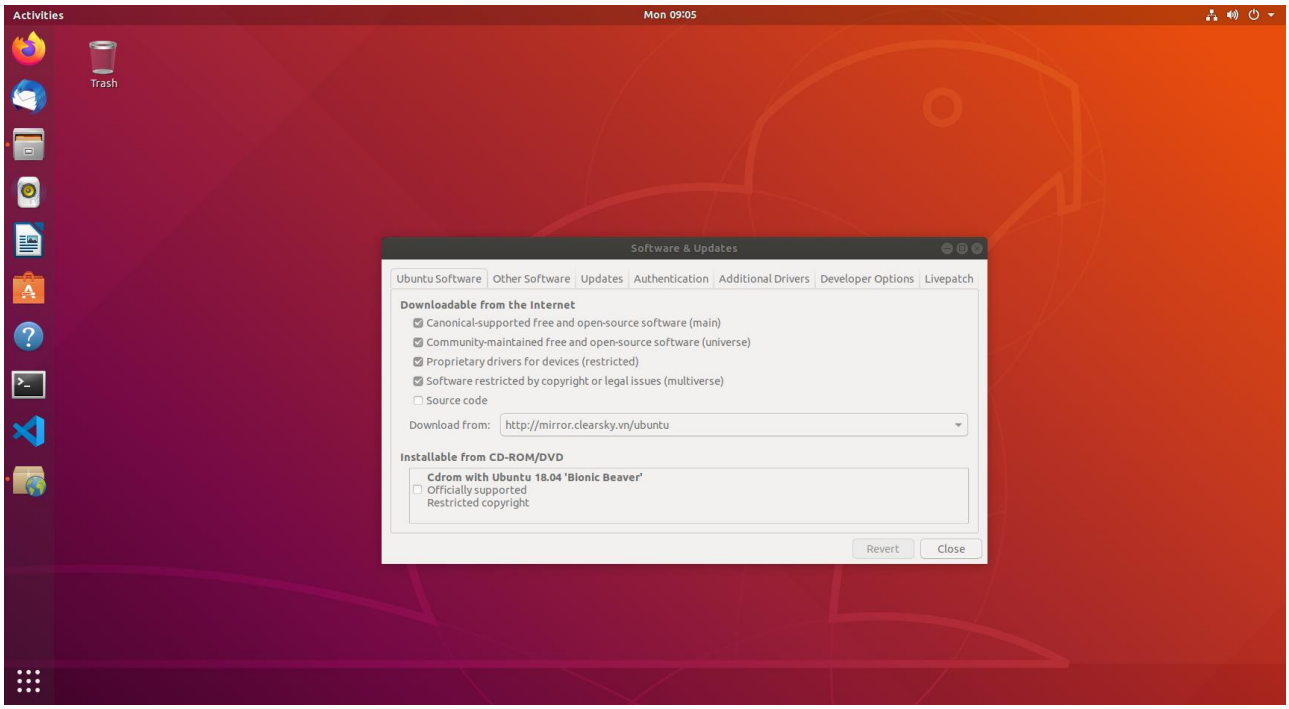
2. Hiểu mã chương trình NachOS

a) Cài đặt NachOS trên nền tảng linux

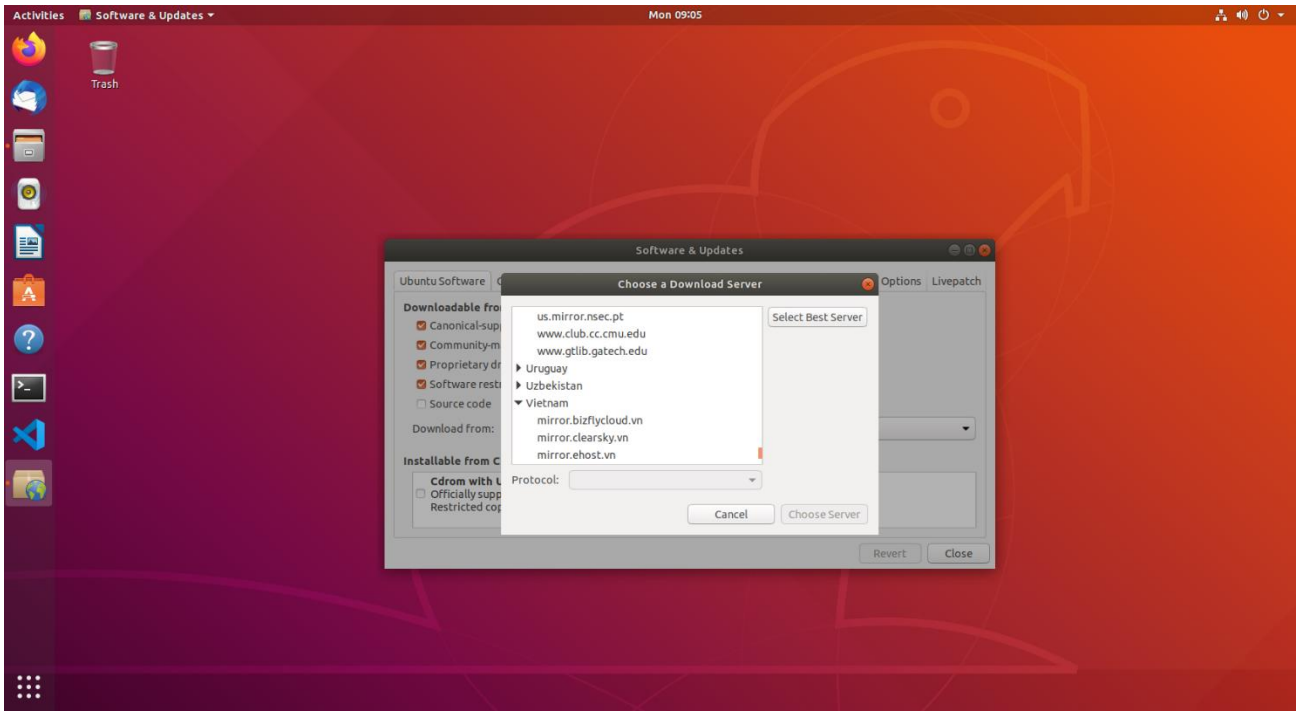
Sau khi cài Ubuntu 18.04, cài các công cụ cần thiết (g++, gcc, make, csh, geany,...). Ta tiến hành cài đặt phần mềm NachOS.

Khởi động terminal trên Ubuntu, nhập các dòng lệnh theo từng bước theo trang https://www.fit.hcmus.edu.vn/~ntquan/os/setup_nachos.html để có thể cài đặt. Có thể chuyển server của Ubuntu 18.04 từ US sang server ở vietnam để có thể cài đặt nhanh hơn.

