

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

---o0o---



BÁO CÁO
PROJECT 1:
HỆ THỐNG QUẢN LÝ TẬP TIN TRÊN WINDOWS

Thành viên nhóm	MSSV
Đỗ Phan Tuấn Đạt	22127057
Phạm Thành Đạt	22127064
Đỗ Đình Hải	22127095
Lê Hồ Phi Hoàng	22127123
Trần Nguyễn Minh Hoàng	22127131

Lớp: 22CLC02
Môn học: Hệ điều hành
Học kỳ: 2
Năm học: 2023-2024

I. Mục lục

I.	Mục lục.....	2
II.	Môi trường làm việc.....	5
1.	Môi trường làm việc:	5
2.	Ngôn ngữ:	5
III.	Tìm hiểu mã nguồn chương trình nachos	5
1.	progtest.cc.....	5
2.	syscall.h	6
3.	exception.cc	6
4.	bitmap.h + bitmap.cc	7
5.	openfile.h	8
6.	translate.h + translate.cc	9
7.	machine.h.....	9
8.	machine.cc	11
a)	Constructor	11
b)	Destructor	12
c)	RaiseException.....	12
d)	Debugger	12
e)	DumpState	13
f)	ReadRegister + WriteRegister.....	13
9.	mipssim.cc	14
a)	Mult	14
b)	TypeToReg	15
c)	Machine::Run	15
d)	Machine::OneInstruction.....	15
10.	console.h + console.cc	21
a)	Constructor	21
b)	Destructor	22
c)	CheckAvail	22

d)	WriteDone	22
e)	GetChar.....	22
f)	PutChar	23
11.	synchconss.h + synchcons.cc	23
a)	SynchReadFunc, SynchWriteFunc.....	23
b)	Constructor không tham số.....	23
c)	Constructor có tham số.....	23
d)	Destructor	24
e)	Write	24
f)	Read.....	24
12.	Test folder.....	25
a)	Halt	25
b)	Shell.....	25
c)	Sort.....	25
d)	Start.....	26
e)	Matmult	26
IV.	Cài đặt xử lý các Exception và các system call nhập xuất	26
1.	IncreasePC	26
2.	Xử lý các Exception không phải system call	27
3.	Chuẩn bị cài đặt các system call	27
a)	Define các system call	27
b)	Khai báo biến toàn cục trong “.code/threads” 2 file “system.h” và “system.cc” ...	27
c)	Thay đổi trong .code/filesys/filesys.h	27
4.	System call ReadInt + PrintInt	28
a)	SC_ReadInt.....	28
b)	SC_PrintInt.....	28
5.	System call ReadFloat + Writefloat	29
a)	SC_ReadFloat.....	29
b)	SC_WriteFloat.....	29
6.	System call ReadChar + WriteChar	30

a) SC_ReadChar	30
b) SC_PrintChar.....	30
7. System call Create	30
8. System call Open + Close.....	31
a) SC_Open.....	31
b) SC_close	31
9. System call Read + Write	32
a) SC_Read	32
b) SC_Write	33
10. System call Seek.....	33
V. Một số chương trình minh họa.....	34
1. Help.....	34
2. ASCII.....	34
3. Quick Sort.....	35
4. Merge Sort	35
VI. Demo sử dụng các chương trình minh họa.....	36
1. Help.....	36
2. ASCII.....	36
3. Quick Sort.....	37
4. Merge Sort	41
VII. Đóng góp	41
VIII. Tài liệu tham khảo	42

II. Môi trường làm việc

1. Môi trường làm việc:

Visual Studio Code trên Ubuntu

2. Ngôn ngữ:

C; C++

III. Tìm hiểu mã nguồn chương trình nachos

1. progtest.cc

Chứa các định nghĩa và hàm dùng để chạy một chương trình người dùng và thử nghiệm việc đọc và ghi từ console, bao gồm:

- **StartProcess**: hàm được sử dụng để chạy một chương trình người dùng. Chức năng chính của StartProcess là mở file thực thi, tạo không gian địa chỉ mới cho chương trình, khởi tạo và khôi phục giá trị đăng ký ban đầu, sau đó chạy chương trình bằng cách gọi `machine->Run()`. Chương trình chỉ trở về khi không gặp lỗi hoặc kết thúc bằng cách gọi syscall "exit".
- **executable**: Một con trỏ đến một tệp mở được sử dụng để đại diện cho chương trình thực thi mà bạn muốn chạy.
- **space**: Một con trỏ đến không gian địa chỉ (`AddrSpace`), đại diện cho không gian bộ nhớ của chương trình thực thi được tạo ra từ tệp executable.
- **console**: Một con trỏ đến một đối tượng Console, được sử dụng để thực hiện giao tiếp với bàn phím và màn hình.
- **readAvail**: Một con trỏ đến Semaphore, được sử dụng để đồng bộ hóa việc đọc từ console.
- **writeDone**: Một con trỏ đến Semaphore, được sử dụng để đồng bộ hóa việc ghi ra console.
- **ReadAvail(int arg)**: Một hàm xử lý ngắt được gọi khi một ký tự được nhập vào từ console, nó thức tỉnh một luồng đang đợi bằng cách tăng giá trị của Semaphore `readAvail`.
- **WriteDone(int arg)**: Một hàm xử lý ngắt được gọi khi một ký tự được ghi ra console, nó thức tỉnh một luồng đang đợi bằng cách tăng giá trị của Semaphore `writeDone`.
- **ConsoleTest(char *in, char *out)**: Một hàm để thử nghiệm việc đọc và ghi từ console. Nó sẽ lặp lại việc đọc một ký tự từ console, sau đó ghi ký tự đó ra console và lặp lại quá trình cho đến khi người dùng nhập ký tự 'q' để thoát

2. syscall.h

Chứa một số khai báo hằng số, kiểu dữ liệu và các hàm cần thiết để giao tiếp giữa các chương trình người dùng và hạt nhân Nachos thông qua các cuộc gọi hệ thống. Dưới đây là mô tả ngắn gọn về nội dung của file:

- Hàm syscall: Định nghĩa các hàm syscall mà các chương trình người dùng có thể gọi để tương tác với hệ điều hành Nachos. Đối với mỗi hàm syscall, có một hằng số (ví dụ: SC_Halt, SC_Exit, SC_Exec, vv.) để xác định loại syscall khi gọi từ một chương trình người dùng.
- Hàm Halt(): Dừng Nachos và in ra các thống kê hiệu suất.
- Hàm Exit(int status): Kết thúc chương trình người dùng, với status chỉ ra trạng thái kết thúc.
- Hàm Exec(char *name): Chạy một chương trình mới từ tệp đã cho và trả về một định danh không gian địa chỉ.
- Hàm Join(SpaceId id): Chờ đến khi một chương trình con đã thực hiện xong và trả về trạng thái kết thúc của nó.
- Hàm Create(char *name): Tạo một tệp Nachos mới.
- Hàm Open(char *name): Mở một tệp Nachos và trả về một định danh tệp mở.
- Hàm Write(char *buffer, int size, OpenFileId id): Ghi các byte từ bộ đệm vào tệp đã mở.
- Hàm Read(char *buffer, int size, OpenFileId id): Đọc các byte từ tệp đã mở vào bộ đệm.
- Hàm Close(OpenFileId id): Đóng tệp đã mở.
- Hàm Fork(void (*func)()): Tạo một luồng mới để chạy một hàm trong không gian địa chỉ hiện tại.
- Hàm Yield(): Nhường CPU cho một luồng khác có thể chạy, cho dù nó ở trong không gian địa chỉ hiện tại hoặc không.

3. exception.cc

Chủ yếu tập trung vào xử lý các ngoại lệ từ các chương trình người dùng, đặc biệt là xử lý syscall để kết thúc chương trình Nachos:

- ExceptionHandler(ExceptionType which): Đây là hàm xử lý ngoại lệ (exception handler) chính của hệ điều hành Nachos. Hàm này được gọi khi một chương trình người dùng gây ra một ngoại lệ (exception), bao gồm cả các syscall và các ngoại lệ bộ xử lý (processor exceptions) như lỗi truy cập bộ nhớ không hợp lệ hoặc lỗi toán học.

- Nếu ngoại lệ là một syscall và loại syscall là SC_Halt, có nghĩa là chương trình người dùng muốn kết thúc chương trình Nachos. Trong trường hợp này, Nachos sẽ gọi hàm interrupt->Halt() để dừng hệ thống.
- Nếu không phải là một syscall hợp lệ hoặc ngoại lệ không được xử lý, chương trình Nachos sẽ in ra thông báo lỗi và kết thúc chương trình Nachos bằng cách gọi hàm ASSERT(FALSE).

4. bitmap.h + bitmap.cc

Chứa class BitMap, là một lớp được thiết kế để quản lý một mảng các bit, trong đó mỗi bit có thể được đặt thành ON (1) hoặc OFF (0). Lớp này thường được sử dụng để quản lý tài nguyên trong hệ thống, như các khối đĩa hoặc trang bộ nhớ, bằng cách đánh dấu chúng là đã được sử dụng hoặc chưa được sử dụng.

Các thuộc tính chính:

- “numbits”: số lượng bits trong bitmap
- “numWords”: số lượng words cần thiết để lưu trữ bitmap, dựa trên “numBits”
- “map”: con trỏ đến mảng các words, mỗi word lưu trữ 32 bit trên hệ thống.

Các phương thức chính:

- “BitMap(int nitems)”: khởi tạo bitmap với “nitems” bits, với tất cả các bits được thiết lập về 0 (clear).
- Destructor “~BitMap()”: giải phóng bộ nhớ được cấp phát cho bitmap.
- “void Mark(int which)”: đặt bit ở vị trí “which” thành 1.
- “void Clear(int which)”: xóa bit ở vị trí “which”, thiết lập nó về 0.
- “bool Test(int which)”: kiểm tra xem bit ở vị trí “which” có đang được set 1 hay không. Trả về “true” nếu bit đó được set.
- “int Find()”: Tìm bit đầu tiên không được set 0 trong bitmap, đặt nó thành 1 và trả về chỉ số của bit đó. Nếu tất cả các bit đều đã được set, phương thức trả về -1.
- “int NumClear()”: đếm và trả về số lượng bits chưa được set 0 trong bitmap.
- “void Print()”: in chỉ số của tất cả các bits được set 1 trong bitmap, dùng cho mục đích debug.
- “void FetchFrom(OpenFile* file)”: đọc bitmap từ file.
- “void WriteBack(OpenFile* file)”: ghi bitmap vào file.

5. `openfile.h`

- Định nghĩa các cấu trúc dữ liệu và giao diện cho việc mở, đóng, đọc, và viết vào các file riêng lẻ. Các thao tác được hỗ trợ tương tự như các thao tác trong UNIX. File này cung cấp hai triển khai: một là “STUB” dùng cho mô phỏng, chuyển các thao tác file trực tiếp thành các thao tác UNIX tương ứng, và một là triển khai “thực” chuyển các thao tác này thành các yêu cầu đọc/ghi các sector trên đĩa. Trong triển khai cơ bản của hệ thống tệp, không xử lý truy cập đồng thời vào hệ thống file từ nhiều luồng khác nhau.
- Lớp “OpenFile” (STUB): phần này được sử dụng khi “FILESYS_STUB” được định nghĩa, nghĩa là triển khai này chủ yếu là một “stub” tạm thời chuyển các thao tác hệ thống tệp Nachos sang các thao tác tệp UNIX.
 - Constructor: khởi tạo một file với đối số là file descriptor “f”. “currentOffset” được khởi tạo là 0.
 - Destructor: đóng file.
 - “ReadAt”: đọc “numBytes” bytes từ file vào buffer “into” bắt đầu từ vị trí “position” không thay đổi “currentOffset”.
 - “WriteAt”: ghi “numBytes” bytes từ buffer “from” vào file bắt đầu từ vị trí “position” không thay đổi “currentOffset”.
 - “Read”: đọc “numBytes” bytes từ file vào buffer “into” bắt đầu từ “currentOffset” và cập nhật “currentOffset”.
 - “Write”: ghi “numbytes” bytes từ buffer “from” vào file bắt đầu từ “currentOffset” và cập nhật “currentOffset”.
 - “Length”: trả về độ dài của file.
- Lớp “OpenFile” (FILESYS): phần này triển khai “thực” của “OpenFile”, dùng khi “FILESYS” được định nghĩa, tức là triển khai này chuyển các thao tác vào file thành các yêu cầu đọc/ghi sector trên đĩa.
 - Constructor: mở một file có header nằm ở sector chỉ định trên đĩa.
 - Destructor: đóng file
 - “Seek”: đặt vị trí từ đó và bắt đầu đọc/ghi – tương tự như lseek trong UNIX.
 - “Read”/”Write”: Đọc/ghi “numBytes” bytes từ/đến file, bắt đầu từ vị trí hiện tại. Trả về số byte thực sự được đọc/ghi và cập nhật vị trí trong file.
 - “ReadAt”/”WriteAt”: đọc/ghi “numBytes” từ/đến file mà không qua vị trí hiện tại. Điều này cho phép truy cập trực tiếp tới một vị trí cụ thể trong file mà không thay đổi vị trí đọc/ghi hiện tại.
 - “Length”: trả về số lượng bytes trong file.

