**TRƯỜNG ĐẠI HỌC KHOA HỌC TỰ NHIÊN**

**KHOA CÔNG NGHỆ THÔNG TIN**

**BỘ MÔN: HỆ ĐIỀU HÀNH**



BÁO CÁO ĐỒ ÁN

FILE SYSTEM

**LỚP: 17CLC3**

**NHÓM: TÊN NHÓM**

**HỌ VÀ TÊN:**

* **LÊ QUỐC VIỆT-1753128**
* **NGUYỄN DUY TÂN-1753099**
* **VÕ KHÁNH VY-1753132**

**GVHD:**

* **THS.LÊ QUỐC HÒA**
* **CHUNG THÙY LINH**

1. **THÔNG TIN THÀNH VIÊN**

|  |  |  |  |
| --- | --- | --- | --- |
| **MSSV** | **Họ tên** | **Email** | **Đóng góp** |
| 1753099 | Nguyễn Duy Tân | 1753099@student.hcmus.edu.vn | 100% |
| 1753128 | Lê Quốc Việt | 1753128@student.hcmus.edu.vn | 100% |
| 1753132 | Võ Khánh Vy | 1753132@student.hcmus.edu.vn | 100% |

1. **TÌM HIỂU VỀ CHƯƠNG TRÌNH NACHOS**

**NachOS** (Not Another Completely Heuristic System) là một phần mềm giả lập hệ điều hành với kiến trúc MIPS. Các bộ phận của một hệ điều hành đều được giả lập và được viết bằng các chương trình C.

Lệnh **“./userprog/nachos-rs 1023-x ./test/halt”**: “./userprog/nachos” là đường dẫn đến NachOS để mở nó lên, “-rs” và “-x” là các tham số, “./test/halt” là đường dẫn đến file halt.noff đã được biên dịch xong. Như vậy cả câu lệnh là để khởi động máy ảo MIPS và hệ điều hành NachOS lên để nạp và thực hiên chương trình halt.

**Các tập tin:**

* **protest.cc**: kiểm tra các thủ tục chạy chương trình người dùng.
* **syscall.h**:các thủ tục mà người dùng có thể gọi ở kernel.
* **exception.cc**:xử lý các system call và các exception ở mức user.
* **bitmap.\***: các hàm xử lý cho lớp bitmap.
* **filesys.h**: lưu các ô nhớ vật lý.
* **openfile.h** : định nghĩa các hàm trong hệ thống file.
* **translate.\***:
* **machine.\***: mô phỏng các thành phần của máy tính khi thực thi chương trình người dùng như bộ nhớ, thanh ghi,..
* **mipssim.cc**: mô phỏng tập lệnh của MIPS R2/3000 processor.
* **console.\***: mô phỏng thiết bị đầu cuối sử dụng UNIX files.
* **synchconsole.\***: các hàm quản lý việc nhập xuất I/O và chạy trong NachOS.
* **../test/\***: chứa các chương trình C sẽ được biên dịch theo MIPS và chạy trong NachOS.

1. **TÌM HIỂU VỀ KIẾN TRÚC HỆ ĐIỀU HÀNH**

* **User MODE:** vùng nhớ của những chương trình ứng dụng chạy trên NachOS.
* **Kernel MODE:** vùng nhớ của hệ điều hành NachOS.
* **Kernel space** và **System space** là các vùng nhớ trong hệ điều hành.

1. **CÀI ĐẶT TỔNG QUAN**
2. **Cài đặt trước**

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

1. **Các bước cài đặt system call**

* Bước 1: ./code/userprog/syscall.h

#define SC\_Create 4 // define syscall để dùng trong switch-case

int Create(char\* name); // Khai báo prototype của hàm

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

.globl Create

.ent Create

Create:

addiu $2, $0, SC\_Create

syscall

j $31

.end Create

* Bước 3: ./code/userprog/exception.cc sửa điều kiện if thành switch.. case (chỉ sửa một lần duy nhất, lần sau cứ theo format sẵn mà làm cho từng case system call)
* Bước 4: Viết chương trình ở mức người dùng để kiểm tra file .c ./code/test Sử dụng hàm như đã khai báo prototype ở /code/userprog/syscall.h
* Bước 5: ./code/test/Makefile

Thêm tên chương trình (tên file) vào dòng all

all: halt shell matmult sort (tên file chương trình ở ./test)

Thêm đoạn sau phía sau malmult

<tên file>o: <tên file>.c

$(CC) $(CFLAGS) -c <tên file>.c

<tên file>: <tên file>.o start.o

$(LD) $(LDFLAGS) start.o <tên file>.o -<tên file>.coff

../bin/coff2noff <tên file>.coff <tên file>

* Bước 6: Biên dịch lại nachos, cd tới ./nachos/code chạy lên “gmake all”.
* Bước 7: Chạy thử chương trình:

./userprog/nachos -rs 1023 -x ./test/<tên chương trình>.

1. **XỬ LÝ EXCEPTION VÀ CÀI ĐẶT SYSTEM CALL**
2. **Xử lý các exception**

Hầu hết các exception đều là **run-time errors** ta có thể thấy rõ từ file **Machine.h/Machine.cc**, khi các exception này xảy ra thì user program không thể khôi phục được.