Vào mục Software & Update để thay đổi service



Download from -> Other



Chọn Vietnam -> dòng 2 (chọn để tránh tình trạng có thể thiếu gói khi cài đặt và được tốc độ tốt nhất)

b) Các file được sử dụng trong đồ án

Folder	File	Ý nghĩa
userprog	syscall.h	Định nghĩa system call mới thông qua macro và giao diện hàm.
	exception.cc	Cài đặt các system call.
	ksyscall.h	Nơi đặt các công cụ bổ trợ cho các hàm syscall sử dụng trong exception.cc và có các hàm thêm hỗ trợ các công cụ nhằm tránh làm sập hệ điều hành
test	start.S	Khai báo thông điệp tương ứng với các system call
	Makefile	Chương trình giúp chuyển các file chạy trên nền MIPS
	createfile.c	Mã chương trình khởi tạo một tập tin
	delete.c	Mã chương trình thực thi việc xóa một tập tin
	cat.c	In ra màn hình nội dung của tập tin
	num_io.c	Mã chương trình đọc vào một số integer và xuất ra số integer đó
	string.c	Mã chương trình đọc vào một chuỗi kí tự và xuất ra chuỗi kí tự đó
	copy.c	Mã chương trình thực thi việc copy nội dung của tập tin 1 sang nội dung của tập tin 2
	concatenate.c	Mã chương trình thực thi việc nối nội dung của 2 tập tin với nhau
	halt.c	Mã chương trình shut down hệ điều hành
filesys	filesys.h	Khai báo bảng file, chứa các lệnh thao tác với file
	filesys.cc	Chứa các lệnh thực thi với file
	tablearr.h	Hỗ trợ các lệnh thao tác với file, lưu trữ các file đang được xử lý
	openfile.h	Cùng với filesys.h hỗ trợ khai báo các hàm liên quan, các lệnh thao tác với file

3. Hiểu thiết kế của hệ điều hành

a) UserMode

Bao gồm các thành phần của chương trình người dùng (tập hợp mã máy), và vùng nhớ của các chương trình ứng dụng chạy trên hệ thống NachOS/MIPS

b) SystemMode

Cung cấp các thành phần nhằm quản lý vùng nhớ, bộ nhớ của hệ điều hành NachOS, quản lí tiến trình và các SystemCall

c) *Giao tiếp với nền tảng MIPS*

Bao gồm các thanh ghi, bộ nhớ chính và các cơ chế đọc ghi đơn giản, xử lí từng lệnh dựa trên tập lệnh MIPS

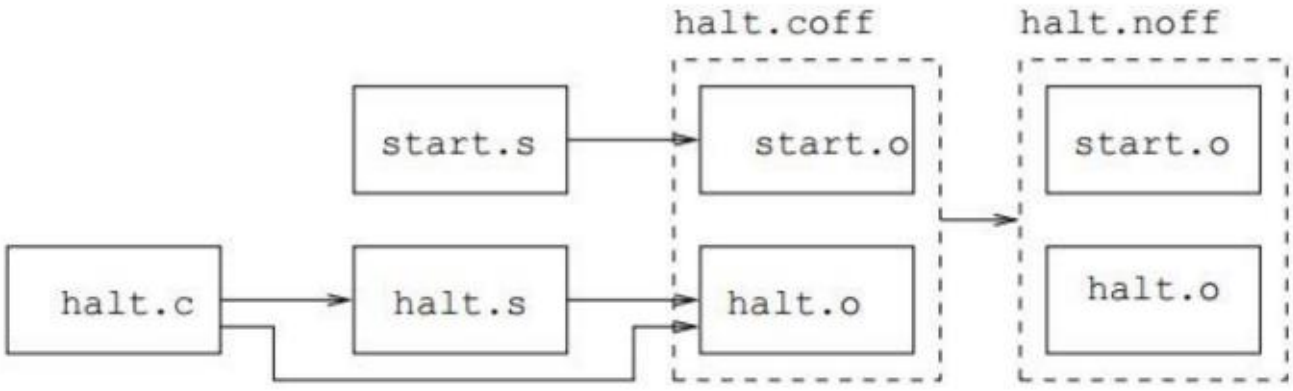
4. Exceptions và System Calls

a) *Quy trình thực thi của một chương trình trên NachOS*

Để một chương trình (VD: halt.c) có thể thực thi nó cần phải được biên dịch. Quá trình biên dịch trên NachOS gồm có ba bước:

- Chương trình halt.c được cross-compiler biên dịch thành tập tin halt.s là mã hợp ngữ chạy trên kiến trúc MIPS
- Tập tin halt.s được liên kết với starts.s (được coi là thư viện) để tạo thành tập tin halt.coff (bao gồm halt.o và start.o) dạng file thực thi trên Linux với kiến trúc MIPS
- Tập tin halt.coff được phần mềm coff2noff chuyển thành tập tin halt.noff đây chính là dạng file thực thi trên NachOS kiến trúc MIPS

Quá trình này được mô tả bằng hình dưới đây



Để thực thi chương trình sử dụng câu lệnh:

~/nachos/NachOS-4.0/code/test> ../build.linux/nachos -x halt

b) *Cách tạo một System Call*

Bước 1: Tạo macro:

– Tránh nhầm lẫn việc gọi hàm, cần define con số cụ thể trong kernel space thành một macro.

➔ Vào syscall.h: – VD: System Call "ReadInt":

• #define SC_ReadInt 44

– Khai báo interface cho user: tạo một hàm để user giao tiếp với hệ điều hành

➔Vào syscall.h:

• Khai báo hàm: int ReadNum();

Bước 2: Định nghĩa hàm trong file assembler

• Ta mở files:

– code/test/start.s

• Và chèn đoạn mã vào, sẽ có những đoạn mẫu sẵn trong file start.s, ta chỉ cần thay đổi với tên hàm ta cần gọi System Call đó:

– VD: System Call "ReadInt"

- .globl ReadNum
- .ent ReadNum

• ReadNum:

- addiu $2, $0, SC_ReadInt
- syscall
- j $31
- .end ReadNum

Bước 3: Định nghĩa cụ thể cho một công việc

Mở file userprog/exception.cc

Chuyển đoạn mã "If.....else" sang đoạn mã "switch....case" với các case là các syscall cần gọi.

Tăng thanh ghi PC → PC + 4 để tránh hiện tượng loop, được gọi bởi hàm PC_counting và lấy đoạn mã mẫu trong case SC_Add.

c) *Ý nghĩa các thanh ghi*

Mã của system call được đưa vào thanh ghi r2.

Các biến người dùng sử dụng được đưa vào thanh ghi r4, r5, r6.

Giá trị trả về của system call được đưa vào thanh ghi r2.

d) *System2User và User2System*

System2User

Chức năng: Hàm thực hiện chuyển một chuỗi được lưu trong hệ điều hành NachOS vào bộ nhớ của chương trình ứng dụng chạy trên NachOS/MIPS.

Cài đặt hàm tại file exception.cc

– Hàm sẽ nhận 3 tham số đầu vào: giá trị vùng nhớ của biến trong chương trình, độ dài chuỗi, chuỗi được lưu trong biến của hệ điều hành NachOS.

– Thực hiện chuyển từng kí tự từ bộ nhớ hệ điều hành NachOS vào vùng nhớ chương trình.

– Hàm trả về là độ dài chuỗi đã chuyển vào vùng nhớ chương trình.

User2System

Chức năng: Hàm thực hiện chuyển một chuỗi được lưu trong vùng nhớ của chương trình chạy trên NachOS/MIPS vào vùng nhớ của hệ điều hành NachOS.

Cài đặt hàm tại file exception.cc

– Hàm sẽ nhận 2 tham số đầu vào: giá trị vùng nhớ của biến lưu chuỗi trong chương trình, giá trị giới hạn của chuỗi

– Thực hiện chuyển từng kí tự từ chuỗi thuộc vùng nhớ của chương trình vào biến thuộc vùng nhớ hệ điều hành NachOS.