6. translate.h + translate.cc

- Cung cấp các cấu trúc dữ liệu và hàm để quản lý việc dịch từ số trang ảo sang số trang vật lý, được sử dụng để quản lý bộ nhớ vật lý cho các chương trình người dùng
- Lớp “TranslationEntry”: là một entry trong bảng dịch, được sử dụng cho cả bảng trang và bộ đệm dịch (TLB). Mỗi entry định nghĩa một ánh xạ từ một số trang ảo tới một số trang vật lý. Ngoài ra, có một số bit phụ trợ để kiểm soát truy cập (valid và read-only) và một số bit để lưu thông tin sử dụng (use và dirty).
 - “virtualPage”: số trang ảo trong bộ nhớ ảo.
 - “physicalPage”: số trang vật lý trong bộ nhớ vật lý (tính từ đầu của “mainMemory”).
 - “valid”: đánh dấu xem entry này có hợp lệ không. Nếu bit này được set, thì dịch sẽ bị bỏ qua.
 - “readOnly”: đánh dấu xem trang này có được phép chỉ đọc không.
 - “use”: đánh dấu xem trang này đã được tham chiếu (sử dụng) hay chưa.
 - “dirty”: đánh dấu xem trang này đã được sửa đổi (dirty) hay chưa.
- Lớp “Machine”:
 - “ReadMem”: đọc “size” (1,2,4) byte từ bộ nhớ ảo tại địa chỉ “addr” vào vị trí được trỏ bởi “value”. Phương thức này trả về FALSE nếu quá trình dịch từ bộ nhớ ảo sang bộ nhớ vật lý thất bại.
 - “WriteMem”: ghi “size” byte từ dữ liệu được trỏ bởi “value” vào bộ nhớ ảo tại địa chỉ “addr”. Phương thức này trả về FALSE nếu quá trình dịch từ bộ nhớ ảo sang bộ nhớ vật lý thất bại.
 - “Translate”: dịch một địa chỉ bộ nhớ ảo thành một địa chỉ bộ nhớ vật lý, sử dụng entries trong bảng trang hoặc TLB. Nếu mọi thứ đều ổn, phương thức này sẽ cài đặt các bit use/dirty trong entry của bảng dịch và lưu địa chỉ bộ nhớ vật lý được dịch vào “physAddr”. Nếu có lỗi, nó sẽ trả về loại ngoại lệ.

7. machine.h

Chứa các định nghĩa và cấu trúc giữ liệu để mô phỏng các phần cứng của thiết bị chạy hệ điều hành nachos khi thực thi chương trình người dùng. Bao gồm:

- Định nghĩa kích thước và định dạng bộ nhớ người dùng: Định nghĩa kích thước trang (PageSize), số trang vật lý (NumPhysPages), và kích thước bộ nhớ (MemorySize). Các định nghĩa này quy định kích thước và định dạng của bộ nhớ mà các chương trình người dùng sẽ chạy trên đó.

- Định nghĩa các loại ngoại lệ (ExceptionType): một enum để biểu diễn các loại ngoại lệ có thể xảy ra trong quá trình thực thi chương trình người dùng.
- Định nghĩa một số hằng số và tên thanh ghi cho việc thao tác với bộ xử lý MIPS (Microprocessor without Interlocked Pipeline Stages), một kiến trúc bộ xử lý thường được sử dụng trong các hệ thống nhúng và máy tính cá nhân. Bao gồm:
 - StackReg: Định danh cho thanh ghi trỏ đến ngăn xếp của người dùng. Trong một chương trình, thanh ghi này thường được sử dụng để lưu trữ địa chỉ của ngăn xếp hiện tại của chương trình.
 - RetAddrReg: Thanh ghi này lưu trữ địa chỉ trả về cho các cuộc gọi hàm (return address). Khi một hàm được gọi, địa chỉ trả về của hàm gọi sẽ được lưu vào thanh ghi này để quay lại vị trí sau lệnh gọi hàm.
 - NumGPRegs: Định nghĩa số lượng thanh ghi tổng cộng trên bộ xử lý MIPS. Trong trường hợp này, có 32 thanh ghi tổng cộng, bao gồm các thanh ghi tổng quát (general-purpose registers) và một số thanh ghi đặc biệt khác.
 - HiReg và LoReg: Đây là các thanh ghi đặc biệt được sử dụng trong quá trình thực hiện các phép nhân (multiply) và chia (divide) trên bộ xử lý MIPS. Kết quả của phép nhân hoặc chia được lưu vào các thanh ghi này.
 - PCReg, NextPCReg, PrevPCReg: Các thanh ghi này lưu trữ các giá trị liên quan đến việc thực hiện lệnh trên bộ xử lý MIPS. PCReg lưu trữ địa chỉ của lệnh đang được thực hiện, NextPCReg lưu trữ địa chỉ của lệnh tiếp theo (dành cho trường hợp thực hiện các nhảy) và PrevPCReg được sử dụng trong quá trình gỡ rối để lưu trữ địa chỉ của lệnh trước đó.
 - LoadReg và LoadValueReg: Các thanh ghi này được sử dụng trong quá trình thực hiện các lệnh load trễ (delayed load) trên bộ xử lý MIPS. LoadReg lưu trữ thanh ghi mục tiêu của lệnh load trễ và LoadValueReg lưu trữ giá trị cần được load vào thanh ghi đó.
 - BadVAddrReg: Thanh ghi này lưu trữ địa chỉ ảo không hợp lệ trong trường hợp xảy ra ngoại lệ (exception) liên quan đến địa chỉ.
 - NumTotalRegs: Đây là số lượng toàn bộ thanh ghi trên bộ xử lý MIPS, bao gồm các thanh ghi tổng quát và các thanh ghi đặc biệt.
- class Instruction: đại diện cho một lệnh trong chương trình máy, được biểu diễn ở cả hai dạng: dạng nhị phân chưa được giải mã và đã được giải mã để xác định các thao tác cần thực hiện, các thanh ghi cần thao tác và bất kỳ giá trị toán hạng ngay lập tức nào. Bao gồm:
 - Phương thức Decode(): Phương thức này được sử dụng để giải mã biểu diễn nhị phân của lệnh, từ đó xác định loại lệnh, các thanh ghi liên quan và các giá trị toán hạng ngay lập tức.
 - Biến value: Biến này lưu trữ biểu diễn nhị phân của lệnh.

- Các biến opCode, rs, rt, rd và extra: Các biến này lưu trữ thông tin chi tiết về lệnh đã được giải mã.
 - opCode: Định danh loại lệnh, không phải là trường opcode thực sự của lệnh, mà là một biến để xác định loại lệnh.
 - rs, rt, rd: Là các thanh ghi từ lệnh, được sử dụng cho các thao tác.
 - extra: Là giá trị toán hạng ngay lập tức, hoặc một giá trị bổ sung như địa chỉ đích hoặc shamt field.
- class Machine: biểu diễn trạng thái của CPU và bộ nhớ của chương trình người dùng. Bao gồm các phương thức để thực thi chương trình, đọc và ghi các thanh ghi CPU, thực hiện việc dịch địa chỉ bộ nhớ, và xử lý các ngoại lệ. Lớp này cũng chứa các cấu trúc dữ liệu như bộ nhớ chính, các thanh ghi CPU, bảng dịch địa chỉ (page table), và TLB (Translation Lookaside Buffer).
- Các hàm tiện ích cho việc chuyển đổi định dạng

8. machine.cc

Cài đặt một số hàm từ file machine.h

a) Constructor

- Trong hàm, tất cả các thanh ghi CPU được khởi tạo với giá trị 0.
- Một mảng mainMemory được cấp phát để biểu diễn bộ nhớ vật lý mà các chương trình người dùng sẽ chạy.
- Nếu được định nghĩa sử dụng TLB (USE_TLB), một mảng các mục TLB cũng được khởi tạo với trạng thái không hợp lệ.
- Cuối cùng, hàm gọi CheckEndian() để kiểm tra endian của hệ thống. Hàm CheckEdian() có nội dung như sau:
 - Hàm này được sử dụng để kiểm tra xem hệ thống hiện tại đang chạy trên kiến trúc big-endian hay little-endian.
 - Trong hàm, một biến kiểu union được khai báo để kiểm tra cả hai dạng endian.
 - Union này có hai thành phần: charword là một mảng 4 ký tự và intword là một số nguyên không dấu.
 - Giá trị các phần tử trong mảng charword được gán giá trị tăng dần từ 1 đến 4.
 - Sau đó, giá trị của intword được so sánh với một giá trị đã xác định để xác định endian của hệ thống.
 - Nếu hệ thống là big-endian, giá trị của intword sẽ là 0x01020304, ngược lại, nó sẽ là 0x04030201.
- Đặc biệt, nếu biến debug là true, chương trình sẽ rơi vào chế độ debug sau mỗi lệnh của chương trình người dùng.

b) Destructor

- Giải phóng bộ nhớ được cấp phát trước đó cho mảng mainMemory bằng lệnh delete[], ngăn chặn rò rỉ bộ nhớ.
- Kiểm tra xem con trỏ tlb có khác NULL không. Nếu tlb không trỏ đến NULL, tức là đã sử dụng TLB (USE_TLB đã được định nghĩa), thì mảng các mục TLB cũng được giải phóng bằng cách sử dụng toán tử delete[].

c) RaiseException

- Xử lý các ngoại lệ (exceptions) xảy ra trong quá trình thực thi của chương trình người dùng trên hệ thống Nachos.
- DEBUG statement:
 - Ghi thông điệp debug vào log nếu DEBUG mode được kích hoạt với chế độ m.
 - Thông điệp này cung cấp thông tin về loại ngoại lệ (exception) đã xảy ra thông qua exceptionNames[which].
- Gán giá trị cho thanh ghi BadVAddrReg:
 - Thanh ghi BadVAddrReg là thanh ghi đặc biệt được sử dụng để lưu trữ địa chỉ ảo (virtual address) gây ra ngoại lệ.
 - Trong hàm này, giá trị badVAddr (địa chỉ ảo gây ra ngoại lệ) được gán cho thanh ghi BadVAddrReg.
- Gọi hàm DelayedLoad(0, 0):
 - Hàm DelayedLoad() được gọi để hoàn thành bất kỳ thao tác nạp trễ (delayed load) nào đang tiến hành.
 - Trong trường hợp này, hai đối số của hàm là 0, điều này có nghĩa là không có thao tác nạp trễ nào cần hoàn thành.
- Đặt trạng thái của interrupt sang SystemMode, ngăn chặn việc các ngoại lệ tiếp theo xảy ra trong quá trình xử lý ngoại lệ hiện tại.
- Gọi hàm ExceptionHandler(which) để xử lý ngoại lệ đã xảy ra.
 - Thông thường, sau khi xử lý xong ngoại lệ, các ngoại lệ khác có thể được kích hoạt (nếu có), và các tác vụ tiếp theo sẽ được tiếp tục thực thi.
- Đặt trạng thái của interrupt sang UserMode: Sau khi xử lý xong ngoại lệ, trạng thái của interrupt được đặt lại sang UserMode, cho phép các ngoại lệ tiếp theo (nếu có) được kích hoạt và thực thi trong chế độ người dùng.

d) Debugger

- Chạy trình gỡ lỗi (debugger) của hệ thống Nachos.
- Cấp phát bộ nhớ cho biến buf: Một mảng ký tự có độ dài 80 được cấp phát để lưu trữ dòng lệnh nhập từ người dùng.
- Gọi hàm interrupt->DumpState(): in ra trạng thái của các interrupt hiện tại.

- Gọi hàm `DumpState()`: của lớp `Machine` được để in ra trạng thái của CPU và bộ nhớ của hệ thống Nachos.
- In ra số tick tổng cộng tính từ khi hệ thống Nachos được khởi động được in ra để người dùng biết được thời gian đã trôi qua.
- Nhận dòng lệnh từ người dùng:
 - Sử dụng hàm `fgets()` để đọc dòng lệnh từ người dùng và lưu vào biến `buf`.
 - Hàm này sẽ đợi đến khi người dùng nhập dữ liệu, và lưu trữ nó trong `buf`.
- Xử lý dòng lệnh nhập từ người dùng:
 - Nếu dòng lệnh có thể được chuyển đổi thành số nguyên, nó sẽ được gán vào biến `runUntilTime`, cho phép chạy đến thời điểm cụ thể.
 - Nếu không, dòng lệnh sẽ được xử lý dựa trên nội dung:
 - Nếu người dùng chỉ nhập `Enter`, không có hành động nào sẽ được thực hiện.
 - Nếu người dùng nhập `'c'`, chế độ chạy từng bước (single step) sẽ được tắt.
 - Nếu người dùng nhập `'?'`, một thông điệp trợ giúp sẽ được in ra, liệt kê các lệnh sử dụng được hỗ trợ bởi trình gỡ lỗi.
- Giải phóng bộ nhớ cho biến `buf` để tránh rò rỉ bộ nhớ.

e) `DumpState`

- In ra trạng thái của CPU của chương trình người dùng trên hệ thống Nachos.
- In ra các thanh ghi tổng quát:
 - Vòng lặp `for` được sử dụng để duyệt qua tất cả các thanh ghi tổng quát (general purpose registers) của CPU.
 - Mỗi thanh ghi được in ra dưới dạng số thập lục phân (hexadecimal) bên cạnh chỉ số của thanh ghi.
 - Có một số lệnh `switch` để xử lý đặc biệt cho các thanh ghi `StackReg` và `RetAddrReg`.
 - Các thanh ghi được in ra dưới dạng cột, mỗi dòng chứa tối đa 4 thanh ghi.
- In ra các thanh ghi đặc biệt:
 - Các thanh ghi đặc biệt như `Hi`, `Lo`, `PC`, `NextPC`, `PrevPC`, `Load`, và `LoadValue` cũng được in ra.
 - Mỗi thanh ghi được in ra dưới dạng số thập lục phân (hexadecimal) cùng với tên của nó.
- Cuối cùng, hàm in ra một dòng trống để tạo ra một định dạng in đẹp hơn và dễ đọc hơn.

f) `ReadRegister + WriteRegister`

- Đọc và ghi giá trị vào các thanh ghi (registers) của chương trình người dùng

- Hàm `ReadRegister(int num)`:
 - Nhận vào một số nguyên `num` là chỉ số của thanh ghi muốn đọc.
 - Kiểm tra xem số `num` có nằm trong khoảng từ 0 đến `NumTotalRegs - 1` không, trong đó `NumTotalRegs` là số lượng tổng cộng của các thanh ghi.
 - Nếu `num` nằm trong phạm vi hợp lệ, hàm trả về giá trị của thanh ghi tương ứng với chỉ số `num`.
- Hàm `WriteRegister(int num, int value)`:
 - Nhận vào hai đối số: `num` là chỉ số của thanh ghi muốn ghi giá trị vào và `value` là giá trị muốn ghi.
 - Tương tự như `ReadRegister()`, hàm này cũng kiểm tra xem `num` có nằm trong phạm vi hợp lệ hay không.
 - Nếu `num` hợp lệ, giá trị `value` sẽ được gán cho thanh ghi tương ứng với chỉ số `num`.