Chỉ có một trường hợp đặc biệt **No Exception** sẽ trả về quyền điều khiển của hệ điều hành.

Các system call được gọi bởi hàm user system call: SC\_Halt, SC\_Create, SC\_ReadInt, SC\_PrintInt, SC\_ReadChar, SC\_PrintChar, SC\_ReadString, SC\_PrintString.

Đối với các exception khác thì gọi tên của các exception và trả về hàm Halt() của hệ thống.

1. **Xử lý việc tăng giá trị của program counter:**

Khi chạy một system call trong NachOS thì mỗi system call sẽ yêu cầu tăng giá trị của program counter để tránh việc vị lặp mãi một system call do việc đè lên kết quả trả về.

1. **Cài đặt syscall**

* **int CreateFile(char \*name)**
* CreateFile system call sẽ sử dụng Nachos FileSystem Object để tạo một file rỗng. Ban đầu chuỗi name đang ở trong user space, do đó buffer phải lưu lại giá trị chuỗi name này trong system space. System call CreateFile trả về 0 nếu thành công và -1 nếu có lỗi.
* Mô tả cài đặt **SC\_CreateFile**:
* Input: Địa chỉ chứa tên file ở User Space
* Output: -1 lỗi, 0 thành công.
* Mục đích: tạo ra file rỗng với tham số là tên file.
* Ta đọc địa chỉ của tham số **name** từ thanh ghi r4, sau đó thực hiện chép giá trị ở **r4** từ vùng nhớ User sang System bằng hàm **User2System()**.Giá trị chép được thực sự chính là tên file. Ta tiếp tục kiểm tra tên file có NULL không và file có được tạo ra với tên file đó không. Nếu thành công thì trả về 0, ngược lại thì trả về -1 vào thanh ghi **r2**.
* **OpenFileID Open(char \*name, int type) và int Close(OpenFileID id)**
* User program có thể mở 2 loại file, file chỉ đọc và file đọc và ghi. Mỗi tiến trình sẽ được cấp một bảng mô tả file với kích thước cố định. Kích thước của bảng mô tả file có thể lưu được đặc tả của 10 files. Trong đó, 2 phần tử đầu, ô 0 và ô 1 để dành cho console input và console output.
* System call mở file phải làm nhiệm vụ chuyển đổi địa chỉ buffer trong user space khi cần thiết và viết hàm xử lý phù hợp trong kernel. Dùng class **FileSystem** trong thư mục **filesys**, System call Open sẽ trả về id của file (**OpenFileID** = một số nguyên), hoặc là -1 nếu quá trình mở file bị lỗi.
* Mở file có thể bị lỗi như trường hợp là không tồn tại tên file hay không đủ ô nhớ trong bảng mô tả file. Tham số type = 0: mở file đọc và ghi, type = 1: chỉ đọc, type = 2: chạy stdin, type = 3: chạy stdout. Nếu tham số truyền bị sai thì system call phải báo lỗi. System call sẽ trả về -1 nếu bị lỗi và 0 nếu thành công.
* Open(char \*name, int type).
* Quy ước giá trị của type:
* Type =0: đọc và ghi.
* Type =1: chỉ đọc.
* Type =2: stdin.
* Type =3: stdout.
* Mô tả cài đặt **SC\_Open**:
* Input: Địa chỉ của tên file trên user space, chế độ mở (type).
* Output: id file nếu thành công, -1 nếu lỗi.
* Mục đích: mở một file với tham số số truyền vào gồm tên file và cách mở file.

Ta đọc địa chỉ của tham số name từ thanh ghi **r4** và tham số **type** từ thanh ghi **r5** sau đó kiểm tra **type** có hợp lệ hay không (0<= **type** <=3), kiểm tra **index** của file trong lớp **FileSystem** có nằm trong bảng mô tả file không – mỗi tiến trình Read/Write sẽ được cấp một bảng đặc tả file có kích thước là 10 - (0 <= **index** <=9). Nếu kiểm tra hai điều kiện trên hợp lệ thì thực hiện chép giá trị ở r4 từ phía User sang System bằng hàm **User2System()**. Giá trị chép được thực sự chính là tên file. Ta tiếp tục kiểm tra tên file, nếu là stdin và type truyền vào là 2 thì trả về cho thanh ghi **r2** id của file là 0, nếu là stdout và type truyền vào là 3 thì trả về cho thanh ghi **r2** id của file là 1, nếu là file bình thường thì type phải khác 2 và 3 và trả về cho thanh ghi **r2** đúng id của file bằng (**index** – 1) của lớp **FileSystem**. Trường hợp mở file không tồn tại hoặc ngược lại với tất cả điều kiện trên thì trả về -1 cho thanh ghi r2.

* int Close(OpenFileID id).

Mô tả cài đặt **SC\_Close**:

* Input: ID file.
* Output: NULL.
* Mục đích: Đóng file với tham số truyền vào là ID của file.