– Hàm trả về là độ dài chuỗi đã chuyển vào vùng nhớ hệ điều hành NachOS.

e) *Lớp SynchConsole*

Chức năng: Hỗ trợ việc nhập xuất từ màn hình console

Gồm các hàm chính:

char GetChar(): Cho phép nhập một kí tự từ màn hình console và lưu biến thuộc vùng nhớ hệ điều hành NachOS.

int GetString(char *buffer, int size): Cho phép nhập một chuỗi kí tự từ màn hình console và lưu biến thuộc vùng nhớ hệ điều hành NachOS.

void PutChar(char ch): Cho phép xuất một kí tự từ biến thuộc vùng nhớ hệ điều hành NachOS ra màn hình console

int PutString(char *buffer, int size): Cho phép xuất một chuỗi kí tự từ biến thuộc vùng nhớ hệ điều hành NachOS ra màn hình console

f) Cài đặt các syscall và các hàm theo yêu cầu

Các System Call cài đặt trong bài:

- SC_Halt: syscall thực hiện việc tắt chương trình
- SC_Add: syscall thực hiện việc cộng 2 số nguyên và trả kết quả ra màn hình console
- SC_Create: syscall thực hiện việc tạo ra một tập tin trong file code/test
- SC_ReadInt: syscall thực hiện việc nhập 1 số nguyên từ màn hình console
- SC_PrintInt: syscall thực hiện việc xuất 1 số nguyên ra màn hình console
- SC_ReadChar: syscall thực hiện việc nhập 1 kí tự từ màn hình console
- SC_PrintChar: syscall thực hiện việc xuất 1 kí tự ra màn hình console
- SC_ReadString: syscall thực hiện việc nhập 1 chuỗi từ màn hình console
- SC_PrintString: syscall thực hiện việc xuất 1 chuỗi ra màn hình console
- SC_Remove: syscall thực hiện việc xóa một tập tin trong file code/test
- SC_Read: syscall thực hiện việc đọc nội dung từ tập tin
- SC_Write: syscall thực hiện việc ghi nội dung vào tập tin
- SC_Open: syscall thực hiện việc mở một tập tin
- SC_Close: syscall thực hiện việc đóng một tập tin
- SC_Seek: syscall thực hiện việc trỏ đến một vùng nhớ trong tập tin

Với mỗi system call được gọi trong userprog/exception.cc thì phải gọi hàm PC_counting() để tránh hiện tượng loop.

Các hàm được cài đặt trong bài, bên trong file userprog/exception.cc:

char* string_User2System(int addr, int convert_length): Hàm thực hiện chuyển một chuỗi được lưu trong vùng nhớ của chương trình chạy trên NachOS/MIPS vào vùng nhớ của hệ điều hành NachOS.

void string_System2User(char* str, int addr, int convert_length): Hàm thực hiện chuyển một chuỗi được lưu trong hệ điều hành NachOS vào bộ nhớ của chương trình ứng dụng chạy trên NachOS/MIPS.

void PC_counting(): Hàm thực hiện tăng thanh ghi PC để tránh hiện tượng loop

void system_ReadInt(): Hàm thực hiện nhận một số nguyên do người dùng nhập vào từ màn hình console và lưu trữ số nguyên đó ở thanh ghi r2 của chương trình hệ thống, sử dụng SysReadNum() trong file userprog/ksyscall.h.

void system_PrintInt(): Hàm thực hiện xuất một số nguyên ra màn hình console, số nguyên đó được lấy từ thanh ghi r4 của chương trình hệ thống, sử dụng SysPrintNum() trong file userprog/ksyscall.h.

void system_ReadString(): Hàm thực hiện nhận một chuỗi kí tự do người dùng nhập vào từ màn hình console và lưu trữ chuỗi kí tự đó ở thanh ghi r2 của chương trình hệ thống, sử dụng SysReadString() trong file userprog/ksyscall.h và string_System2User (int addr, int convert_length) để chuyển vùng nhớ của chuỗi đó vào bộ nhớ của chương trình system.

void system_PrintString(): Hàm thực hiện xuất một số nguyên ra màn hình console, số nguyên đó được lấy từ thanh ghi r4 của chương trình hệ thống, sử dụng SysPrintNum() trong file userprog/ksyscall.h và string_User2System (char* str, int addr, int convert_length) để chuyển lại vùng nhớ từ chương trình về hệ thống để có thể in ra màn hình.

void system_CreateFile(): Hàm thực hiện nhận một chuỗi kí tự chính là tên tập tin cần khởi tạo do người dùng nhập vào từ màn hình console và chuỗi kí tự đó được lấy từ thanh ghi r4 của chương trình hệ thống, sử dụng SysCreateFile(char* filename) trong file userprog/ksyscall.h.

void system_OpenFile(): Hàm thực hiện nhận một chuỗi kí tự chính là tên tập tin cần được mở do người dùng nhập vào từ màn hình console và chuỗi kí tự đó được lấy từ thanh ghi r4 của chương trình hệ thống, sử dụng SysOpen(char* filename) trong file userprog/ksyscall.h.

void system_CloseFile(): Hàm thực hiện nhận một mã số đã được định dạng chính là mã số của tập tin cần được đóng do người dùng đã mở trước đó và mã số đó được lấy từ thanh ghi r4 của chương trình hệ thống.

void system_ReadFile(): Hàm thực hiện nhận một chuỗi kí tự để chứa nội dung được đọc trong tập tin, một số nguyên là kích thước của nội dung đó trong tập tin và một mã số của tập tin đó cần được mở để đọc vào do người dùng nhập đã mở trước đó và chuỗi kí tự, số nguyên và mã số đó lần lượt được lấy từ thanh ghi r4, r5, r6 của chương trình hệ thống.

void system_WriteFile(): Hàm thực hiện nhận một chuỗi kí tự để chứa nội dung được đọc trong tập tin, một số nguyên là kích thước của nội dung đó trong tập tin và một mã số của tập tin đó cần được mở để ghi vào do người dùng nhập đã mở trước đó và chuỗi kí tự, số nguyên và mã số đó lần lượt được lấy từ thanh ghi r4, r5, r6 của chương trình hệ thống.

void system_SeekLocation(): Hàm thực hiện nhận một số nguyên là vị trí đặt con trỏ và một mã số của tập tin do người dùng nhập đã mở trước đó và số nguyên đó cùng với mã số tập tin ở thanh ghi r4, r5 của chương trình hệ thống.

void system_RemoveFile(): Hàm thực hiện nhận một chuỗi kí tự chính là tên tập tin cần được xóa do người dùng nhập vào từ màn hình console và chuỗi kí tự đó được lấy từ thanh ghi r4 của chương trình hệ thống, sử dụng SysRemove(char* filename) trong file userprog/ksyscall.h.

5. *Một số chương trình*

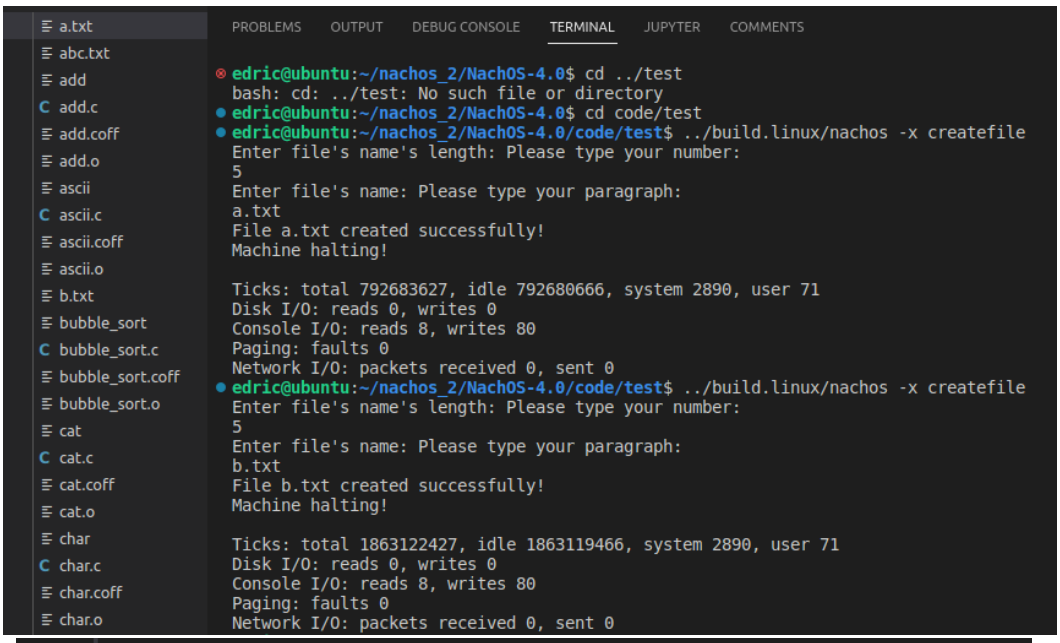
- Chương trình ***add*** để cộng hai số nguyên