9. mipssim.cc

Chứa các hàm dùng để mô phỏng hệ thống xử lý của MIPS R2/3000. Bao gồm:

a) Mult

- Nhân 2 số nguyên `a` và `b`.
- Kiểm tra nếu một trong hai số `a` hoặc `b` bằng 0, thì kết quả của phép nhân cũng sẽ là 0. Trong trường hợp này, cả `hiPtr` và `loPtr` đều được gán bằng 0 và hàm kết thúc.
- Xác định dấu của kết quả nếu sử dụng phép tính có dấu:
 - Nếu `signedArith` được đặt là `true`, tức là phép tính sử dụng dấu, hàm này xác định dấu của kết quả bằng cách kiểm tra dấu của `a` và `b`. Nếu `a` hoặc `b` là số âm, biến `negative` sẽ được gán `true` và tương ứng với giá trị tuyệt đối của `a` và `b`.
- Thực hiện phép nhân:
 - Trong vòng lặp, mỗi bit của `a` sẽ được kiểm tra. Nếu bit tại vị trí đó của `a` là 1, thì giá trị của `b` sẽ được cộng vào `lo`.
 - Tại mỗi vòng lặp, `b` sẽ được dịch trái một bit và nếu bit cao nhất của `b` là 1, thì bit đó sẽ được chuyển vào `hi`.
 - Vòng lặp này được lặp lại 32 lần, tương ứng với 32 bit của `a`.
- Xử lý dấu của kết quả:
 - Nếu biến `negative` được đặt, kết quả sẽ được tính theo dạng bù hai của kết quả bằng cách đảo ngược tất cả các bit trong `hi` và `lo`, sau đó tăng `lo` lên một đơn vị. Nếu `lo` bằng 0 sau khi tăng, thì `hi` cũng sẽ được tăng lên một đơn vị.
- Gán kết quả cho `hiPtr` và `loPtr` thông qua con trỏ, để có thể được sử dụng sau khi kết thúc hàm.

b) TypeToReg

- Truy xuất số thanh ghi (register) được tham chiếu trong một lệnh (instruction).
- Nhận vào loại thanh ghi và con trỏ đến một đối tượng của class Instruction, trả về số nguyên là giá trị của thanh ghi tương ứng hoặc -1 nếu giá trị không hợp lệ /không xác định

c) Machine::Run

- Mô phỏng việc thực thi một chương trình cấp người dùng trên nachos
- Khởi tạo biến lưu trữ lệnh đã giải mã:
 - Biến instr là một con trỏ được khởi tạo để lưu trữ các lệnh đã được giải mã trong quá trình thực thi.
- Hiện thị thông báo khởi đầu (nếu có Debug đang được bật):
 - Nếu chế độ Debug cho máy ('m') được bật, hàm sẽ in ra một thông báo báo cáo bắt đầu thực thi một luồng (thread) với tên và thời gian hiện tại.
- Thiết lập trạng thái chế độ người dùng và vòng lặp vô hạn:
 - Hàm đặt trạng thái của ngắt sang chế độ người dùng để bắt đầu thực thi chương trình người dùng.
 - Vòng lặp vô hạn được sử dụng để thực thi các lệnh từ chương trình người dùng.
 - Trong mỗi lần lặp, hàm OneInstruction(instr) được gọi để thực hiện một lệnh từ chương trình.
 - Sau mỗi lần lặp, hàm interrupt->OneTick() được gọi để mô phỏng thời gian trôi qua.
 - Nếu biến singleStep được đặt và thời gian thực thi đạt đến giá trị được chỉ định trong runUntilTime, hàm Debugger() được gọi để dừng thực thi và chuyển sang chế độ gỡ lỗi.

d) Machine::OneInstruction

- Thực thi một lệnh từ chương trình người dùng
- Khai báo biến cục bộ:
 - int raw: Biến này được sử dụng để lưu trữ giá trị nhị phân của lệnh được đọc từ bộ nhớ.
 - int nextLoadReg và int nextLoadValue: Các biến này được sử dụng để ghi nhận thao tác tải trễ (delayed load operation), để áp dụng trong tương lai.
- Lấy lệnh từ bộ nhớ:
 - Hàm ReadMem() được gọi để đọc 4 byte từ bộ nhớ, bắt đầu từ địa chỉ được lưu trong thanh ghi PCReg.
 - Nếu không thành công trong việc đọc (trả về false), có thể là do xảy ra ngoại lệ, hàm kết thúc ngay lập tức.

- Giải mã lệnh: Giá trị đọc được từ bộ nhớ (trong biến raw) được gán cho instr->value, sau đó được giải mã để xác định loại lệnh, các thanh ghi tương ứng, và giá trị ngay (immediate value), nếu có.
- In thông tin debug (nếu Debug đang được bật):
 - Nếu chế độ Debug cho máy ('m') được kích hoạt, thông tin về lệnh đang được thực thi sẽ được in ra.
 - Thông tin này bao gồm giá trị của thanh ghi và loại lệnh đang thực thi, được trích xuất từ instr, cùng với vị trí hiện tại của chương trình (địa chỉ được lưu trong thanh ghi PCReg).
- Tính toán địa chỉ của lệnh tiếp theo: Biến pcAfter được sử dụng để tính toán địa chỉ của lệnh tiếp theo sau lệnh hiện tại bằng cách cộng thêm 4 đơn vị vào giá trị hiện tại của thanh ghi NextPCReg, đại diện cho địa chỉ của lệnh tiếp theo trong bộ nhớ.
- Khai báo các biến cục bộ:
 - Các biến sum, diff, tmp, value được khai báo để sử dụng trong quá trình thực thi các lệnh.
 - Các biến rs, rt, imm được khai báo để lưu trữ các giá trị của các thanh ghi (rs và rt) và giá trị ngay (imm).
- Thực thi các lệnh:
 - Sử dụng một câu lệnh switch-case để xác định loại lệnh (opCode) và thực hiện các thao tác tương ứng.
 - Mỗi case trong switch tương ứng với một loại lệnh MIPS.
 - Các lệnh được thực thi bao gồm:
 - OP_ADD, OP_ADDI, OP_ADDIU, OP_ADDU: Thực hiện phép cộng.
 - OP_AND, OP_ANDI: Thực hiện phép AND bit.
- Kiểm tra tràn số:
 - Trong mỗi lệnh cộng (OP_ADD, OP_ADDI, OP_ADDIU, OP_ADDU), trước khi gán giá trị vào thanh ghi đích, một kiểm tra tràn số được thực hiện.
 - Nếu có tràn số xảy ra (tức là kết quả có dấu của phép cộng khác với dấu của từng phần tử tham gia phép cộng), một ngoại lệ OverflowException sẽ được kích hoạt bằng cách gọi hàm RaiseException().
- Gán giá trị cho thanh ghi đích: Kết quả của phép tính được gán vào thanh ghi đích tương ứng, như là thanh ghi đích của lệnh đó.
- OP_BEQ (Branch if Equal): Nếu giá trị của thanh ghi rs và rt bằng nhau, chương trình sẽ nhảy tới địa chỉ được tính bằng cách cộng IndexToAddr(instr->extra) vào giá trị của thanh ghi NextPCReg.
- OP_BGEZAL (Branch if Greater Than or Equal to Zero And Link): Thực hiện tương tự như OP_BGEZ nhưng cũng ghi lại địa chỉ lệnh tiếp theo vào thanh ghi R31 (hoặc RetAddrReg).

- OP_BGEZ (Branch if Greater Than or Equal to Zero): Nếu giá trị của thanh ghi rs không có bit dấu, chương trình sẽ nhảy tới địa chỉ được tính bằng cách cộng IndexToAddr(instr->extra) vào giá trị của thanh ghi NextPCReg.
- OP_BGTZ (Branch if Greater Than Zero): Nếu giá trị của thanh ghi rs lớn hơn 0, chương trình sẽ nhảy tới địa chỉ được tính bằng cách cộng IndexToAddr(instr->extra) vào giá trị của thanh ghi NextPCReg.
- OP_BLEZ (Branch if Less Than or Equal to Zero): Nếu giá trị của thanh ghi rs nhỏ hơn hoặc bằng 0, chương trình sẽ nhảy tới địa chỉ được tính bằng cách cộng IndexToAddr(instr->extra) vào giá trị của thanh ghi NextPCReg.
- OP_BLTZAL (Branch if Less Than Zero And Link): Thực hiện tương tự như OP_BLTZ nhưng cũng ghi lại địa chỉ lệnh tiếp theo vào thanh ghi R31 (hoặc RetAddrReg).
- OP_BLTZ (Branch if Less Than Zero): Nếu giá trị của thanh ghi rs có bit dấu, chương trình sẽ nhảy tới địa chỉ được tính bằng cách cộng IndexToAddr(instr->extra) vào giá trị của thanh ghi NextPCReg.
- OP_BNE (Branch if Not Equal): Nếu giá trị của thanh ghi rs và rt không bằng nhau, chương trình sẽ nhảy tới địa chỉ được tính bằng cách cộng IndexToAddr(instr->extra) vào giá trị của thanh ghi NextPCReg.
- OP_DIV (Divide): Thực hiện phép chia có dấu của giá trị của thanh ghi rs cho giá trị của thanh ghi rt, kết quả được gán vào thanh ghi LoReg là phần dư và vào thanh ghi HiReg là phần thương.
- OP_DIVU (Divide Unsigned): Thực hiện phép chia không dấu của giá trị của thanh ghi rs cho giá trị của thanh ghi rt, kết quả được gán vào thanh ghi LoReg là phần dư và vào thanh ghi HiReg là phần thương.
- OP_JAL:
 - Giá trị của thanh ghi R31 được gán là địa chỉ của lệnh tiếp theo sau khi gọi hàm IndexToAddr để tính địa chỉ nhảy tới.
 - Sau đó, giá trị của pcAfter được cập nhật với địa chỉ tính toán của lệnh nhảy (OP_JAL chia thành trường hợp OP_J sau khi gán thanh ghi R31).
- OP_J: Giá trị của pcAfter được cập nhật với địa chỉ được tính toán bằng cách kết hợp phần trên của địa chỉ hiện tại với giá trị của trường extra của lệnh (instr->extra).
- OP_JALR:
 - Giá trị của thanh ghi instr->rd được gán bằng địa chỉ của lệnh tiếp theo sau khi gọi hàm IndexToAddr.
 - Sau đó, giá trị của pcAfter được cập nhật bằng giá trị của thanh ghi instr->rs.
- OP_JR: Giá trị của pcAfter được cập nhật bằng giá trị của thanh ghi instr->rs.
- OP_LB, OP_LBU, OP_LH, OP_LHU:
 - Đây là các lệnh đọc dữ liệu từ bộ nhớ (load byte/halfword).

