Trường Đại Học Khoa Học Tự Nhiên

Đại Học Quốc Gia Hồ Chí Minh

**BÀI BÁO CÁO**

**ĐỒ ÁN – HỆ ĐIỀU HÀNH**

**THÔNG TIN NHÓM**

1. Trần Duy Phương – 1512418
2. Dương Vinh Quang – 1512428
3. Nguyễn Đăng Quang – 1512432

NỘI DUNG

[I. BÁO CÁO CÁCH THIẾT KẾ VÀ CÀI ĐẶT CÁC SYSCALL TRÊN NACHOS 2](#_Toc496905437)

[1. Moodle 1: hiểu mã chương trình nachos 2](#_Toc496905438)

[2. Moodle 2: Hiểu thiết kế của hệ điều hành 2](#_Toc496905439)

[3. Moodle 3: Cách viết và cài đặt các syscall trên nachos 3](#_Toc496905440)

[II. CÁC CHỨC NĂNG LÀM ĐƯỢC VÀ CHƯA LÀM ĐƯỢC 5](#_Toc496905441)

[A. Đã làm được 5](#_Toc496905442)

[B. Chưa làm được 5](#_Toc496905443)

[III. ĐÁNH GIÁ MỨC ĐỘ HOÀN THÀNH ĐỒ ÁN: 100% 6](#_Toc496905444)

# BÁO CÁO CÁCH THIẾT KẾ VÀ CÀI ĐẶT CÁC SYSCALL TRÊN NACHOS

* Lưu ý:
* Nachos là phần mềm giả lâp hệ điều hành với kiến trúc MIPS, ở đồ án này Nachos được chạy trên môi trường LINUX với kiến trúc x86
* GCC 3x là cross-compiler để biên dịch các chương trình C thành các chương trình thực thi trên hệ điều hành LINUX cho kiến trúc máy MIPS