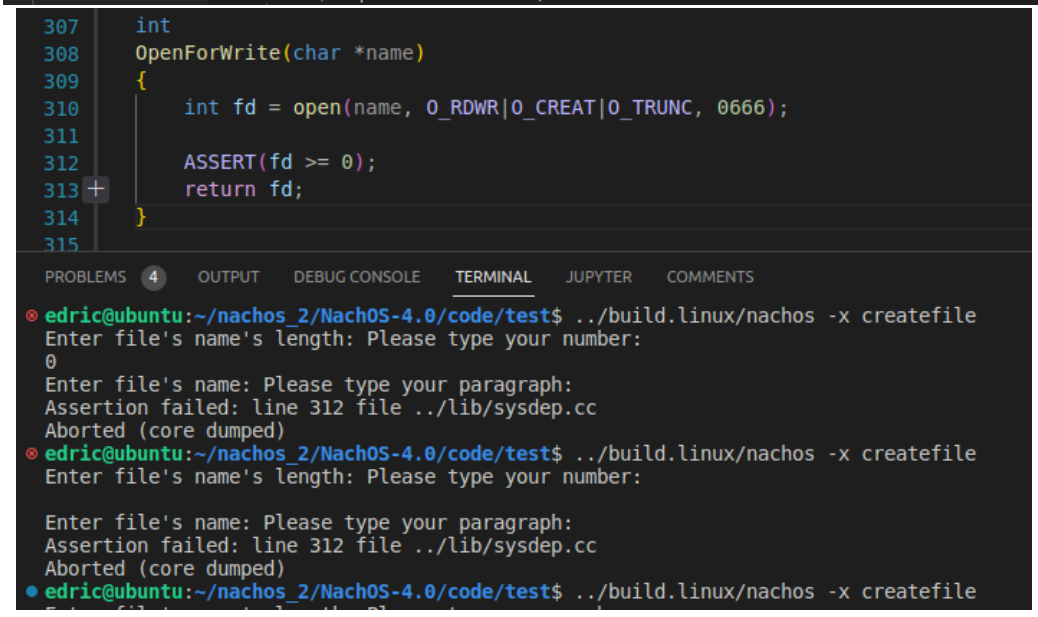
  Sử dụng syscall Add để cộng hai tham số đầu vào, hai tham số được truyền lần lượt vào thanh ghi số 4 và số 5, chương trình thực hiện phép cộng sau đó chuyển kết quả nhận được vào thanh ghi số 2 và xuất ra màn hình để hiển thị kết quả.
- Chương trình ***halt*** để tắt các chương trình

  Chương trình gọi hàm Halt() từ kernel để tắt chương trình và biểu diễn kết quả mà chương trình trả về.
- Chương trình ***createfile*** để đọc và khởi tạo một tập tin mới lưu trữ ở code/test

  Ở phần Create(char* name), sử dụng syscall Create và syscall ReadString để đọc một chuỗi kí tự do người dùng nhập tên tập tin vào cho chương trình hệ thống, sau

đó theo các thủ tục của ReadString(char buffer[], int length), chuỗi kí tự đó truyền xuống cho chương trình hệ thống lưu trữ. Ở đây cần được sử dụng hàm Create(char* name) được khai báo trong filesys, tập tin được khởi tạo với chế độ OpenForWrite(char* name) để có thể ghi nội dung vào tập tin đó. Có các ngoại lệ xử lý như tên tập tin rỗng, kích thước chuỗi tên quá dài hoặc không có kích thước thì trả về kết quả thất bại cho chương trình createfile. Nếu như khởi tạo tập tin thành công, trả về 0 và lưu trữ vào thanh ghi kết quả r2 của chương trình hệ thống, ngược lại trả về -1.

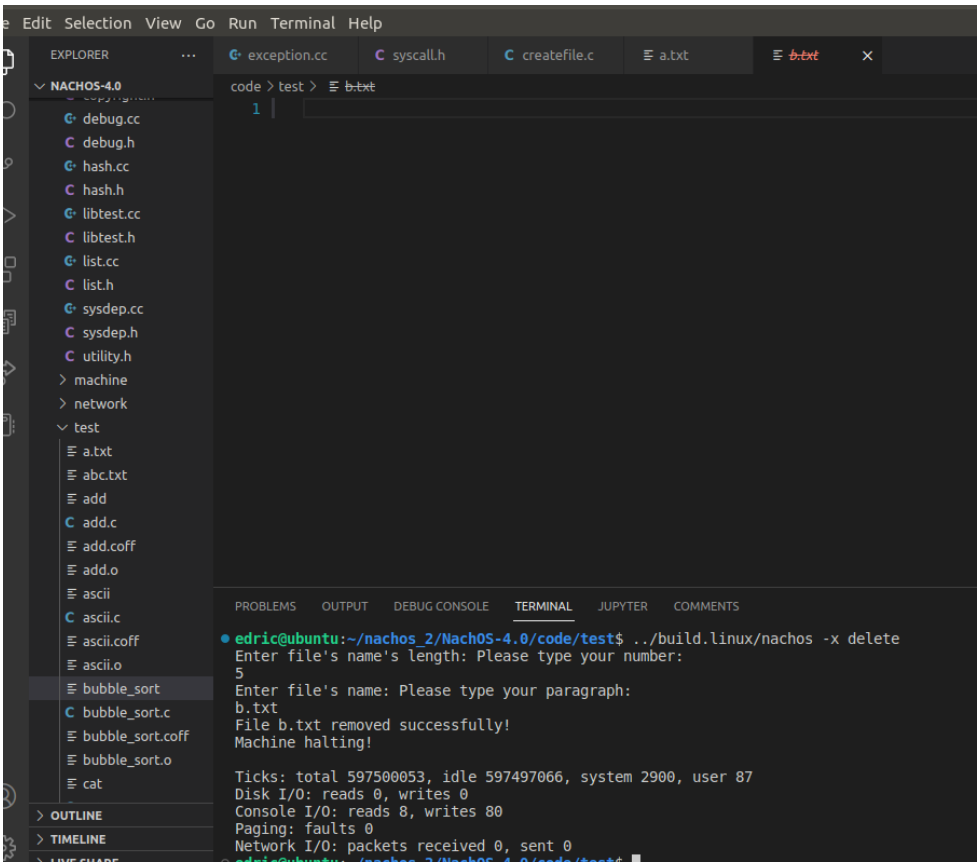




Đã được gài sẵn kích thước chuỗi nhập phải khác 0

- Chương trình ***delete*** để đọc và xóa một tập tin lưu trữ ở code/test

  Ở phần Remove(char* name), sử dụng syscall Remove và syscall ReadString để đọc một chuỗi kí tự do người dùng nhập tên tập tin vào cho chương trình hệ thống, sau đó theo các thủ tục của ReadString(char buffer[], int length), chuỗi kí tự đó truyền xuống cho chương trình hệ thống lưu trữ. Ở đây cần được sử dụng hàm Remove(char* name) được khai báo trong filesys, tập tin được xóa khỏi chương trình. Có các ngoại lệ xử lý như tên tập tin rỗng, kích thước chuỗi tên quá dài hoặc không có kích thước cũng như là tập tin đang mở thì trả về kết quả thất bại cho chương trình delete. Nếu như xóa tập tin thành công, trả về 0 và lưu trữ vào thanh ghi kết quả r2 của chương trình hệ thống, ngược lại trả về -1.