- Đầu tiên, tính toán địa chỉ bằng cách cộng giá trị của thanh ghi `instr->rs` với giá trị ngay (`instr->extra`).
- Sau đó, đọc dữ liệu từ bộ nhớ tại địa chỉ tính toán và gán giá trị vào biến `value`.
- Trong trường hợp của `OP_LB` và `OP_LH`, kiểm tra dấu của giá trị đọc được và mở rộng nếu cần thiết.
- Gán giá trị đọc được vào thanh ghi tiếp theo và lưu giá trị và thanh ghi vào `nextLoadValue` và `nextLoadReg` để thực hiện `delayed load`.
- **OP_LUI:**
 - Nạp giá trị ngay 16-bit vào 16 bit trọng trong thanh ghi.
 - Giá trị của thanh ghi `instr->rt` được gán bằng giá trị ngay được dịch trái 16 bit (`instr->extra << 16`).
- **OP_LW:** Đọc một từ (4 byte) từ bộ nhớ.
 - Đầu tiên, tính toán địa chỉ bằng cách cộng giá trị của thanh ghi `instr->rs` với giá trị ngay (`instr->extra`).
 - Kiểm tra xem địa chỉ tính toán có được cấp phát cho một từ không bằng cách kiểm tra 2 bit cuối cùng.
 - Đọc dữ liệu từ bộ nhớ tại địa chỉ tính toán và gán giá trị vào biến `value`.
 - Gán giá trị đọc được vào thanh ghi tiếp theo và lưu giá trị và thanh ghi vào `nextLoadValue` và `nextLoadReg` để thực hiện `delayed load`.
- **OP_LWL:** Đọc một từ (4 byte) từ bộ nhớ bắt đầu từ địa chỉ có phần cuối của từ được lưu vào thanh ghi.
 - Đầu tiên, tính toán địa chỉ bằng cách cộng giá trị của thanh ghi `instr->rs` với giá trị ngay (`instr->extra`).
 - Kiểm tra xem địa chỉ tính toán có được cấp phát cho một từ không bằng cách kiểm tra 2 bit cuối cùng.
 - Đọc dữ liệu từ bộ nhớ tại địa chỉ tính toán và gán giá trị vào biến `value`.
 - Thực hiện việc cập nhật giá trị của `nextLoadValue` bằng cách lấy từng byte từ giá trị đọc được và thêm vào `nextLoadValue` theo đúng vị trí.
 - Gán giá trị `nextLoadReg` bằng thanh ghi `instr->rt` để thực hiện `delayed load`.
- **OP_LWR:** Đọc một từ (4 byte) từ bộ nhớ kết thúc tại địa chỉ có phần đầu của từ được lưu vào thanh ghi.
 - Các bước thực hiện tương tự như `OP_LWL`, nhưng giá trị từ bộ nhớ được đọc từ phải sang trái và cập nhật vào `nextLoadValue` tương ứng.
- **OP_MFHI:** sao chép giá trị của thanh ghi đầu kết quả (Hi) vào thanh ghi đích được chỉ định bởi `instr->rd`.
- **OP_MFLO:** sao chép giá trị của thanh ghi thứ hai kết quả (Lo) vào thanh ghi đích được chỉ định bởi `instr->rd`.
- **OP_MTHI:** sao chép giá trị của thanh ghi nguồn (`instr->rs`) vào thanh ghi Hi.

- OP_MTLO: sao chép giá trị của thanh ghi nguồn (instr->rs) vào thanh ghi Lo.
- OP_MULT: thực hiện phép nhân giữa hai toán tử (đã được ký hiệu) và lưu kết quả trong cặp thanh ghi Hi và Lo. Điều này được thực hiện bằng cách gọi hàm Mult với signedArith được đặt là TRUE.
- OP_MULTU: thực hiện phép nhân không dấu giữa hai toán tử (đã được ký hiệu) và lưu kết quả trong cặp thanh ghi Hi và Lo. Điều này được thực hiện bằng cách gọi hàm Mult với signedArith được đặt là FALSE.
- OP_NOR: thực hiện phép NOT (OR bitwise), tức là lấy phủ định của giá trị kết quả của phép OR giữa hai toán hạng, sau đó gán kết quả vào thanh ghi đích được chỉ định bởi instr->rd.
- OP_OR: thực hiện phép OR bitwise giữa hai toán hạng và lưu kết quả vào thanh ghi đích được chỉ định bởi instr->rd.
- OP_ORI: thực hiện phép OR bit với một giá trị số nguyên nhỏ và lưu kết quả vào thanh ghi đích được chỉ định bởi instr->rt.
- OP_SB: ghi một byte từ thanh ghi rt vào bộ nhớ tại địa chỉ được tính toán bằng cách thêm extra vào giá trị của thanh ghi rs.
- OP_SH: ghi một từ (2 byte) từ thanh ghi rt vào bộ nhớ tại địa chỉ được tính toán bằng cách thêm extra vào giá trị của thanh ghi rs.
- OP_SLL: thực hiện phép dịch trái logic (shift left logical) trên giá trị của thanh ghi rt và lưu kết quả vào thanh ghi đích được chỉ định bởi instr->rd.
- OP_SLLV: thực hiện phép dịch trái logic (shift left logical) trên giá trị của thanh ghi rt với một số lượng bit được xác định bởi giá trị của thanh ghi rs, sau đó lưu kết quả vào thanh ghi đích được chỉ định bởi instr->rd.
- OP_SLT: so sánh giữa hai toán hạng (rs và rt) và gán 1 vào thanh ghi đích nếu rs nhỏ hơn rt, ngược lại gán 0.
- OP_SLTI: so sánh giữa một thanh ghi và một giá trị nguyên được xác định trước (extra) và gán 1 vào thanh ghi rt nếu giá trị của thanh ghi nhỏ hơn giá trị nguyên này, ngược lại gán 0.
- OP_SLTIU: so sánh không dấu giữa một thanh ghi và một giá trị nguyên không dấu được xác định trước (extra) và gán 1 vào thanh ghi rt nếu giá trị của thanh ghi nhỏ hơn giá trị nguyên này, ngược lại gán 0.
- OP_SLTU: so sánh không dấu giữa hai thanh ghi (rs và rt) và gán 1 vào thanh ghi đích nếu rs nhỏ hơn rt, ngược lại gán 0.
- OP_SRA: thực hiện phép dịch phải số học (shift right arithmetic) trên giá trị của thanh ghi rt và lưu kết quả vào thanh ghi đích được chỉ định bởi instr->rd.
- OP_SRAV: thực hiện phép dịch phải số học (shift right arithmetic) trên giá trị của thanh ghi rt với một số lượng bit được xác định bởi giá trị của thanh ghi rs, sau đó lưu kết quả vào thanh ghi đích được chỉ định bởi instr->rd.

- OP_SRL: thực hiện phép dịch phải logic (shift right logical) trên giá trị của thanh ghi rt và lưu kết quả vào thanh ghi đích được chỉ định bởi instr->rd.
- OP_SRLV: thực hiện phép dịch phải logic (shift right logical) trên giá trị của thanh ghi rt với một số lượng bit được xác định bởi giá trị của thanh ghi rs, sau đó lưu kết quả vào thanh ghi đích được chỉ định bởi instr->rd.
- OP_SUB: thực hiện phép trừ giữa hai toán hạng (rs và rt) và gán kết quả vào thanh ghi đích được chỉ định bởi instr->rd. Nếu có overflow (tràn số) xảy ra, sẽ ném một ngoại lệ.
- OP_SUBU: thực hiện phép trừ không dấu giữa hai toán hạng (rs và rt) và gán kết quả vào thanh ghi đích được chỉ định bởi instr->rd.
- OP_SW: ghi một từ (4 byte) từ thanh ghi rt vào bộ nhớ tại địa chỉ được tính toán bằng cách thêm extra vào giá trị của thanh ghi rs.
- OP_SWL: ghi một từ (4 byte) từ thanh ghi rt vào bộ nhớ tại địa chỉ được tính toán bằng cách thêm extra vào giá trị của thanh ghi rs. Tuy nhiên, việc ghi được thực hiện một cách phức tạp hơn so với lệnh OP_SW. Giá trị từ bộ nhớ tại địa chỉ được tính toán được đọc và chỉ một phần của nó được thay đổi bởi giá trị của thanh ghi rt. Sau đó, giá trị được ghi trở lại bộ nhớ tại địa chỉ đã được đọc. Quá trình này giúp giữ nguyên các giá trị khác của từ đó.
- OP_SWR: tương tự như OP_SWL, nhưng nó thực hiện việc ghi vào từ cuối cùng (little-endian) của một từ 4 byte tại địa chỉ tính toán. Tương tự, nó thay đổi một phần của từ đó bằng giá trị của thanh ghi rt, sau đó ghi lại từ này vào bộ nhớ.
- OP_SYSCALL: kích hoạt một ngoại lệ syscall, thông báo cho hệ điều hành rằng một syscall đã được gọi.
- OP_XOR: thực hiện phép XOR logic giữa hai giá trị từ hai thanh ghi rs và rt, sau đó kết quả được gán vào thanh ghi đích rd.
- OP_XORI: thực hiện phép XOR logic giữa một giá trị từ thanh ghi rs và một hằng số nguyên, sau đó kết quả được gán vào thanh ghi rt.
- OP_RES và OP_UNIMP: Cả hai lệnh này đều ném một ngoại lệ để báo hiệu rằng lệnh đang được thực thi là không được triển khai hoặc không được hỗ trợ.
- Trong trường hợp mặc định (default), chúng ta gặp một ASSERT(FALSE). Trong ngữ cảnh này, ASSERT(FALSE) là một khẳng định rằng điều kiện luôn luôn sai. Điều này được sử dụng như một cơ chế bảo vệ để đảm bảo rằng chương trình không thể thực hiện đến dòng mã này vì không có lệnh hợp lệ nào trong tập lệnh của máy MIPS có thể đưa đến điều kiện mặc định. Nếu chương trình thực hiện đến trạng thái này, điều đó có nghĩa là đã có lỗi xảy ra trong việc giải mã và thực thi lệnh. Việc gặp phải điều kiện này là một tình huống không mong muốn và cần được giải quyết để xác định vấn đề trong mã hoặc dữ liệu đầu vào. Thông thường, một thông điệp lỗi cần được in ra để cung cấp thông tin chi tiết về lỗi và hỗ trợ sửa chữa. Trong môi trường thực tế, thay vì

sử dụng ASSERT(FALSE), chúng ta thường sẽ sử dụng các cơ chế xử lý lỗi như in ra thông báo lỗi hoặc ném một ngoại lệ để báo cáo lỗi và dừng chương trình một cách an toàn.

10. console.h + console.cc

Chứa class console, dùng để mô phỏng các thiết bị I/O sử dụng UNIX file. Thời gian để đọc và ghi 1 ký tự tối đa là 100 milli giây.

Các thuộc tính:

- readFileNo, writeFileNo: số thứ tự của file UNIX dùng để mô phỏng bàn phím và màn hình.
- incoming: nếu có dữ liệu, chứa char sẽ được đọc, nếu không, chứa ký tự EOF.
- putBusy: cờ báo hiệu chức năng putchar có đang được dùng hay không. Nếu có thì không thể khởi tạo putchar thêm.
- handleArg: chứa tham số sẽ truyền vào hàm gián đoạn.
- writeHandler, readHandler: hàm gián đoạn cho nhập và xuất.
 - writeHandler: được gọi khi hàm putchar I/O hoàn thành.
 - readHandler: được gọi khi có ký tự được truyền đến từ bàn phím.

Các chức năng:

a) Constructor

- Bắt đầu giả lập phần cứng nhập xuất của máy tính. Chấp nhận 5 tham số:
 - readFile: file UNIX dùng để giả lập bàn phím/thiết bị nhập, nếu trống dùng stdin, nhập từ console.
 - writeFile: file UNIX dùng để giả lập màn hình/thiết bị ghi,xuất, nếu trống dùng stdout, in ra console.
 - readAvail: hàm gián đoạn khi có ký tự được nhập từ bàn phím/đọc được ký tự từ file.
 - writeDone: hàm gián đoạn khi một ký tự được in ra thiết bị xuất/màn hình, để báo rằng có thể yêu cầu ký tự tiếp theo được xuất. Cách hoạt động:
 - Nếu readFile rỗng, thay thế bằng stdin, nếu không, mở chỉ đọc file nhập UNIX.
 - Nếu writeFile rỗng, thay thế bằng stdout, nếu không, mở chỉ viết cho file xuất UNIX.
 - Sau đó, thiết lập các biến giả lập hàm gián đoạn bất đồng bộ.

- Cuối cùng, thiết lập hàm gián đoạn CPU, lắng nghe sự kiện, hàm này cũng đánh dấu bắt đầu nhận các packet từ thiết bị nhập xuất, có 4 tham số:
 - handler: phương thức gián đoạn được dùng.
 - arg: tham số truyền vào phương thức gián đoạn
 - when: tham số xác định thời gian gián đoạn, hàm gián đoạn sẽ hoạt động tại thời điểm hàm lắng nghe được gọi + thời gian gián đoạn truyền vào.
 - type: mã phân cứng dùng để tạo sự kiện gián đoạn

b) Destructor

- Dùng để đóng các file giả lập UNIX. Cách hoạt động:
 - Nếu readFile không phải stdin, đóng readFile.
 - Nếu writeFile không phải stdout, đóng writeFile

c) CheckAvail

- Được gọi theo chu kỳ để kiểm tra nếu một ký tự được nhập vào từ bàn phím. Chỉ đọc các ký tự được nhập nếu nó được đẩy ra khỏi buffer bởi kernel. Gọi hàm gián đoạn của input mỗi khi một ký tự được thêm vào buffer.
- Cách hoạt động:
 - Khởi tạo char.
 - Lên thời gian cho lần kế tiếp lắng nghe nhập/nhận một packet.
 - Nếu kết thúc file hoặc khi char trong buffer không còn, kết thúc hàm.
 - Nếu không, đọc char theo số byte, trả về lỗi nếu có, tăng biến đếm số ký tự từ đã đọc lên 1.
 - Gọi hàm gián đoạn CPU cho read.

d) WriteDone

- Được gọi khi đến lúc gọi hàm gián đoạn để báo cho kernel biết việc xuất/in ký tự đã hoàn thành.
- Cách hoạt động:
 - Cờ putBusy được đặt thành False, cho phép gọi hàm putchar để xuất/in.
 - Tăng biến đếm số ký tự được in lên 1.
 - Gọi hàm gián đoạn CPU cho write.

e) GetChar

- Đọc một ký tự từ buffer nhập, và trả về ký tự đó hoặc trả về EOF nếu buffer rỗng.
- Cách hoạt động:

- Gán biến char bằng giá trị của incoming.
- Đặt lại giá trị của incoming.
- Trả về biến char.

f) PutChar

- Hàm cho phép xuất một ký tự ra thiết bị mô phỏng xuất, lên lịch trình hàm gián đoạn cho sau này.
- Cách hoạt động:
 - Nếu putChar đang có 1 tiến trình đang chạy, báo lỗi.
 - Nếu không, viết vào thiết bị mô phỏng một char với kích thước n bytes.
 - Bật putBusy lên True
 - Gọi hàm thiết lập gián đoạn cho write.