## Moodle 1: hiểu mã chương trình nachos

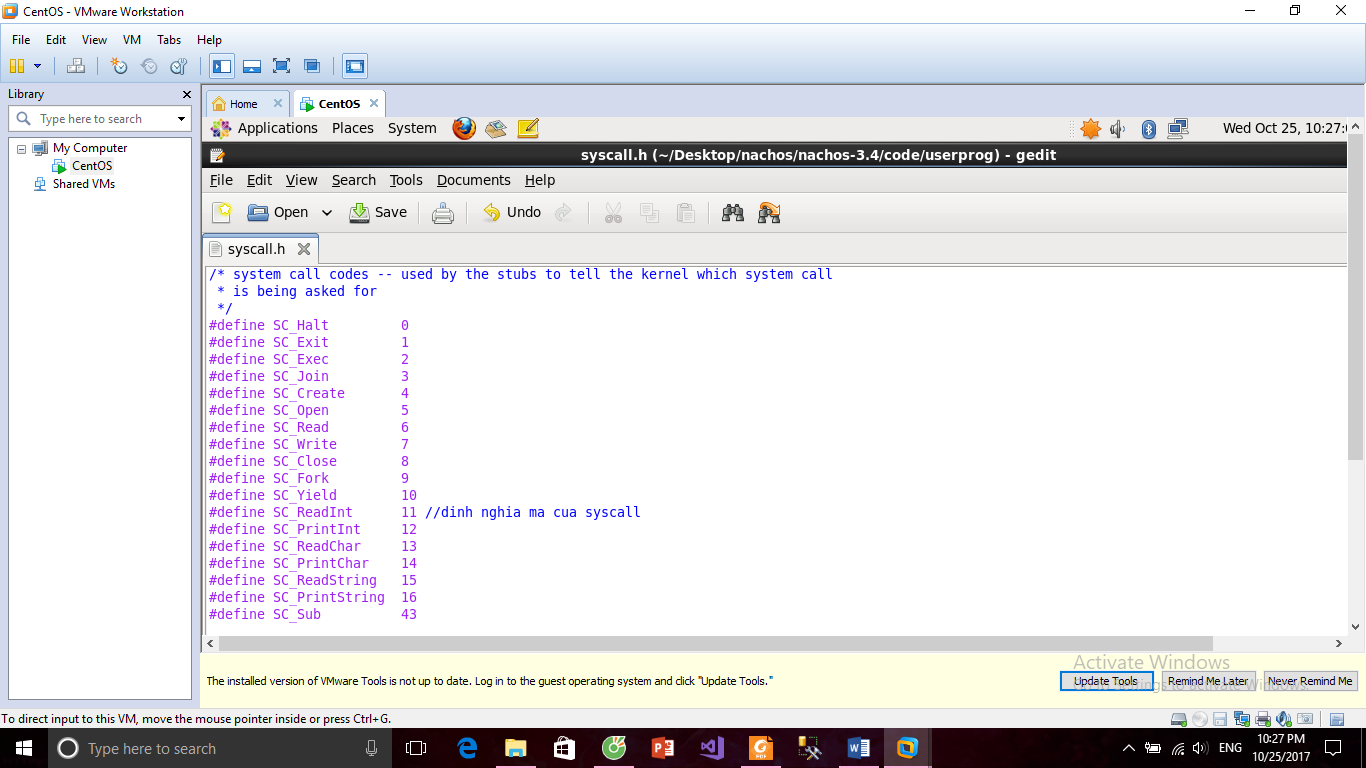
* Lệnh *./userprog/nachos –rs 1023 –x ./test/halt* có ý nghĩa như sau :
* *./userprog/nachos* là đường dẫn đến NachOS , có nghĩa là mở NachOS lên
* *–rs 1023 –x* , trong đó –rs và –x là các tham số , 1023 là một con số tùy chọn, chúng cách nhau bằng 1 khoảng trắng.
* *./test/halt* là đường dẩn của file halt.noff đã được biên dịch xong ta mở halt lên nạp vào MIPS ảo rồi chạy
* Nguyên câu lệnh này là khởi động máy ảo MIPS và hệ điều hành NachOS lên để nạp và thực hiện chương trình halt (tắt máy)
* Các file trong thư mục code/test sẽ được biên dịch bởi GCC 2.95.3 thành file.coff , rồi được phần mềm coff2noff chuyển từ đuôi .coff sang đuôi .noff. Vì ta sửa trong Makefile của thư mục *code/test* cái GCCDIR, nên khi Linux dùng GCC 3x biên dịch NachOS , nó sẽ chừa thư mục *code/test* ra và Linux sẽ cho GCC 2.95.3 ra biên dịch thư mục *code/test* đó
* Các chương trình người dùng trên NachOS nằm trong thư mục code/test sẽ được biên dịch như sau: khi biên dịch NachOS GCC sẽ biên dịch file abc.c được viết bằng C chuẩn do người lập trình làm ra
* Trong *code/test* có sẵn 1 file hợp ngữ start.s (đây được coi là file thư viện). Khi biên dịch, nó sẽ biên dịch file mã nguồn abc.c thành file hợp ngữ abc.s
* Sau đó lấy file abc.s này link với file start.s tạo thành 1 file tổng hợp abc.noff (bao gồm abc.o và start.o ) . Đây là file thực thi cho Linux/MIPS
* Sau đó dùng phần mềm coff2noff (trong thư mục bin) chuyển qua file abc.noff. Đây là file thực thi cho NachOS/MIPS