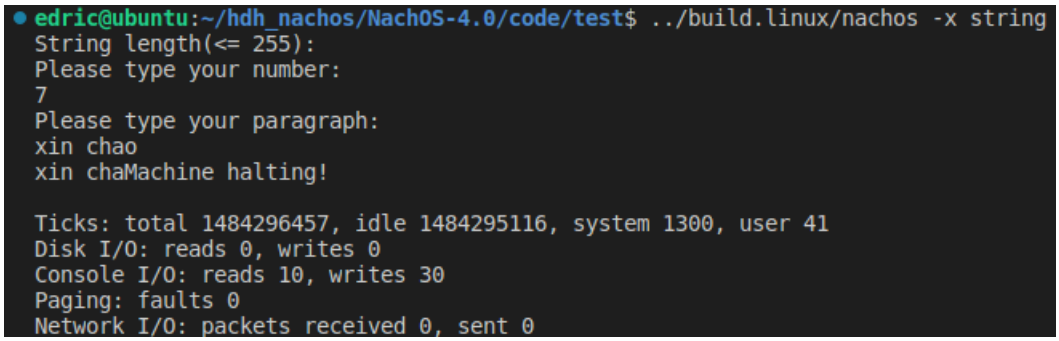


- Chương trình ***string*** để đọc và xuất một chuỗi kí tự ra màn hình

  Ở phần ReadString(char buffer[], int length), sử dụng syscall ReadString để đọc một chuỗi ký tự do người dùng nhập vào cho chương trình hệ thống thông qua việc dùng hàm GetChar() trong synchConsoleIn, sau đó dùng string_System2User(char* str, int addr, int convert_length) để có thể ghi được chuỗi ký tự đó vào bộ nhớ của chương trình. Đầu tiên ta tạo một mảng buffer với kích thước là size do người dùng

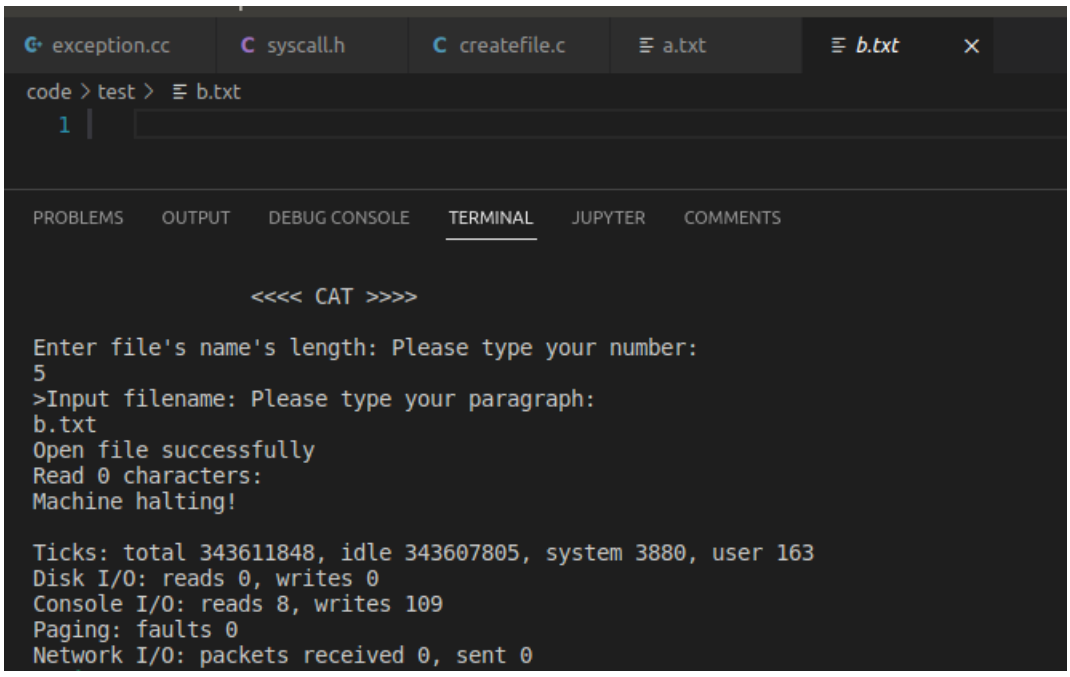
nhập vào (sử dụng hàm ReadNum() để đọc số nguyên do người dùng nhập vào) cộng thêm một (kí tự kết thúc chuỗi), với tham số đầu vào là một mảng ký tự thì ta ghi vào thanh ghi số 4, còn kích thước mảng người dùng nhập thì ghi vào thanh ghi số 5. Gọi vòng lặp và dùng GetChar() để đọc từng kí tự trong chuỗi và lưu vào mảng buffer đã tạo và khi kết thúc vòng lặp thêm kí tự '\0' để kết thúc chuỗi. Sau đó ta có được một mảng chứa chuỗi ký tự mà người dùng nhập vào, gọi string_System2User để ghi xuống bộ nhớ của chương trình, ghi xong ta có thể xóa mảng mà ta đã tạo để chứa chuỗi ký tự của người dùng nhập vào.

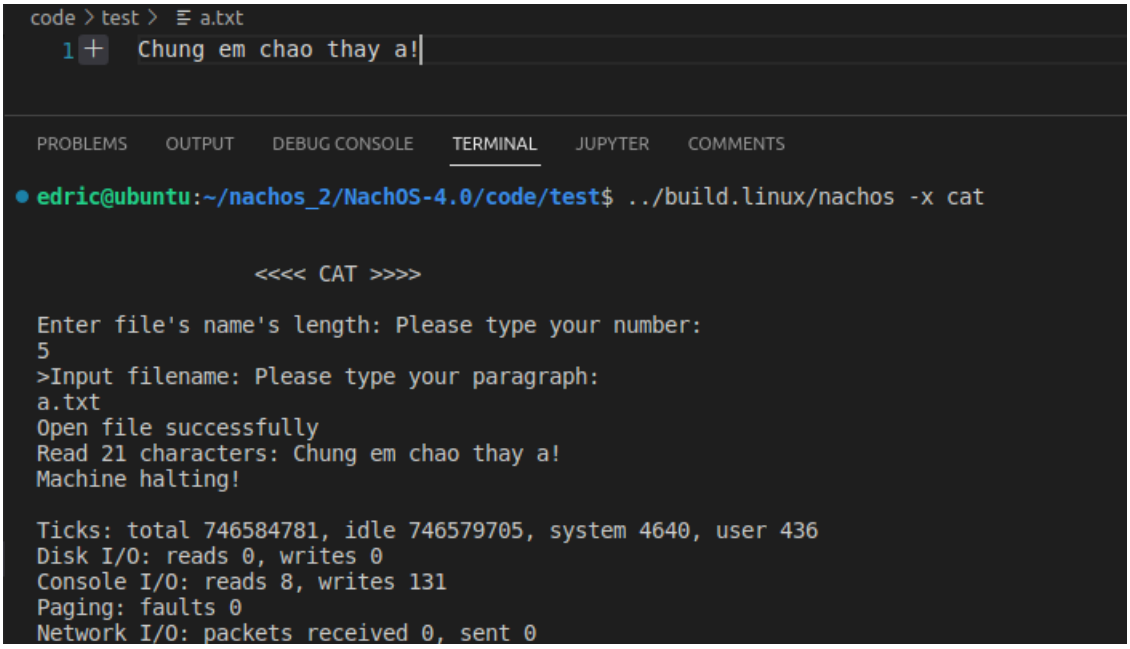
Ở phần PrintString(char buffer[]), sử dụng syscall PrintString để xuất một chuỗi ký tự do người dùng nhập vào cho chương trình hệ thống thông qua việc dùng hàm PutChar() trong synchConsoleOut, sau đó dùng string_User2System(int addr, int convert_length) để có thể đọc (lấy) được chuỗi ký tự đó từ bộ nhớ của chương trình, với tham số đầu tiên truyền vào là địa chỉ của mảng chứa chuỗi ký tự đó và được truyền vào thanh ghi số 4. Gọi vòng lặp và dùng PutChar() để ghi từng kí tự trong chuỗi ra màn hình console, khi hoàn tất ta có thể xóa đi mảng lưu chuỗi ký tự đọc được từ bộ nhớ chương trình hệ thống.