11. synchconss.h + synchcons.cc

Chứa class synchconcole, giả lập thiết bị đồng bộ nhập xuất

Các thuộc tính:

- cons: một con trỏ đến một object console, nơi ta sẽ chứa thiết bị nhập xuất bất đồng bộ.
- Các biến tĩnh của object Semaphore: luồng tiến trình sẽ khiến các yêu cầu nhập xuất chờ Semaphore cho tới khi nhập xuất hoàn thành.

Các chức năng:

a) SynchReadFunc, SynchWriteFunc

- Khởi động luồng yêu cầu nhập xuất.

b) Constructor không tham số

- Tạo một object console với stdin và stdout, sử dụng SynchReadFunc và SynchWriteFunc làm hàm gián đoạn.
- Gán cho các biến Semaphore tương ứng với nhập xuất khả dụng, nhập xuất theo các dòng.

c) Constructor có tham số

- Tương tự constructor không tham số nhưng thay vì dùng stdin và stdout thì sử dụng readFunc và writeFunc cho thiết bị mô phỏng nhập xuất tương ứng.

d) Destructor

- Xoá bỏ console, và các biến tĩnh của Semaphore.

e) Write

- Chức năng: Viết các byte theo số lượng byte cho trước, từ buffer cho trước vào thiết bị nhập xuất.
- Cách hoạt động:
 - Khởi tạo biến loop.
 - Từ biến Semaphore tĩnh WLineBlock có được khi tạo synchconsole, gọi chức năng P của Semaphore:
 - Hàm P chờ khi giá trị của Semaphore lớn hơn 0, rồi giảm giá trị đó dần.
 - Việc kiểm tra giá trị Semaphore và việc giảm giá trị đó phải được thực hiện không gián đoạn. Vì vậy, ta phải tắt hàm gián đoạn khi thực hiện các chức năng này.
 - Đầu tiên, ta tắt hàm gián đoạn.
 - Tiếp theo, ta kiểm tra nếu giá trị của Semaphore bằng 0 hay không. Nếu có, đưa luồng tiến trình hiện tại vào hàng chờ và cho luồng tiến trình ngủ.
 - Nếu không, đồng nghĩa Semaphore khả dụng, ta giảm giá trị của Semaphore.
 - Trả giá trị cũ cho hàm gián đoạn, đồng nghĩa với việc bật hàm gián đoạn.
 - Tạo một vòng lặp có độ dài bằng số byte truyền vào.
 - Nhập char từ buffer truyền vào, vào console bằng hàm putChar.
 - Gọi hàm Semaphore để chặn các yêu cầu nhập xuất cho một ký tự.
 - Từ biến Semaphore tĩnh WLineBlock có được khi tạo synchconsole, gọi chức năng V của Semaphore:
 - Tăng giá trị cho giá trị của Semaphore, đánh thức luồng tiến trình nếu cần thiết.
 - Đầu tiên, tắt hàm gián đoạn.
 - Lấy tiến trình đầu tiên ở trong hàng chờ. Nếu tiến trình này không trống thì đánh thức tiến trình này, tăng giá trị Semaphore.
 - Cuối cùng bật hàm gián đoạn.
 - Trả về số byte được đọc.

f) Read

- Chức năng: đọc số byte bằng số lượng byte cho trước từ buffer, vào thiết bị nhập xuất.
- Cách hoạt động:
 - Từ buffer được cho trước, khiến tất cả các byte từ đầu đến vị trí bằng offset số byte đã cho thành 0.

- Gọi hàm P cho readLineBlock
- Tạo vòng lặp đến khi vượt quá số byte đọc hoặc gặp EOL
 - Tạo vòng lặp do...while... khi mà byte được đọc là EOL
 - Chặn tiến trình cho phần đọc 1 ký tự
 - Đọc char từ console.
 - Nếu char đọc được là '\012' hoặc '\001' thì EOL và kết thúc vòng lặp.
 - Nếu không thì thêm char vào loop theo như các vị trí mà ta đã format trong buffer.
- Gọi hàm V cho readLineBlock.
- Trả về số byte đã đọc hoặc -1 nếu byte đọc được là CTRL-A

12. Test folder

Chứa các file C và object dùng để chạy và kiểm thử NachOS, mô phỏng các chương trình.

a) Halt

- Chức năng: dùng để kiểm thử chạy một chương trình trong NachOS có được không.
- Cách hoạt động: Gọi Syscall HALT để shutdown NachOS.

b) Shell

- Chức năng: Mô phỏng shell trong các hệ điều hành. Sử dụng để chạy các chương trình
- Cách hoạt động:
 - Gán address space ID cho chương trình shell.
 - Tạo 2 OpenFileID chứa input và output cho shell, sử dụng stdin và stdout.
 - Vòng lặp vĩnh viễn:
 - Viết -- ra shell, thể hiện bắt đầu nhận input.
 - Đọc từng ký tự được nhập vào shell cho đến khi gặp ký tự xuống dòng. Sau đó gán giá trị kết thúc chuỗi cho chuỗi đã đọc được.
 - Nếu user đã nhập chuỗi, thì tạo tiến trình tương ứng với chuỗi đã nhập và chuyển sang tiến trình đó.

c) Sort

- Chức năng: chương trình kiểm thử sắp xếp một chuỗi lớn bao gồm các số nguyên. Dùng để tạo áp lực/lấp đầy hệ thống bộ nhớ ảo(RAM ảo).
- Cách hoạt động:
 - Tạo mảng số nguyên có kích cỡ 1024.
 - Tạo mảng giảm dần 1024 chữ số.
 - Sắp xếp mảng theo thuật toán Selection Sort.

- Gọi hàm kết thúc với `exitCode` là số đầu tiên của mảng, nếu hoạt động đúng thì `exit code` là số 0.

d) Start

- Chức năng: hỗ trợ ngôn ngữ Assembly for các chương trình chạy trên Hệ điều hành NachOS.
- Cách hoạt động: Sử dụng hợp ngữ MIPS để lấy, nhập, thực hiện các phép toán trên các thanh ghi.

e) Matmult

- Chức năng: chương trình kiểm thử, nhân ma trận cỡ lớn. Dùng để tạo áp lực/lấp đầy hệ thống bộ nhớ ảo (RAM ảo).
- Cách hoạt động:
 - Tạo 3 mảng 2D có kích cỡ 20x20.
 - Lấp đầy các hàng của 2 matrix đầu với các giá trị, matrix thứ 3 sẽ bằng 0.
 - Thực hiện nhân 2 ma trận đầu, ma trận thứ 3 sẽ lưu kết quả.
 - Thoát chương trình và trả về `exitCode` là phần tử cuối cùng của ma trận thứ 3. Nếu hoạt động đúng, `exitCode` là 0.

IV. Cài đặt xử lý các Exception và các system call nhập xuất

1. IncreasePC

- Hàm được viết trong file “exception.cc” trong thư mục userprog.
- Bắt đầu với việc đọc giá trị hiện tại của thanh ghi bằng cách sử dụng hàm `ReadRegister` của machine. Giá trị này là địa chỉ của lệnh hiện tại trong máy được lưu vào biến “counter”. Lưu giá trị này vào thanh ghi trước đó (`PrevPCReg`) bằng cách dùng hàm `WriteRegister` của machine, truyền vào 2 biến `PrevPCReg` và “counter”.
- Đọc giá trị của thanh ghi tiếp theo (`PCNextReg`), giá trị này là địa chỉ của lệnh tiếp theo sẽ được thực thi trong máy.
- Ghi giá trị của PC hiện tại vào thanh ghi PC bằng hàm `WriteRegister`, điều này đảm bảo rằng PC trở đến lệnh tiếp theo sẽ được thực thi khi hàm này kết thúc.
- Cuối cùng, cập nhật thanh ghi tiếp theo của PC để trở đến lệnh kế tiếp (`PC+4`). Mỗi lệnh trong máy thường có độ dài 4 byte nên để thực hiện lệnh tiếp theo, giá trị của PC cần được tăng lên 4.

2. Xử lý các Exception không phải system call

Bên trong thư mục .code/machine, file “machine.h” có chứa danh sách các exception. Sau đó, qua file “exception.cc” trong thư mục .code/userprog, ta viết các case trong hàm ExceptionType, in ra lỗi tương ứng. Cuối mỗi case sử dụng halt() để kết thúc chương trình.

3. Chuẩn bị cài đặt các system call

a) Define các system call

Trong file “syscall.h”, các system call được define như sau:

- SC_ReadChar và SC_PrintChar là 13 và 14
- SC_ReadInt và SC_PrintInt là 15 và 16
- SC_ReadString và SC_PrintString là 17 và 18
- SC_ReadFloat và SC_PrintFloat là 19 và 20.

b) Khai báo biến toàn cục trong “.code/threads” 2 file “system.h” và “system.cc”

- Trong file “system.h”, khai báo thư viện “synchcons.h” và một biến con trỏ gSynchConsole để dùng cho việc read và write ở console
- Trong file “system.cc”, cũng khai báo con trỏ gSynchConsole từ SynchConsole, cài đặt tạo mới và giải phóng bộ nhớ.

c) Thay đổi trong .code/filesys/filesys.h

- Trong lớp FileSystem,
 - Khai báo “openfile” là một mảng con trỏ đến các đối tượng “OpenFile”, được sử dụng để theo dõi các file đang mở trong file system. Mỗi đối tượng “OpenFile” đại diện cho một file đang mở và cung cấp các phương thức để thực hiện thao tác đọc ghi, di chuyển trong file.
 - Khai báo biến “index” để theo dõi số lượng các file đang mở trong file system. Nó chỉ ra vị trí kế tiếp trong mảng “openfile” mà một file mới sẽ được thêm vào.
- Ở các hàm Open, tăng giá trị index mỗi khi kết thúc hàm để chỉ ra vị trí kế tiếp trong mảng “openfile”.
- Khai báo và cài đặt hàm open file với type chỉ định.

4. System call ReadInt + PrintInt

a) SC_ReadInt

Xử lý system call ReadInt được thực hiện qua các bước:

- Khai báo một biến maxBytes để xác định kích thước tối đa của buffer đọc từ console.
- Cấp phát bộ nhớ động để tạo buffer để đọc dữ liệu từ console. Kích thước của buffer được xác định bởi maxBytes.
- Gọi hàm Read từ đối tượng class gSynchConsole để đọc dữ liệu từ console vào buffer. bytesRead lưu trữ số lượng byte thực sự đã đọc.
- Tạo vòng lặp for, duyệt qua từng ký tự trong buffer để kiểm tra xem chúng có phải là một số nguyên hợp lệ không.
 - Nếu buffer rỗng hoặc chứa ký tự '-' ở vị trí đầu tiên, vòng lặp tiếp tục mà không làm gì cả (sử dụng continue).
 - Nếu ký tự tại vị trí i không nằm trong khoảng ['0', '9'] (tức không phải là một chữ số), thì in ra thông báo lỗi và ghi giá trị 0 vào thanh ghi r2.
- Nếu buffer chứa một chuỗi số nguyên hợp lệ, chuyển đổi nó thành một số nguyên bằng hàm atoi.
- Ghi số nguyên đã chuyển đổi vào thanh ghi r2.
- Tăng giá trị của thanh ghi PC để di chuyển tới lệnh tiếp theo.
- Giải phóng bộ nhớ đã cấp phát cho buffer bằng cách sử dụng delete[] buffer.

b) SC_PrintInt

Xử lý system call PrintInt được thực hiện qua các bước:

- Đọc giá trị số nguyên từ thanh ghi r4 vào biến number.
 - Kiểm tra xem số nguyên đọc được có bằng 0 không. Nếu có, chỉ đơn giản là ghi ký tự '0' ra console và tăng giá trị của thanh ghi PC để di chuyển tới lệnh tiếp theo.
- Đếm số chữ số của số nguyên bằng cách lặp qua từng chữ số của số đó và tăng biến digitCount cho đến khi số đó đã được đếm hết.
- Cấp phát bộ nhớ động cho buffer để chứa chuỗi kết quả. Kích thước của buffer được xác định bởi digitCount (số chữ số của số nguyên).
- Gán ký tự kết thúc chuỗi vào vị trí cuối cùng của buffer.
- Kiểm tra xem số nguyên có phải là số âm không. Nếu có, đặt cờ negative thành true và biến đổi số âm thành dạng dương để tiện xử lý sau này.
- Vào vòng lặp for để, chuyển đổi số nguyên thành chuỗi bằng cách lấy từng chữ số của số nguyên và chuyển đổi nó thành ký tự ASCII tương ứng, sau đó gán vào buffer từ cuối đến đầu.