Đọc tham số id của file từ thanh ghi **r4**, sau đó kiểm tra xem file cần đóng có tồn tại không bằng **openf[id]** đã cài đặt trong lớp **FileSystem** và kiểm tra id của file có nằm ngoài bảng mô tả file không. Nếu có một trong hai lỗi trên thì xuất ra thông báo lỗi và gọi syscall **Halt()** để tắt hệ thống, nếu thành công thì xóa đi dữ liệu **openf[id]** và gán lại bằng NULL.

* **int Read (char\* buffer, int charcount, OpenFileID id) và int Write (char\* buffer, int charcount, OpenFileID id)**

Các System call đọc và ghi vào file với id cho trước. Phải chuyển vùng nhớ giữa user space và system space, cần phân biệt giữa **Console IO** (OpenFileID 0, 1) và **File**. Lệnh **Read** và **Write** sẽ làm việc như sau: Phần console read và write sẽ sử dụng lớp **SynchConsole**. Được khởi tạo qua biến toàn cục **gSynchConsole**. Sử dụng các hàm mặc định của **SynchConsole** để đọc và ghi. Đọc và ghi với Console sẽ trả về số bytes đọc và ghi thật sự, chứ không phải số bytes được yêu cầu. Trong trường hợp đọc hay ghi vào console bị lỗi thì trả về -1. Nếu đang đọc từ console và chạm tới cuối file thì trả về -2. Đọc và ghi vào console sẽ sử dụng dữ liệu ASCII để cho input và output, (ASCII dùng kết thúc chuỗi là NULL (\0)). Phần đọc, ghi vào file sẽ sử dụng các lớp được cung cấp trong file system. Sử dụng các hàm mặc định có sẳn của **FileSystem** và thông số trả về cũng phải giống như việc trả về trong **synchconsole**. Cả read và write trả số kí tự đọc, ghi thật sự. Cả Read và Write trả về -1 nếu bị lỗi và -2 nếu cuối file. Cả Read và Write sử dụng dữ liệu binary.

* int Read (char\* buffer, int charcount, OpenFile id).

Mô tả cài đặt **SC\_Read**:

* + - Input: Buffer, số ký tự cho phép, id của file.
    - Ouput: -1 nếu lỗi, -2 nếu thành công với số byte được đọc.
    - Mục đích: Đọc file với tham số là **buffer**, số ký tự cho phép (**charcount**) và **id** của file.

Ta đọc địa chỉ của tham số **buffer** từ thanh ghi **r4**, tham số **charcount** từ thanh ghi **r5** và **id** của file từ thanh ghi **r6**, sau đó ta tiến hành kiểm tra id của file truyền vào có nằm ngoài bảng mô tả file không, file cần đọc có tồn tại không và file cần đọc có phải là stdout với type = 3 không. Nếu vi phạm các điều kiện trên thì trả về -1 cho thanh ghi **r2** ngược lại là hợp lệ thì lấy vị trí con trỏ ban đầu trong file bằng phương thức **GetCurrentPos()** của lớp **FileSystem** gọi là **OldPos** và thực hiện chép giá trị ở r4 từ phía User sang System bằng hàm **User2System()**. Giá trị chép được là **buffer** chứa chuỗi kí tự. Xét trường hợp đọc file stdin với type = 2, ta gọi phương thức Read của lớp **SynchConsole** đọc buffer với độ dài charcount, trả về số byte thực sự đọc được cho thanh ghi **r2** và chép **buffer** từ phía System sang User bằng hàm **System2User()**. Xét trường hợp đọc file bình thường, thì ta lấy vị trí con trỏ hiện tại trong file bằng phương thức **GetCurrentPos()** của lớp **FileSystem** gọi là **NewPos**, trả về số byte thực sự đọc được cho thanh ghi r2 bằng công thức: **NewPos – OldPos** và cũng chép **buffer** từ phía System sang User bằng hàm **System2User().** Trường hợp còn lại là đọc file rỗng thì trả về -2 cho thanh ghi **r2**.

* int Write (char\* buffer, int charcount, OpenFileID id).

Mô tả cài đặt **SC\_Write**:

* Input: Buffer, số ký tự cho phép, id của file.
* Ouput: -1 nếu lỗi, Số byte thực sự ghi được nếu thành công.
* Mục đích: Ghi file với tham số là **buffer**, số ký tự cho phép (**charcount**) và **id** của file.

Ta đọc địa chỉ của tham số buffer từ thanh ghi **r4**, tham số **charcount** từ thanh ghi **r5** và **id** của file từ thanh ghi **r6**, sau đó ta tiến hành kiểm tra id của file truyền vào có nằm ngoài bảng mô tả file không, file cần ghi có tồn tại không và file cần ghi có phải là stdin với type = 2 hay là file chỉ đọc với type = 1. Nếu vi phạm các điều kiện trên thì trả về -1 cho thanh ghi **r2** ngược lại là hợp lệ thì lấy vị trí con trỏ ban đầu trong file bằng phương thức **GetCurrentPos()** của lớp **FileSystem** gọi là **OldPos** và thực hiện chép giá trị ở r4 từ phía User sang System bằng hàm **User2System()**. Giá trị chép được là **buffer** chứa chuỗi kí tự. Xét trường hợp ghi file đọc và ghi với type = 0, thì ta lấy vị trí con trỏ hiện tại trong file bằng phương thức **GetCurrentPos()** của lớp **FileSystem** gọi là **NewPos**, trả về số byte thực sự ghi được cho thanh ghi **r2** bằng công thức: **NewPos – OldPos**. Xét trường hợp ghi file stdout với type = 3, ta gọi phương thức Write của lớp **SynchConsole** để ghi từng kí tự trong buffer và kết thúc là kí tự xuống dòng ‘\n’, trả về số byte thực sự ghi được cho thanh ghi **r2**.