- Chương trình ***cat*** để đọc nội dung trong tập tin và xuất ra màn hình nội dung đó

Sử dụng chương trình Open(char* name), Read(char *buffer, int size, OpenFileId id), PrintString() để có thể mở tập tin để đọc và xuất nội dung ra màn hình. Theo các thủ tục Open, ta có thể dễ dàng mở tập tin đó để đọc nội dung và nội dung đó được chuyển đến vùng lưu trữ buffer của hàm Read và xuất nội dung đó ra màn hình thông qua PrintString. Cần có thủ tục Seek(int position, OpenFileId id) để biết chính xác được vị trí muốn đọc trong tập tin đó (đọc từ đầu hoặc đọc ở vị trí bất kì) và thủ tục Read() ở code/filesys/openfile để có thể đọc được. Nếu mở thành công trả về mã số của tập tin đó, đọc tập tin thành công sẽ trả về kích thước của nội dung đọc được tương ứng với vị trí đọc cũng được trả về và đều lưu qua thanh ghi r2. Khi thực hiện xong thì phải Close(int fileId) tập tin mà chúng ta đã mở.

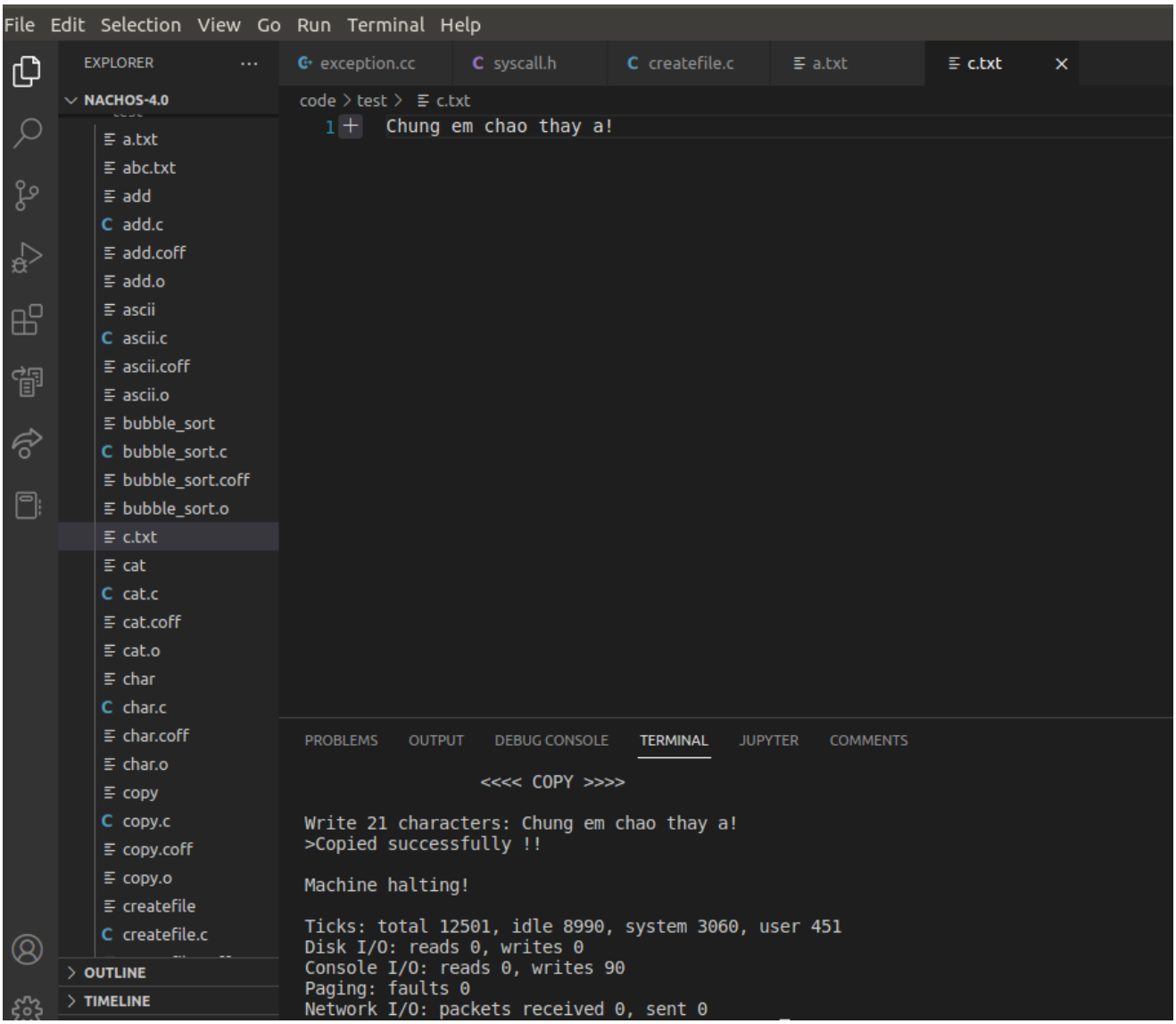




- Chương trình ***copy*** để đọc nội dung trong tập tin và ghi nội dung đó ở một tập tin khác

Tương tự như chương trình ***cat*** ở phần đầu đọc nội dung trong tập tin và buffer chứa nội dung đó. Bây giờ chúng ta sẽ gọi Open(char* name) để mở tập tin đích, tập tin mà ta muốn copy vào nội dung từ tập tin ban đầu. Gọi Seek() để tìm vị trí muốn ghi vào ở tập tin đích sau đó gọi Write(char *buffer, int size, OpenFileId id) tương tự như Read(), sẽ ghi được nội dung từ buffer đó vào tập tin đích và cần thủ tục Write()