- Nếu số nguyên ban đầu là số âm, thêm ký tự '-' vào đầu chuỗi.
- Ghi chuỗi đã được tạo ra vào console để hiển thị số nguyên. Kích thước của buffer là `digitCount + 1` để bao gồm cả ký tự kết thúc chuỗi.
- Tăng giá trị của thanh ghi PC để di chuyển tới lệnh tiếp theo.
- Giải phóng bộ nhớ đã cấp phát cho buffer bằng cách sử dụng `delete[] buffer`.

5. System call ReadFloat + Writefloat

a) SC_ReadFloat

- Xác định số lượng ký tự tối đa mà có thể đọc được từ bàn phím, trong trường hợp này, số lượng ký tự tối đa được đặt là 255.
- Tạo một buffer có kích thước 255 để lưu trữ dữ liệu đọc từ console, sau đó sử dụng phương thức "Read" của đối tượng "gSynchConsole" để đọc dữ liệu từ bàn phím và lưu trữ vào buffer.
- Khởi tạo một biến check bằng true. Nếu số ký tự '.' trong buffer nhiều hơn 1 hoặc các ký tự không phải là không phải là số thì biến check bằng false.
- Nếu check vẫn bằng true thì chuyển từ buffer sang biến float "f" bằng hàm `sscanf` của c. Sau đó dùng kỹ thuật type punning để chuyển float "f" thành int "f_int" nhờ con trỏ mà không làm mất dữ liệu trên bộ nhớ. Sau đó lưu "f_int" vào register số 2 để syscall trả về.
- Nếu check bằng false thì sẽ in ra màn hình console dữ liệu vừa nhập không phải là số thực và lưu 0 vào register số 2 để syscall trả về.
- Cuối cùng tăng giá trị program counter.

b) SC_WriteFloat

- Đọc số thực từ thanh ghi 4 của máy vào biến "f_int".
- Sau khi đọc ký tự từ thanh ghi, ta lại dùng kỹ thuật type punning để chuyển "f_int" từ kiểu int sang biến "f" kiểu float.
- Sử dụng hàm `sprintf` của c để lưu của ký tự của số thực cần in vào mảng char tên "buffer".
- Sau đó dùng phương thức "Write" của đối tượng "gSynchConsole" để ghi ký tự đó ra màn hình. Ký tự được ghi bằng cách truyền địa chỉ của "c" và số lượng ký tự cần ghi, trong trường hợp này là 1.
- Cuối cùng, tăng giá trị program counter.

6. System call ReadChar + WriteChar

a) SC_ReadChar

- Xác định số lượng kí tự tối đa mà có thể đọc được từ bàn phím, trong trường hợp này, số lượng kí tự tối đa được đặt là 255.
- Tạo một buffer có kích thước 255 để lưu trữ dữ liệu đọc từ console, sau đó sử dụng phương thức “Read” của đối tượng “gSynchConsole” để đọc dữ liệu từ bàn phím và lưu trữ vào buffer.
- Nếu số lượng ký tự đọc được lớn hơn 1, in ra thông báo lỗi cho người dùng và ghi thông điệp lỗi vào debug. Sau đó, đặt giá trị trả về của thanh ghi 2 là 0 để chỉ ra rằng có lỗi xảy ra. Nếu số lượng ký tự đọc được là 0, in ra thông báo lỗi cho người dùng và ghi lỗi vào debug. Đặt giá trị trả về của thanh ghi 0 để chỉ ra có lỗi xảy ra. Nếu số lượng kí tự đọc được là 1, lấy kí tự đầu tiên từ buffer và ghi vào thanh ghi 2 của máy.
- Sau khi đọc xong, giải phóng bộ nhớ của buffer để tránh leaking và tăng program counter.

b) SC_PrintChar

- Đọc kí tự từ thanh ghi 4 của máy vào biến “c”.
- Sau khi đọc kí tự từ thanh ghi, dùng phương thức “Write” của đối tượng “gSynchConsole” để ghi kí tự đó ra màn hình. Kí tự được ghi bằng cách truyền địa chỉ của “c” và số lượng kí tự cần ghi, trong trường hợp này là 1.
- Cuối cùng, tăng giá trị program counter.

7. System call Create

- CreateFile sẽ sử dụng Nachos FileSystem Object để tạo một file rỗng. System call CreateFile trả về 0 nếu thành công và -1 nếu có lỗi.
- Input: tên file
- Output: 0: thành công hoặc -1: lỗi
- Các bước thực hiện:
 - Đọc địa chỉ ảo của tên file từ thanh ghi 4: đầu tiên, hàm đọc địa chỉ ảo của tên file từ thanh ghi 4 của máy. Địa chỉ này là địa chỉ ảo của vùng nhớ mà tên file được lưu trữ.
 - Chuyển đổi địa chỉ ảo sang địa chỉ hệ thống và lấy tên file: Sử dụng hàm “User2System” để chuyển đổi và lấy tên file.
 - Kiểm tra tính hợp lệ của tên file: kiểm tra tên file có rỗng không bằng cách sử dụng hàm “strlen”. Nếu tên file rỗng, hàm in ra thông báo lỗi, trả về giá trị thanh ghi 2 là -1 và tăng giá trị counter để chuyển đến lệnh tiếp theo.

- Kiểm tra khả năng tạo file trong hệ thống: kiểm tra xem hệ thống có đủ bộ nhớ để tạo file không. Nếu không đủ bộ nhớ, hàm in ra thông báo lỗi, đặt giá trị thanh ghi 2 là -1, giải phóng bộ nhớ đã cấp phát và tăng giá trị counter để chuyển đến lệnh tiếp theo.
- Hàm gọi phương thức “Create” của fileSystem để tạo file với tên đã được nhận từ bước trước. Nếu việc tạo file thành công, hàm đặt giá trị trả về cho thanh ghi 2 là 0, giải phóng bộ nhớ đã cấp phát và tăng giá trị counter để chuyển đến lệnh tiếp theo.
- Cuối cùng, hàm giải phóng bộ nhớ đã cấp phát cho tên file và tăng giá trị counter.

8. System call Open + Close

Dùng để mở/đóng 1 file

a) SC_Open

- Input: tên file trên user space và type (0: read và write, 1: read only).
- Output: Id của file nếu thành công và -1 nếu thất bại.
- Các bước thực hiện:
 - Đọc địa chỉ ảo của tên file từ thanh ghi số 4 của máy. Địa chỉ này là địa chỉ ảo của vùng nhớ mà tên file được lưu trữ.
 - Sau đó, sẽ đọc type từ thanh ghi số 5 của máy.
 - Kiểm tra xem số lượng file đã mở đạt tối đa hay chưa (tối đa là 10). Nếu đã tối đa, hàm đặt giá trị trả về thanh ghi 2 là -1, điều này có nghĩa là mở file thất bại.
 - Dùng hàm “User2System” để chuyển đổi địa chỉ ảo sang địa chỉ hệ thống và lấy tên file từ địa chỉ đó.
 - Kiểm tra xem tên file có phải là “stdin” hoặc “stdout” không. Nếu là 1 trong 2 thì in ra thông báo và đặt giá trị trả về thanh ghi 2 là 1 (“stdin”) hoặc 0 (“stdout”) để thông báo việc mở file thành công.
 - Gọi phương thức “Open” của fileSystem để mở file với tên file đã được nhận từ các bước trên. Nếu mở file thành công, hàm in ra thông báo và đặt giá trị trả về thanh ghi 2 là chỉ số của file đã mở trong mảng “openfile”. Nếu không, hàm in thông báo lỗi và đặt giá trị thanh ghi 2 là -1.
 - Cuối cùng, giải phóng bộ nhớ đã cấp phát cho tên file và tăng giá trị của counter.

b) SC_close

- Input: ID của file

- Output: NULL
- Các bước thực hiện:
 - Đọc giá trị file descriptor từ thanh ghi 4 của máy. File descriptor này là chỉ số của file trong mảng “openfile” mà muốn đóng.
 - Kiểm tra xem file descriptor đọc được có hợp lệ hay không bằng cách so sánh với số lượng file đã mở trong hệ thống. Nếu file descriptor lớn hơn số lượng file đã mở, điều này chỉ ra rằng file không tồn tại hoặc không hợp lệ và in ra thông báo lỗi. Sau đó, đặt giá trị trả về cho thanh ghi 2 là -1.
 - Nếu file descriptor hợp lệ, tiến hành đóng file bằng cách gán “NULL” cho vị trí tương ứng trong mảng “openfile”. Sau đó, giải phóng bộ nhớ đã cấp phát cho file.
 - Cuối cùng, đặt giá trị trả về của thanh ghi 2 là 0, tăng program counter.

9. System call Read + Write

Dùng để đọc, viết file

a) SC_Read

- Input: buffer, số byte cần đọc, ID của file.
- Output: -1 nếu thất bại, số byte đã đọc nếu thành công.
- Các bước thực hiện:
 - Đọc giá trị địa chỉ ảo của bộ đệm từ thanh ghi 4 của máy, đọc số lượng ký tự cần đọc từ thanh ghi 5 của máy, đây là số lượng tối đa các ký tự sẽ được đọc và sao chép vào bộ đệm.
 - Đọc giá trị file descriptor từ thanh ghi 6 của máy. File descriptor là chỉ số của file trong mảng “openfile” muốn đọc.
 - Kiểm tra xem file descriptor có hợp lệ hay không bằng cách so sánh với số lượng file đã mở trong hệ thống và kiểm tra xem có phải là “stdout” hay không. Nếu file descriptor không hợp lệ hoặc đang cố gắng đọc từ stdout, hàm sẽ in ra thông báo lỗi, trả giá trị thanh ghi số 2 là -1.
 - Kiểm tra xem file có mở được hay không bằng cách kiểm tra vị trí tương ứng trong mảng “openfile” có trỏ đến một đối tượng file hay không. Nếu không, trả giá trị thanh ghi 2 là -1 và kết thúc hàm.
 - Cấp phát một bộ nhớ đệm trong kernel với kích thước là số lượng ký tự cần đọc.
 - Nếu file đang đọc là stdin, sử dụng hàm “Read” của “gSynchConsole” để đọc dữ liệu từ bàn phím và sao chép vào bộ đệm. Nếu là file khác, hàm sử dụng phương thức “Read” của file để đọc dữ liệu và sao chép vào bộ đệm.

- Sau khi đọc dữ liệu, hàm sử dụng “System2User” để sao chép dữ liệu từ bộ đệm trong không gian kernel sang bộ đệm trong không gian người dùng.
- Nếu việc đọc dữ liệu thành công, đặt giá trị trả về thanh ghi 2 là số lượng ký tự đã đọc và tăng giá trị này thêm 1 để đánh dấu vị trí kết thúc chuỗi. Nếu không, đặt giá trị của thanh ghi 2 là -1 chỉ ra đọc dữ liệu thất bại. Sau đó, tăng program counter.

b) SC_Write

- Input: buffer, số lượng ký tự cần ghi, ID của file.
- Output: -1 nếu thất bại, số byte đã đọc nếu thành công.
- Các bước thực hiện:
 - Đọc giá trị địa chỉ ảo của bộ đệm từ thanh ghi 4 của máy, đọc số lượng ký tự cần đọc từ thanh ghi 5 của máy, đây là số lượng tối đa các ký tự sẽ được đọc và sao chép vào bộ đệm.
 - Đọc giá trị file descriptor từ thanh ghi 6 của máy. File descriptor là chỉ số của file trong mảng “openfile” muốn đọc.
 - Kiểm tra xem file descriptor có hợp lệ hay không bằng cách so sánh với số lượng file đã mở trong hệ thống và kiểm tra xem có phải là “stdin” hay không. Nếu file descriptor không hợp lệ hoặc đang cố gắng ghi vào stdin, hàm sẽ in ra thông báo lỗi, trả giá trị thanh ghi số 2 là -1.
 - Tiếp theo, kiểm tra xem file có mở được hay không bằng cách kiểm tra vị trí tương ứng trong mảng “openfile” có trỏ đến một đối tượng file hay không. Nếu không, trả giá trị thanh ghi 2 là -1 và kết thúc hàm.
 - Kiểm tra xem file có phải là file chỉ đọc không bằng cách kiểm tra thuộc tính “type” trong đối tượng file. Nếu file là file chỉ đọc, in ra thông báo lỗi, đặt giá trị trả về thanh ghi 2 là -1, tăng program counter, kết thúc hàm.
 - Nếu file đang được ghi là stdout, sử dụng hàm “Write” của “gSynchConsole” để ghi dữ liệu vào stdout. Nếu là file khác thì dùng phương thức “Write” của file để ghi dữ liệu vào file.
 - Nếu ghi dữ liệu thành công, sao chép dữ liệu từ bộ đệm trong không gian kernel sang bộ đệm trong không gian người dùng.
 - Nếu việc đọc dữ liệu thành công, đặt giá trị trả về thanh ghi 2 là số lượng ký tự đã ghi, tăng program counter. Nếu không thì không thay đổi giá trị thanh ghi 2 và kết thúc hàm.