* **int Seek (int pos, OpenFileID id )**

**Seek** sẽ phải chuyển con trỏ tới vị trí thích hợp. **Pos** lưu vị trí cần chuyển tới, nếu pos = -1 thì di chuyển đến cuối file. Trả về vị trí thực sự trong file nếu thành công và -1 nếu bị lỗi. Gọi Seek trên console phải báo lỗi.

Mô tả cài đặt **SC\_Seek:**

* Input: Vị trí cần chuyển tới, id của file.
* Output: -1: Lỗi, Vị trí thực sự trong file: Thành công.
* Mục đích: Di chuyển con trỏ đến vị trí thích hợp trong file với tham số là vị trí cần dịch chuyển và id của file.

Ta đọc tham số pos từ thanh ghi **r4** và id của file từ thanh ghi **r5**, sau đó ta tiến hành kiểm tra id của file truyền vào có nằm ngoài bảng mô tả file không, file cần di chuyển con trỏ có tồn tại không và kiểm tra người dùng có gọi **Seek** trên console không. Nếu vi phạm các điều kiện trên thì trả về -1 cho thanh ghi **r2** ngược lại là hợp lệ thì kiểm tra nếu pos = -1 thì gán pos bằng độ dài của file bằng phương thức **Length()** của lớp **FileSystem**. Gọi phương thức Seek của lớp **FileSystem** với tham số truyền vào là **pos** để dịch chuyển con trỏ đến vị trí mong muốn và trả về vị trí dịch chuyển cho **r2**.

* **Chương trình echo**

Với chương trình này, mỗi khi nhập một dòng từ console thì console xuất lại dòng đó.

Ta gọi lại system call Open để mở file stdin với type quy ước bằng 2 và stdout với type quy ước bằng 3. Nếu mở thành công thì gọi system call Read đọc nội dung nhập từ stdin vào buffer đồng thời trả về số byte thực sự đọc được, sau đó gọi system call Write để ghi nội dung vừa đọc từ buffer ra stdout với charcount bằng sốbyte thực sự đọc được ở trên. Cuối cùng là đóng file stdin và file stdout với system call Close.

* **Chương trình cat**

Với chương trình này, yêu cầu nhập vô filename rồi hiển thị nội dung của file đó.

Ta gọi lại system call ReadString để nhập vào filename. Sau đó gọi system call Open để mở file đó với type bằng 1, nếu mở file thành công thì gọi system call Seek để dịch chuyển con trỏ về cuối file lấy kích thước thực sự của file và di chuyển lại con trỏ ra đầu file. Tiến hành đọc từng kí tự trong file bằng system call Read và in từng kí tự đó ra màn hình bằng system call PrintChar cho đến hết kích thước của file. Cuối cùng là đóng file với system call Close.

* **Chương trình copy**

Chương trình này yêu cầu nhập tên file nguồn và file đích và thực hiện copy nội dung từ file nguồn sang file đích.

Ta gọi lại system call ReadString để nhập vào tên file nguồn và tên file đích. Sau đó gọi system call Open để mở file nguồn với type bằng 1 và file đích với type bằng 0, id của file nguồn và file đích phải khác nhau để tránh trường hợp người dùng mở cùng một file. Nếu mở file thành công thì gọi system call Seek để trỏ đến cuối file nguồn và lấy kích thước thực sự của file nguồn, di chuyển con trỏ trở lại đầu file nguồn và đầu file đích. Tiến hành đọc từng kí tự từ đầu file nguồn bằng system call Read và ghi từng kí tự đó ra file đích bằng system call Write cho đến hết kích thước của file nguồn. Cuối cùng là đóng file nguồn và file đích với system call Close.