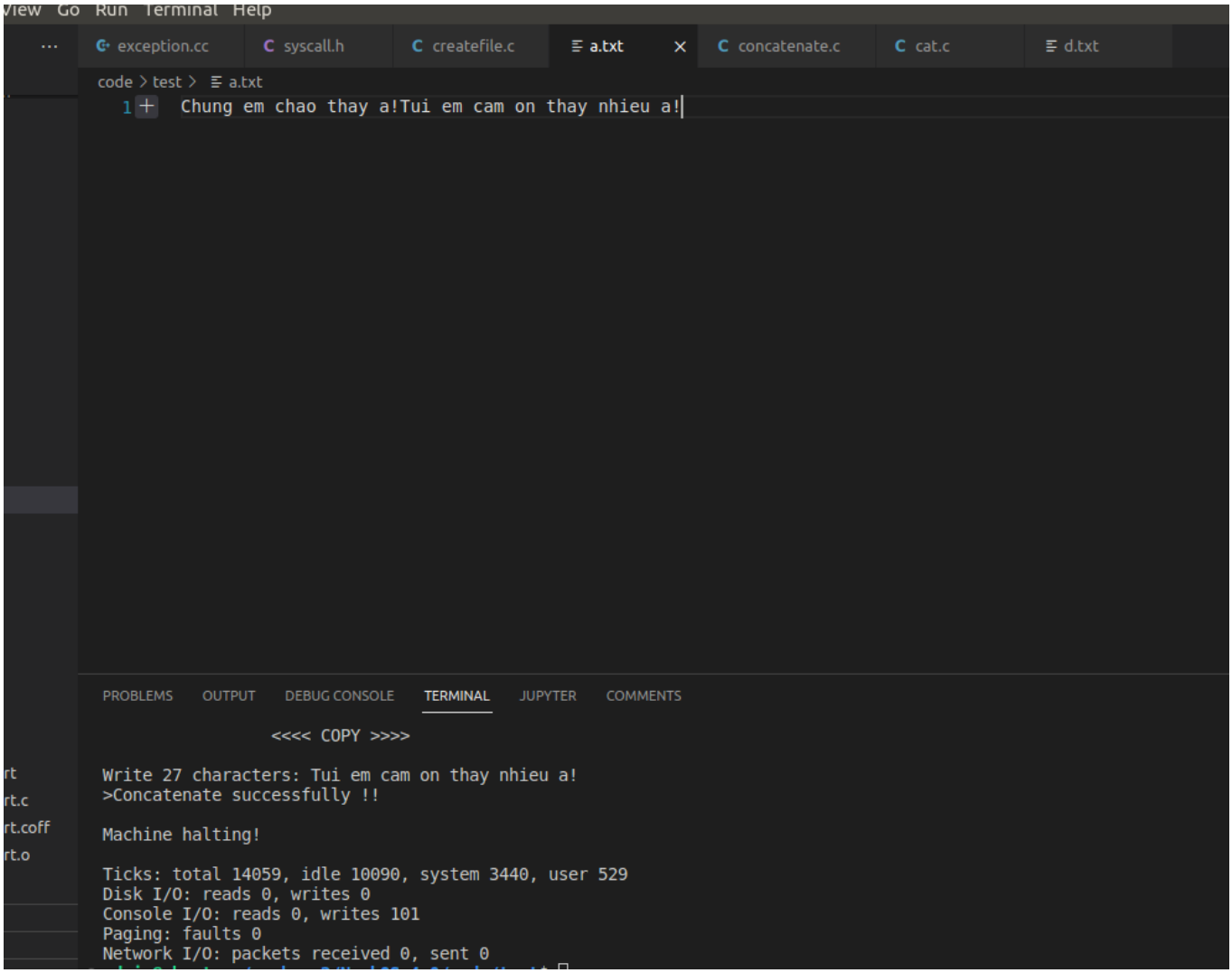
ở code/filesys/openfile để có thể ghi được. Sau đó chúng ta cũng đóng lại tập tin với Close() sau khi mở.



- Chương trình ***concatenate*** để để đọc nội dung trong hai tập tin và nối nội dung ở hai tập tin lại với nhau

  Tương tự như chương trình ***copy*** nhưng chỉ khác ở việc gọi Seek() để tìm vị trí muốn ghi vào ở tập tin mà muốn nối nội dung vào, Seek(-1, id) để biết được kích thước nội dung tập tin đã đọc hoặc lấy kích thước của mảng buffer để tìm vị trí nối nội dung vào. Có thể đọc thành hai mảng buffer ở 2 tập tin sau đó ghi 2 mảng buffer

vào một tập tin mới (Create()) hoặc như chương trình ***copy***, đọc nội dung của tập tin đích rồi ghi tiếp nội dung vào tập tin nguồn ngay sau vị trí đã có nội dung có sẵn. Gọi Write(char *buffer, int size, OpenFileId id) tương tự như Read(), sẽ ghi được nội dung từ buffer đó vào tập tin muốn và cần thủ tục Write() ở code/filesys/openfile để có thể ghi được. Sau đó chúng ta cũng đóng lại các tập tin với Close() sau khi mở để thao tác.



- Chương trình ***num_io*** để đọc và xuất một số nguyên ra màn hình

  Ở phần ReadNum(), sử dụng syscall ReadNum để đọc một số nguyên do người dùng nhập vào cho chương trình hệ thống, sau đó số nguyên đó được ghi vào thanh ghi số 2 để truyền xuống cho chương trình hệ thống lưu trữ. Ở đây có sử dụng các ngoại lệ như là người dùng nhập vào một số nguyên quá lớn, vượt quá phạm vi của

số nguyên trong C, nhập 1 kí tự thay vì một số nguyên, nhập số thực, thì system call ReadInt tiến hành xử lý các trường hợp đó trong chương trình bubble_sort, xử lý ngay từ khâu ban đầu là nhập số nguyên.

Dùng thư viện ksyscall.h để xử lý người dùng nhập vào các kí tự, các khoảng trắng, hoặc không nhập gì, kiểm tra số nguyên nhập vượt quá phạm vi. Đầu tiên tạo một mảng kí tự với kích thước là chiều dài tối đa (10 kí tự cho 1 số nguyên có số chữ số nằm trong vùng giới hạn tối đa) cộng thêm 2 kí tự bao gồm cả dấu của số nguyên và kí tự kết thúc file nếu có. Đọc vào 1 kí tự đầu tiên của số nguyên thông qua hàm GetChar() trong synchConsoleIn, kiểm tra kí tự đặc biệt (khoảng trắng, kết thúc file,…), nếu có thì trả về 0, không thì chuyển kí tự đó vào mảng kí tự đã tạo ban đầu, sau đó đọc tiếp với hàm GetChar(), thực hiện vòng lặp đến khi có kí tự không thỏa xuất hiện, nếu chiều dài của mảng vượt quá phạm vi giới hạn thì cũng dừng vòng lặp và trả về 0.

Sau khi xử lý qua thư viện ksyscall.h, ta tiếp tục xét tính thỏa mãn của mảng kí tự. Nếu kích thước mảng là 0 trả về số 0, so sánh mảng với số nguyên bé nhất trong phạm vi, nếu bằng trả về 0. Tiếp đến sẽ xét lượng dấu và số lượng số 0 ở đầu của số nguyên do người dùng nhập vào. Nếu có trong mảng có kí tự ‘-‘ thì bỏ vị trí đầu, đọc tiếp các vị trí còn lại trong mảng, đếm số lượng số 0 ở đầu để có thể trả về số đó không hợp lệ, nếu kí tự đó không phải là số thì trả về 0. Thỏa mãn hết điều kiện thì chuyển các phần tử trong mảng về lại các chữ số và lưu trong một biến nhớ kiểu số nguyên, biến nhớ nhân thêm 10 và cộng với lần lượt các phần tử trong mảng (đã chuyển về kiểu dữ liệu số nguyên) để tiện cho việc so sánh số nguyên và mảng kí tự lưu trữ số nguyên đó ở phần PrintNum(int number)

Ở phần PrintNum(int number), sử dụng syscall PrintNum để xuất ra một số nguyên do người dùng nhập vào cho chương trình hệ thống, số nguyên đó sau khi được xử lý tính thỏa mãn bởi thư viện ksyscallhelper.h thì đang được lưu trữ ở thanh ghi số 2, nhưng đối với chương trình PrintNum thì số nguyên đó ở đây đang là tham số, nên sẽ truyền vào thanh ghi số 4 để tiếp tục xử lý cho việc in ra màn hình.