10. System call Seek

Dùng để di chuyển con trỏ đọc file đến vị trí nào đó trong file

- Input: vị trí cần chỉ tới, ID của file.
- Output: -1 nếu thất bại, vị trí thực sự trong file nếu thành công.
- Các bước thực hiện:
 - Đọc giá trị vị trí cần di chuyển đến từ thanh ghi 4, đại diện cho vị trí mới mà con trỏ sẽ được di chuyển tới.
 - Đọc giá trị file descriptor từ thanh ghi 55 của máy. File descriptor này là chỉ số của file trong mảng “openfile”.
 - Kiểm tra xem file descriptor có hợp lệ không bằng cách so sánh với số lượng file đã mở trong hệ thống và kiểm tra xem vị trí tương ứng trong mảng “openfile” có trỏ đến một đối tượng file không. Nếu không hợp lệ, đặt giá trị trả về của thanh ghi 2 là -1 và kết thúc hàm.
 - Lấy độ dài của tệp mà con trỏ muốn di chuyển đến. Độ dài này được sử dụng để kiểm tra tính hợp lệ của vị trí mới được chỉ định.
 - Kiểm tra xem vị trí mới được chỉ định có hợp lệ không, tức là không lớn hơn độ dài của tệp. Nếu không hợp lệ, đặt giá trị trả về của thanh ghi 2 là -1 và kết thúc hàm.
 - Nếu vị trí mới là hợp lệ, hàm này sẽ di chuyển con trỏ tệp đến vị trí mới bằng cách sử dụng phương thức “Seek” của fileSystem.
 - Ghi vị trí mới của con trỏ tệp vào thanh ghi 2, tăng program counter.

V. Một số chương trình minh họa

1. Help

Chương trình help có chức năng in ra màn hình các dòng giới thiệu thông tin cơ bản về nhóm. Dưới đây là phân tích mã nguồn của chương trình help.c trong thư mục .code/test:

- Khai báo thư viện “syscall.h” và hằng số MAX_BUFFER_LENGTH bằng 1000
- Vào hàm main khởi động chương trình, mở file “mota.txt” với quyền đọc (1)
 - Nếu không mở được file, chương trình in ra thông báo lỗi.
- Chương trình đọc nội dung của file nếu mở file thành công.
 - Nếu không đọc được nội dung của file, chương trình in ra thông báo lỗi.
 - Nếu đọc thành công, chương trình in nội dung của file ra console.
- Chạy lệnh Halt để dừng chương trình sau khi thực hiện xong tất cả các công việc.

2. ASCII

Chương trình ascii dùng để in ra màn hình bảng mã ascii. Cách hoạt động như sau:

- Sử dụng syscall Open để mở file "ascii.txt" ở chế độ viết.
- Sau đó, sử dụng syscall Write để viết các chuỗi: "ASCII code: ", số thứ tự của mã ASCII được chuyển từ số nguyên sang chuỗi, " Character: ", và ký tự được chuyển từ kiểu char sang char*, cuối cùng là ký tự xuống dòng. Điều này sẽ viết tất cả các ký tự ASCII vào file ascii.txt.
- Làm lại tương tự những việc trên nhưng thay vì viết vào file thì ta in ra console. Cuối cùng, thông báo thành công và gọi hàm Halt() dừng NachOS.

3. Quick Sort

Chương trình được cài đặt trong file quicksort.c trong thư mục .code/test để sắp xếp một mảng số nguyên bằng thuật toán quick sort. Cách hoạt động như sau:

- Chương trình yêu cầu người dùng nhập số lượng phần tử của mảng.
- Nhập các phần tử của mảng tuần tự.
- Sau khi nhập mảng thì gọi hàm “quickSort” để sắp xếp các phần tử trong mảng.
- Một file tên “quicksort.txt” được mở để ghi kết quả vào.
- Nếu không mở được thì thông báo lỗi và kết thúc.
- Các phần tử trong mảng được chuyển thành chuỗi ký tự bằng hàm “intToString” và sau đó ghi vào file “quicksort.txt”, cách nhau bởi dấu cách.
- Sau khi ghi xong, file “quicksort.txt” được đóng.
- Chương trình in ra màn hình một thông báo cho người dùng biết rằng kết quả đã được ghi vào file “quicksort.txt”.
- Cuối cùng, kết thúc chương trình bằng Halt().

4. Merge Sort

Chương trình được cài đặt trong file mergesort.c trong thư mục .code/test để sắp xếp một mảng số nguyên bằng thuật toán merge sort. Cách hoạt động như sau:

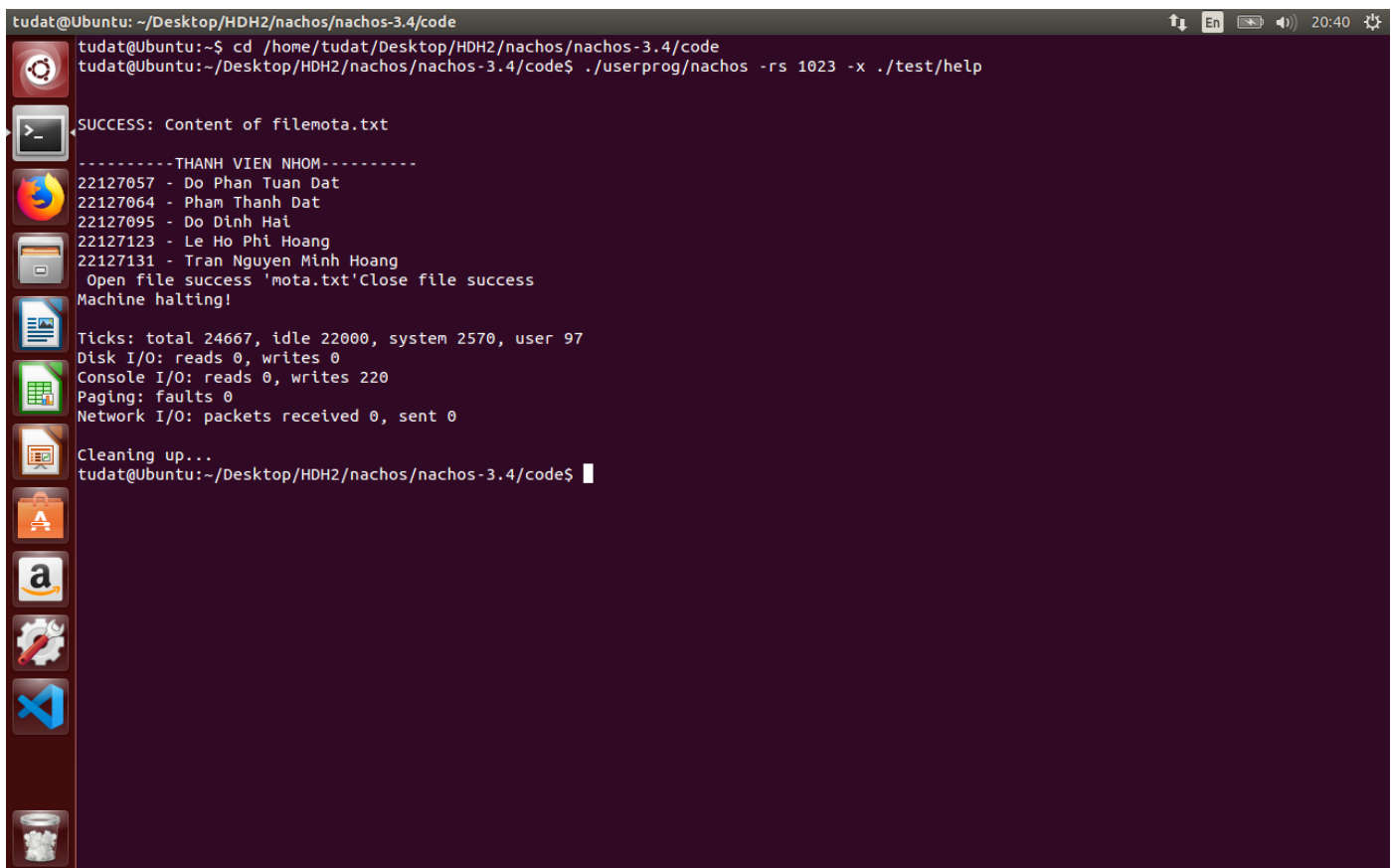
- Chương trình yêu cầu người dùng nhập số lượng phần tử của mảng.
- Nhập các phần tử của mảng tuần tự.
- Sau khi nhập mảng thì gọi hàm “mergeSort” để sắp xếp các phần tử trong mảng.
- Một file tên “mergesort.txt” được mở để ghi kết quả vào.
- Nếu không mở được thì thông báo lỗi và kết thúc.
- Các phần tử trong mảng được chuyển thành chuỗi ký tự bằng hàm “intToString” và sau đó ghi vào file “mergesort.txt”, cách nhau bởi dấu cách.

- Sau khi ghi xong, file “mergesort.txt” được đóng.
- Chương trình in ra màn hình một thông báo cho người dùng biết rằng kết quả đã được ghi vào file “mergesort.txt”.
- Cuối cùng, kết thúc chương trình bằng Halt().

VI. Demo sử dụng các chương trình minh họa

1. Help

- cd vào folder ./nuchos/nuchos-3.4/code
- Chạy chương trình help bằng lệnh: ./userprog/nuchos -rs 1023 -x ./test/help
- Trên màn hình ngay lập tức sẽ in ra thông tin nhóm như sau:



```
tudat@Ubuntu: ~/Desktop/HDH2/nuchos/nuchos-3.4/code
tudat@Ubuntu:~/Desktop/HDH2/nuchos/nuchos-3.4/code$ ./userprog/nuchos -rs 1023 -x ./test/help
SUCCESS: Content of filemota.txt
-----THANH VIEN NHOM-----
22127057 - Do Phan Tuan Dat
22127064 - Pham Thanh Dat
22127095 - Do Dinh Hai
22127123 - Le Ho Phi Hoang
22127131 - Tran Nguyen Minh Hoang
Open file success 'mota.txt'Close file success
Machine halting!
Ticks: total 24667, idle 22000, system 2570, user 97
Disk I/O: reads 0, writes 0
Console I/O: reads 0, writes 220
Paging: faults 0
Network I/O: packets received 0, sent 0
Cleaning up...
tudat@Ubuntu:~/Desktop/HDH2/nuchos/nuchos-3.4/code$
```

2. ASCII

- cd vào folder ./nuchos/nuchos-3.4/code
- Chạy chương trình ascii bằng lệnh: ./userprog/nuchos -rs 1023 -x ./test/ascii
- Trên màn hình ngay lập tức sẽ in ra bảng ascii. Đồng thời file ascii.txt với nội dung như sau sẽ được tạo ra

- Để file `ascii.txt` không bị lỗi, cần chạy lệnh sau trước khi mở file: `getis -encoding UTF-8 ascii.txt`

```

Terminal
ASCII code: 36 Character:$
ASCII code: 37 Character:%
ASCII code: 38 Character:&
ASCII code: 39 Character:'
ASCII code: 40 Character:(
ASCII code: 41 Character:)
ASCII code: 42 Character:*
ASCII code: 43 Character:+
ASCII code: 44 Character:,
ASCII code: 45 Character;-
ASCII code: 46 Character:.
ASCII code: 47 Character:/
ASCII code: 48 Character:0
ASCII code: 49 Character:1
ASCII code: 50 Character:2
ASCII code: 51 Character:3
ASCII code: 52 Character:4
ASCII code: 53 Character:5
ASCII code: 54 Character:6
ASCII code: 55 Character:7
ASCII code: 56 Character:8
ASCII code: 57 Character:9
ASCII code: 58 Character::
ASCII code: 59 Character:;
ASCII code: 60 Character:<
ASCII code: 61 Character:=
ASCII code: 62 Character:>
ASCII code: 63 Character:?
ASCII code: 64 Character:@
ASCII code: 65 Character:A
ASCII code: 66 Character:B
ASCII code: 67 Character:C
ASCII code: 68 Character:D
ASCII code: 69 Character:E
ASCII code: 70 Character:F
ASCII code: 71 Character:G
ASCII code: 72 Character:H
ASCII code: 73 Character:I
ASCII code: 74 Character:J
ASCII code: 75 Character:K
ASCII code: 76 Character:L
ASCII code: 77 Character:M
ASCII code: 78 Character:N

seapeas@ubuntu: ~/Desktop/nachos-hcmus-2024/nachos/nachos-3.4/code
ASCII code: 41 Character:)
ASCII code: 42 Character:*
ASCII code: 43 Character:+
ASCII code: 44 Character:,
ASCII code: 45 Character;-
ASCII code: 46 Character:.
ASCII code: 47 Character:/
ASCII code: 48 Character:0
ASCII code: 49 Character:1
ASCII code: 50 Character:2
ASCII code: 51 Character:3
ASCII code: 52 Character:4
ASCII code: 53 Character:5
ASCII code: 54 Character:6
ASCII code: 55 Character:7
ASCII code: 56 Character:8
ASCII code: 57 Character:9
ASCII code: 58 Character::
ASCII code: 59 Character:;
ASCII code: 60 Character:<
ASCII code: 61 Character:=
ASCII code: 62 Character:>
ASCII code: 63 Character:?
ASCII code: 64 Character:@
  
```

3. Quick Sort

- `cd` đến folder `./nachos/nachos-3.4/code`
- Chạy chương trình quicksort bằng lệnh: `./userprog/nachos -rs 1023 -x ./test/quicksort`
- Hướng dẫn sử dụng chương trình:
 - Nhập số lượng phần tử trong mảng (< 100)
 - Nhập tuần tự từng phần tử (sau mỗi phần tử, enter xuống hàng)
 - Khi chạy thành công sẽ tạo ra những dòng bên dưới