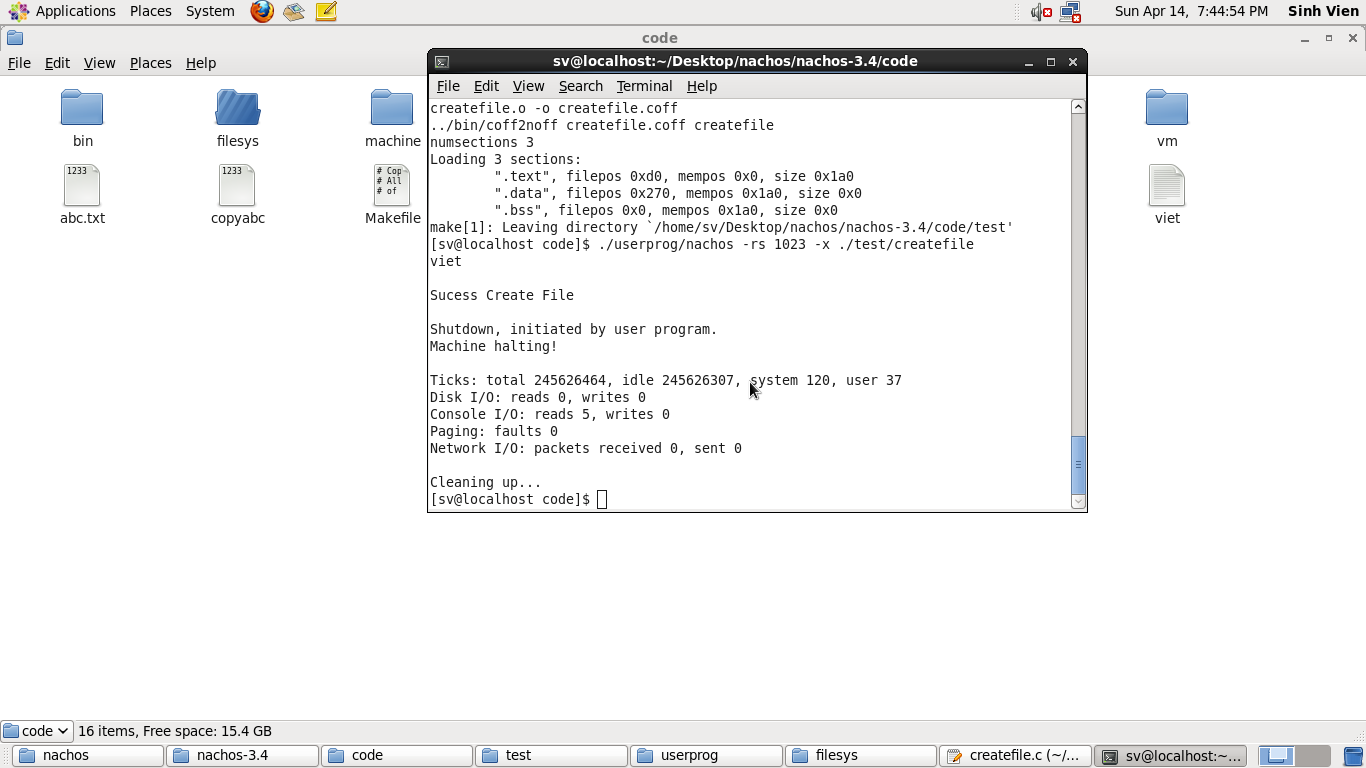
* **Chương trình reverse**

Chương trình này yêu cầu nhập vô file nguồn và file đích. Thực hiện đảo ngược file nguồn rồi viết vô file đích.