Tiếp theo, ở thư viện ksyscall.h cũng có một hàm kiểm tra số nguyên nhập vào (đã xử lý qua hàm ReadNum() và được lưu vào một biến nhớ kiểu số nguyên mới để tiện cho việc kiểm tra, so sánh) và mảng kí tự đã tạo từ số nguyên đó có trùng khớp nhau hay không. Nếu số 0, so sánh mảng kí tự với số 0, đúng trả về true, nếu số đó âm nhưng trong mảng kí tự không có ‘-‘ đằng trước thì trả về false, nếu là số âm thì bỏ vị trí đầu của trong mảng kí (khi PrintNum sẽ in ra) và cập nhật lại số đó là số dương để tiện cho việc kiểm tra từng kí tự trong mảng với từng chữ số

của số nguyên có khớp với nhau hay không. Lấy phần dư của số nguyên khi chia với 10 để lấy từng chữ số, kiểm tra với từng phần tử trong mảng kí tự, nếu sai trả về false, sau đó lấy phần nguyên của số nguyên khi chia với 10 để lấy tiếp các chữ số khác. Nếu kích thước của mảng bằng 0 thì hoàn tất việc so sánh và trả về true.

Sau khi xử lý qua thư viện ksyscall.h, ta sẽ dùng hàm PutChar(character) trong synchConsoleOut để in lần lượt các kí tự trong mảng ra màn hình console. Nếu tham số truyền vào là 0 thì in 0, nếu là số nhỏ nhất trong phạm vi số nguyên thì in kí tự '-' trước sau đó in lần lượt các chữ số trong mảng chứa số nguyên bé nhất ra. Nếu tham số truyền vào là số âm thì in kí tự '-' và chuyển số đó thành số dương để dễ dàng in ra màn hình, chuyển các chữ số lần lượt vào một mảng số nguyên, vì chuyển vào theo phương pháp chia lấy phần dư, chia lấy phần nguyên, nên khi in ra để ra số đúng, ta phải in từ cuối mảng số nguyên trở về đầu và các phần tử trong mảng phải được trả về kiểu dữ liệu số nguyên.

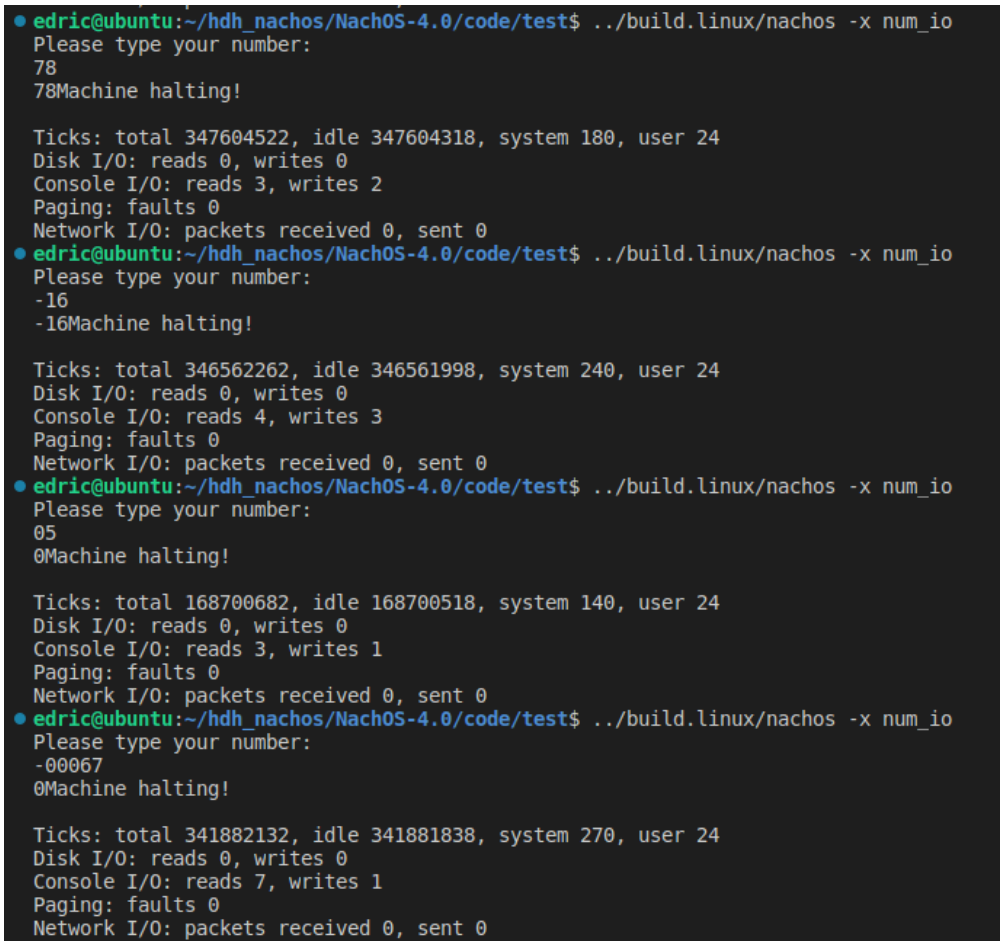
```
edric@ubuntu:~/hdh_nachos/NachOS-4.0/code/test$ ../build.linux/nachos -x num_io
Please type your number:

0Machine halting!

Ticks: total 195303962, idle 195303858, system 80, user 24
Disk I/O: reads 0, writes 0
Console I/O: reads 2, writes 1
Paging: faults 0
Network I/O: packets received 0, sent 0
```

```
edric@ubuntu:~/hdh_nachos/NachOS-4.0/code/test$ ../build.linux/nachos -x num_io
Please type your number:
fdfkertg
0Machine halting!

Ticks: total 248684652, idle 248684288, system 340, user 24
Disk I/O: reads 0, writes 0
Console I/O: reads 9, writes 1
Paging: faults 0
Network I/O: packets received 0, sent 0
```



Note: chưa xử lý được các ngoại lệ và nhập tên tập tin ở copy và concatenate

TÀI LIỆU THAM KHẢO

Danh mục tài liệu tham khảo:

[1]*https://en.m.wikipedia.org/wiki/Not_Another_Completely_Heuristic_Operating_System?fbclid=IwAR3wuzs3Suq4QTLP9UEi8PduKqmOsfWmVsHFaFzBWRw_4jhv83v1GD-xJr0*

[2]*https://homes.cs.washington.edu/~tom/nachos/?fbclid=IwAR2TSW5u_sANlN0nk7fmGfa6wx3CeAvjRsePMY1gN04935AmjXMFwsGkSCo*

[3]*https://cseweb.ucsd.edu/classes/sp18/cse120a/projects/nachos.pdf?fbclid=IwAR0FZJFHG4DvNlE6LRfzSo2-OteK3uBABVxoU1rgrrMVtO1hVX69gEAngEw*

[4]*Các file hướng dẫn trên moodle – trang web môn học*