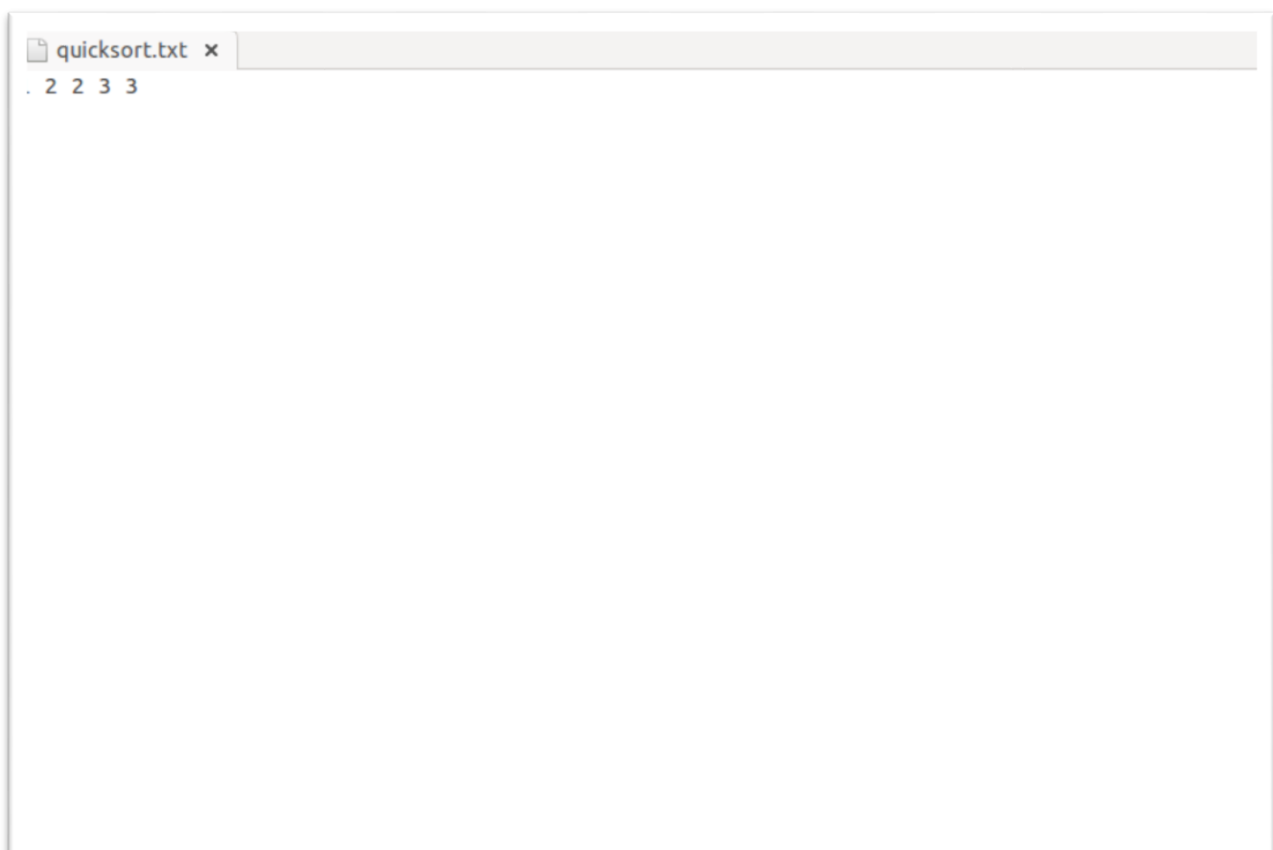
```
dat@ubuntu: ~/Desktop/nachos/nachos-3.4/code
make[1]: Leaving directory `/home/dat/Desktop/nachos/nachos-3.4/code/test'
dat@ubuntu:~/Desktop/nachos/nachos-3.4/code$ ./userprog/nachos -rs 1023 -x ./test/quickSort
Enter the length of array of integer:
5
Enter all element of array:
2
3
1
2
3

Open file success 'quickSort.txt'Close file success
The result is in file quickSort.txt
Machine halting!

Ticks: total 608997791, idle 608994297, system 1800, user 1694
Disk I/O: reads 0, writes 0
Console I/O: reads 12, writes 107
Paging: faults 0
Network I/O: packets received 0, sent 0

Cleaning up...
Assertion failed: line 254, file "../machine/sysdep.cc"
Aborted (core dumped)
dat@ubuntu:~/Desktop/nachos/nachos-3.4/code$
```

- Kết quả sẽ được lưu vào file quicksort.txt ở folder .nachos/nachos-3.4/code.



- Testcase khi xuất hiện số âm:

```
dat@ubuntu: ~/Desktop/nachos/nachos-3.4/code
t/quicksort
Enter the length of array of integer:
5
Enter all element of array:
-9
101
-1
2
-3

Open file success 'quicksort.txt'Close file success
The result is in file quicksort.txt
Machine halting!

Ticks: total 1346949279, idle 1346945140, system 1920, user 2219
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

Cleaning up...
Assertion failed: line 254, file "../machine/sysdep.cc"
Aborted (core dumped)
dat@ubuntu:~/Desktop/nachos/nachos-3.4/code$
```

```
quicksort.txt x
-9 -3 -1 2 101 |

Plain Text ▾ Tab Width: 8 ▾ Ln 1, Col 16 INS
```

- Testcase trường hợp nhập số lượng phần tử là 0:

```
dat@ubuntu: ~/Desktop/nachos/nachos-3.4/code
Cleaning up...
Assertion failed: line 254, file "../machine/sysdep.cc"
Files (core dumped)
dat@ubuntu:~/Desktop/nachos/nachos-3.4/code$ ./userprog/nachos -rs 1023 -x ./test/quickstort
Enter the length of array of integer:
0
Enter all element of array:

Open file success 'quickstort.txt'Close file success
The result is in file quickstort.txt
Machine halting!

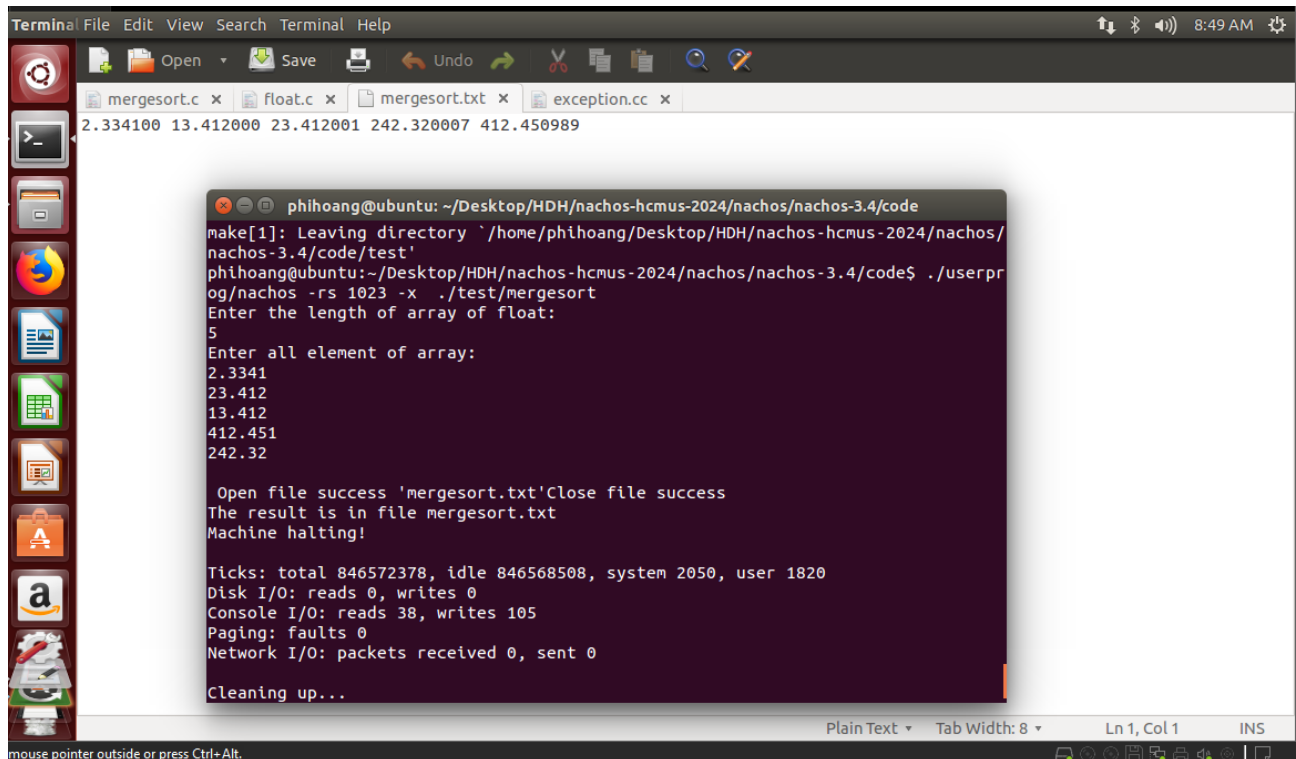
Ticks: total 171561709, idle 171560190, system 1400, user 119
Disk I/O: reads 0, writes 0
Console I/O: reads 2, writes 107
Paging: faults 0
Network I/O: packets received 0, sent 0

Cleaning up...
Assertion failed: line 254, file "../machine/sysdep.cc"
Aborted (core dumped)
dat@ubuntu:~/Desktop/nachos/nachos-3.4/code$
```

```
quickstort.txt x
Plain Text ▾ Tab Width: 8 ▾ Ln 1, Col 1 INC
```


4. Merge Sort

- cd đến folder ./nuchos/nuchos-3.4/code
- Chạy chương trình quicksort bằng lệnh: ./userprog/nuchos -rs 1023 -x ./test/mergesort
- Hướng dẫn sử dụng chương trình:
 - Nhập số lượng phần tử trong mảng (< 100)
 - Nhập tuần tự từng phần tử (sau mỗi phần tử, enter xuống hàng)
 - Khi chạy thành công sẽ tạo ra những dòng bên dưới trên console, đồng thời sẽ tạo ra 1 file mergesort.txt chứa kết quả



```
Terminal File Edit View Search Terminal Help
mergesort.c x float.c x mergesort.txt x exception.cc x
2.334100 13.412000 23.412001 242.320007 412.450989

phihoang@ubuntu: ~/Desktop/HDH/nuchos-hcmus-2024/nuchos/nuchos-3.4/code
make[1]: Leaving directory `/home/phihoang/Desktop/HDH/nuchos-hcmus-2024/nuchos/nuchos-3.4/code/test'
phihoang@ubuntu:~/Desktop/HDH/nuchos-hcmus-2024/nuchos/nuchos-3.4/code$ ./userprog/nuchos -rs 1023 -x ./test/mergesort
Enter the length of array of float:
5
Enter all element of array:
2.3341
13.412
23.412
412.451
242.32

Open file success 'mergesort.txt'Close file success
The result is in file mergesort.txt
Machine halting!

Ticks: total 846572378, idle 846568508, system 2050, user 1820
Disk I/O: reads 0, writes 0
Console I/O: reads 38, writes 105
Paging: faults 0
Network I/O: packets received 0, sent 0
Cleaning up...
```

VII. Đóng góp

Thành viên	MSSV	Công việc	Tỉ lệ (%) hoàn thành	Tổng đóng góp cho nhóm
Trần Nguyễn Minh Hoàng	22127131	<ul style="list-style-type: none">- Viết SC_Open và SC_Close và xử lý các exception- Viết báo cáo phân tích các phần tương ứng- Tổng hợp và thiết kế báo cáo	100%	20%
Lê Hồ Phi Hoàng	22127123	<ul style="list-style-type: none">- Tìm hiểu source code phần progtest.h, syscall.h và bitmap.*	100%	20%

		<ul style="list-style-type: none"> - Viết SC_ReadFloat và SC_PrintFloat và chương trình merge sort - Viết báo cáo phân tích và demo các phần tương ứng 		
Đỗ Đình Hải	22127095	<ul style="list-style-type: none"> - Tìm hiểu source code phần console.*, synchconsole.* và ../test/* - Viết SC_ReadString, và SC_PrintString và chương trình ascii - Viết báo cáo phân tích và demo các phần tương ứng 	100%	20%
Phạm Thành Đạt	22127064	<ul style="list-style-type: none"> - Tìm hiểu source code phần openfile.h, translate.* và machine.* - Viết SC_ReadChar, SC_PrintChar, SC_CreateFile, SC_Read, SC_Write và chương trình quick sort - Viết báo cáo phân tích và demo các phần tương ứng 	100%	20%
Đỗ Phan Tuấn Đạt	22127057	<ul style="list-style-type: none"> - Tìm hiểu source code phần machine.* và mipssim.cc - Viết SC_PrintInt và SC_ReadInt và chương trình help - Viết báo cáo phân tích và demo các phần tương ứng 	100%	20%

VIII. Tài liệu tham khảo

- Slide tài liệu Hệ điều hành của thầy Lê Viết Long, ĐHKHTN-ĐHQG HCM
- Source code nachos 3.4 cung cấp bởi thầy Lê Viết Long
- https://www.youtube.com/playlist?list=PLRgTVtca98hUgCN2_2vzsAAXPiTFbvHpO
- https://www.youtube.com/playlist?list=PLtK_L8azvdPa8QfM-4CW3vTBFMwZx1ucv
- <https://users.cs.duke.edu/~chase/nachos-guide/guide/nachos.htm>