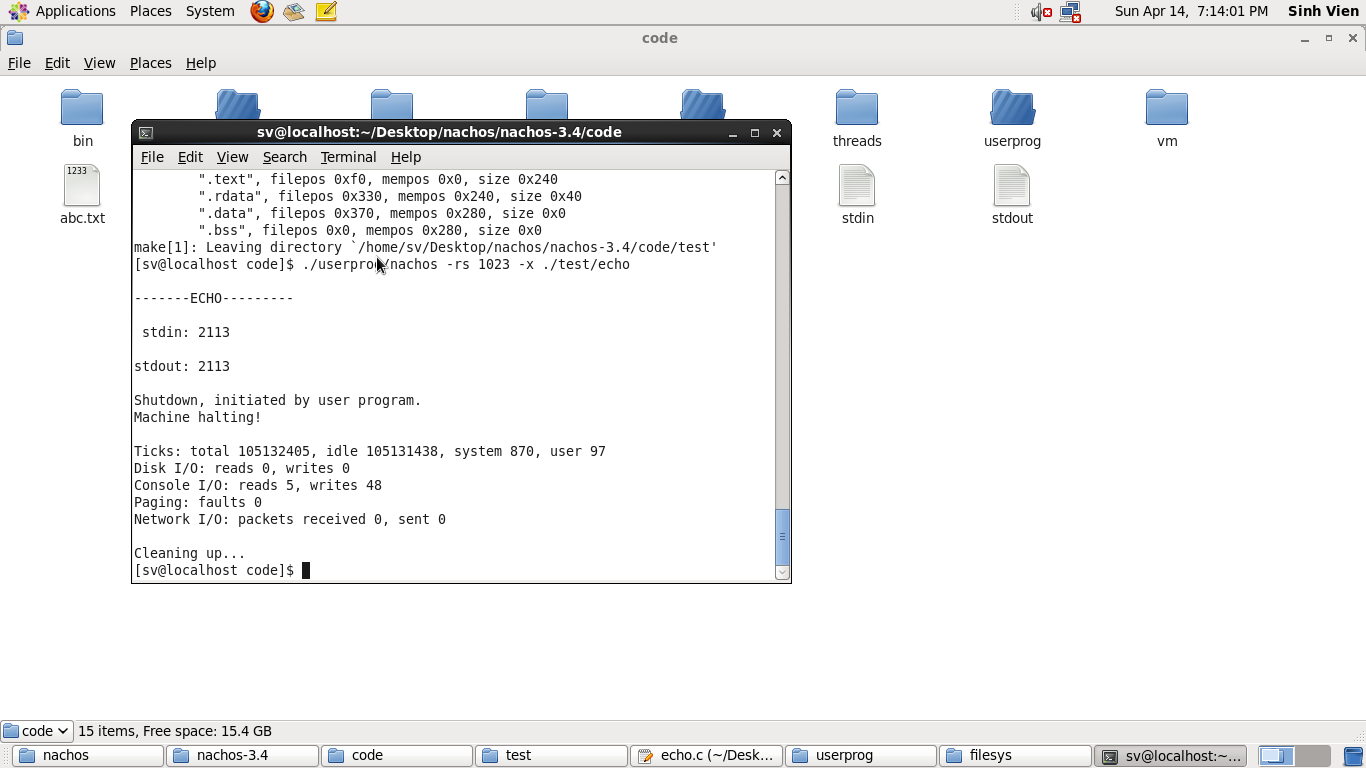
* **Chương trình append**

Chương trình này yêu cầu nhập vô 2 file nguồn và 1 file đích. Nối file nguồn thứ hai vào cuối file nguồn thứ nhất rồi viết vào file đích.

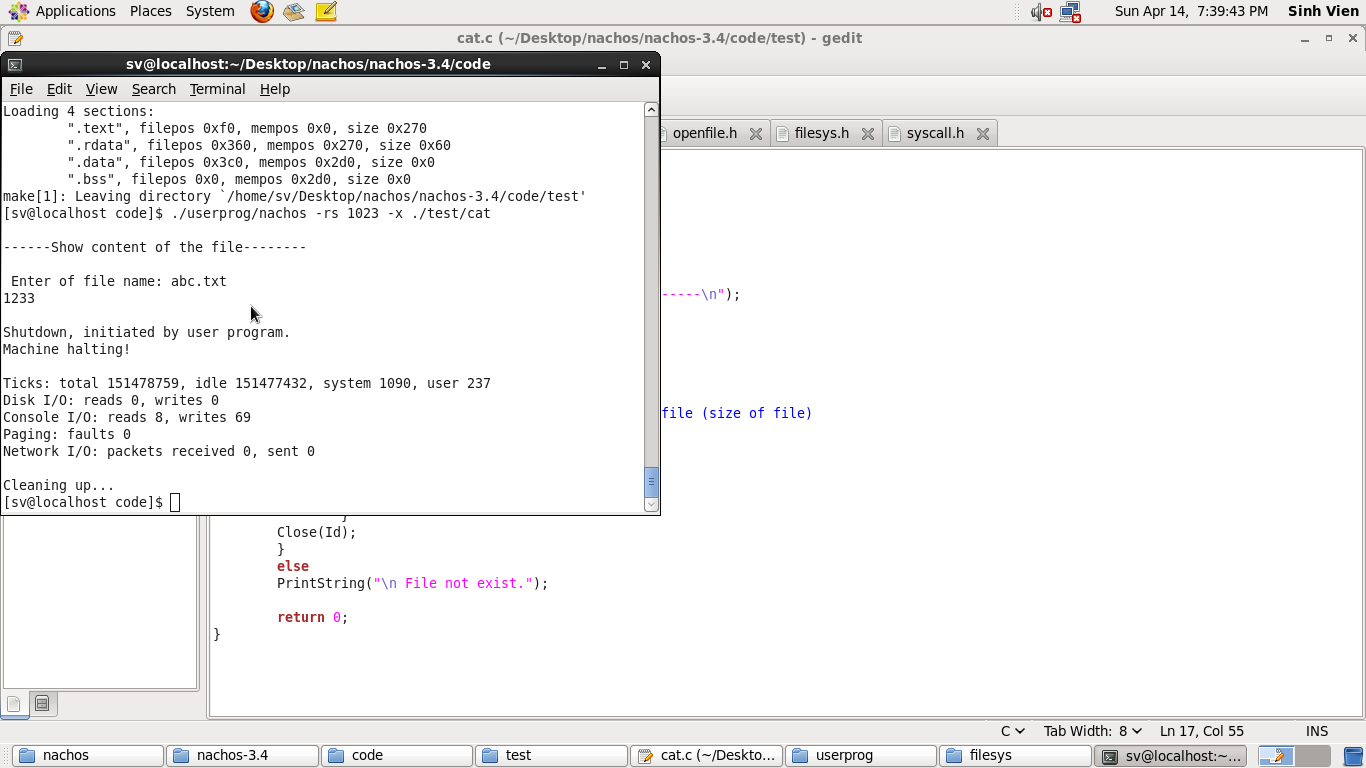
1. **DEMO HÌNH ẢNH MINH HỌA**
2. **Createfile**

****

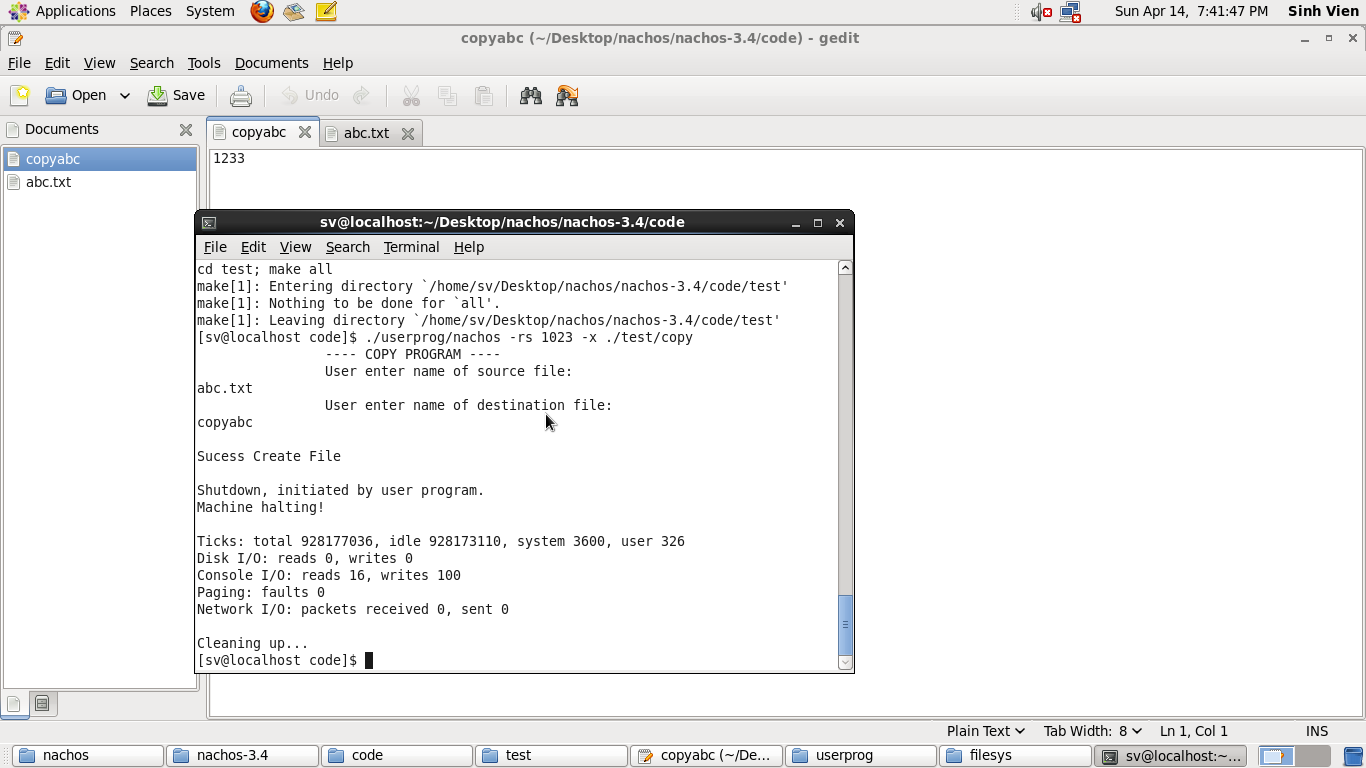
1. **Echo**

****

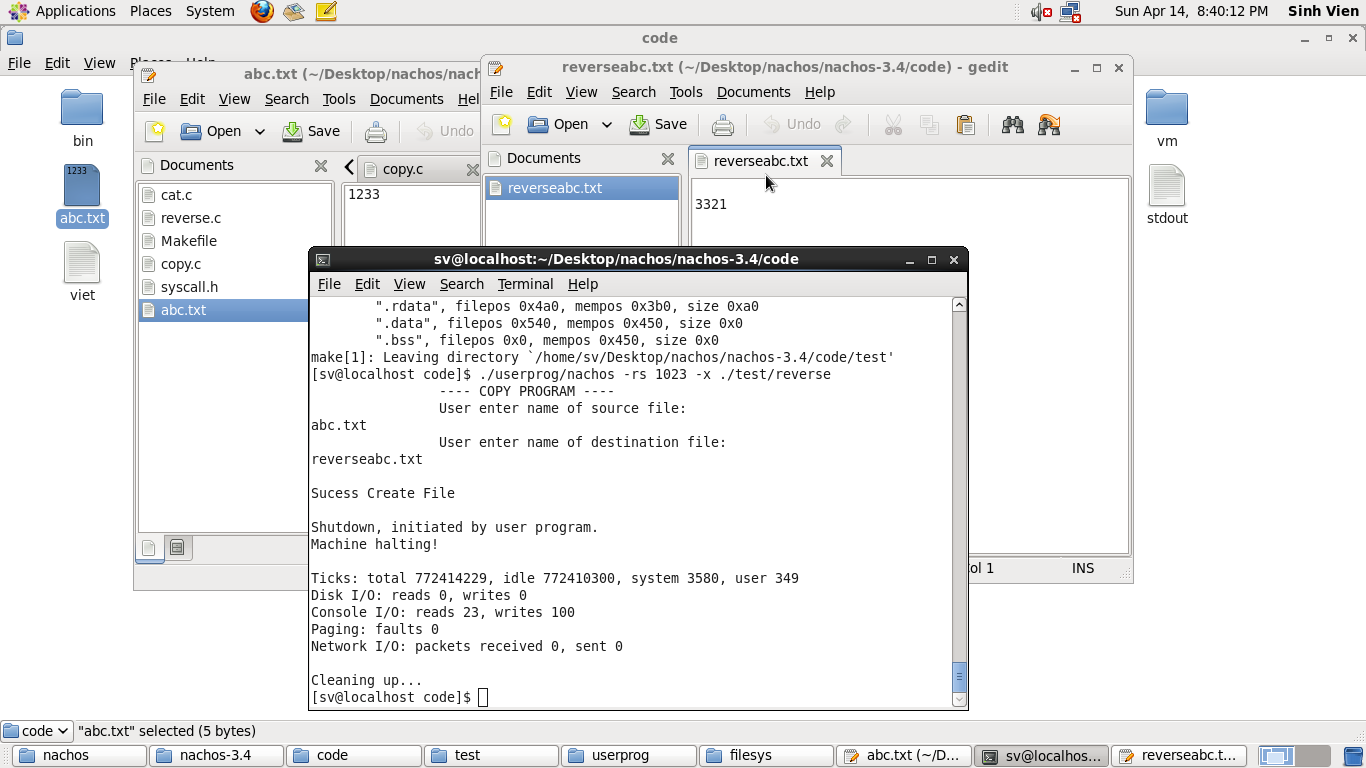
1. **Cat**

****

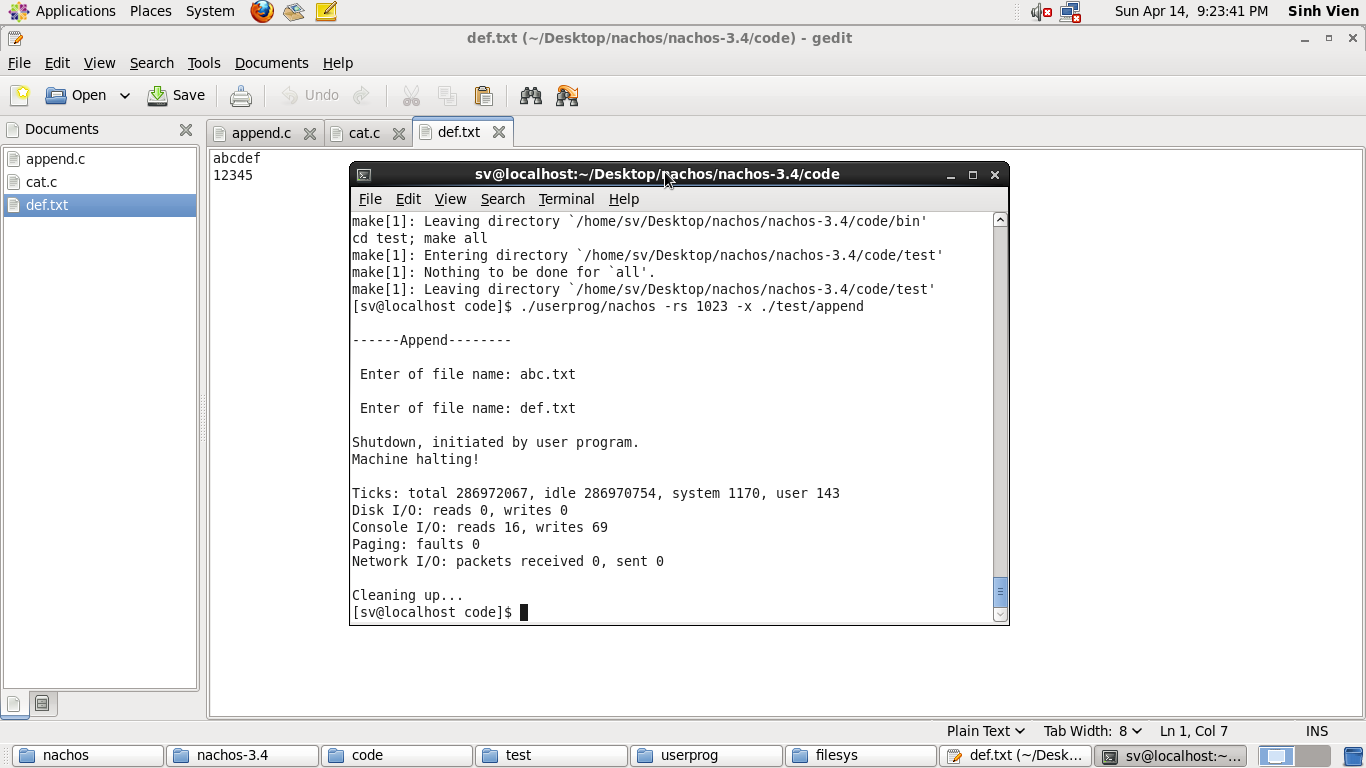
1. **Copy**

****

1. **Reverse**

****

1. **Append**

****