## Moodle 2: Hiểu thiết kế của hệ điều hành

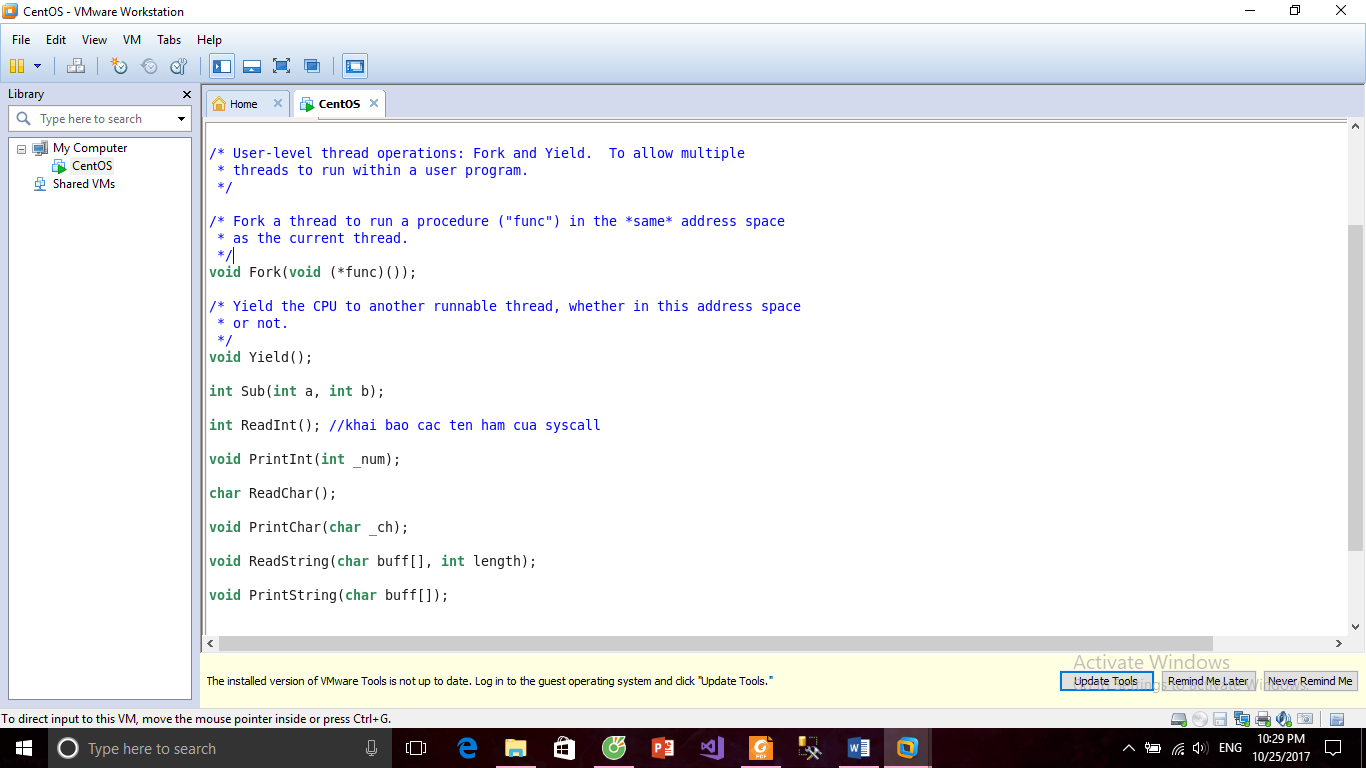
* Mỗi chương trình trong hệ thống phải có các thông tin cục bộ của nó, bao gồm program counters, registers, stack pointers, và file system handler. Trước khi gọi một lệnh trong hệ thống thì các tham số truyền vào cần thiết phải được nạp vào các thanh ghi của CPU và các tham số truyền vào thanh ghi bắt đầu từ thanh ghi R4 trở đi.
* Để chuyển một biến mang giá trị, tiến trình chỉ việc ghi giá trị vào thanh ghi. Để chuyển một biến tham chiếu, thì giá trị lưu trong thanh ghi được xem như là “user space pointer”. Bởi vì userspace pointer không có ý nghĩa đối với kernel, mà chúng ta cần là chuyển nội dung từ user space vào kernel sao cho ta có thể xử lý dữ liệu này. Khi trả thông tin từ system về user space, thì các giá trị phải đặt trong các thanh ghi của CPU.
* Bên cạnh đó ta cần có 2 hàm để chuyển từ USER SPACE sang kernel space và ngược lại vì hệ điều hành chỉ hiểu những biến trong KERNEL SPACE
* Hệ thống nachos gồm chương trình ứng dụng do người dùng viết nằm trong thư mục code/test, hệ điều hành nachos cấu thành từ các mã nguồn (userprog, threads,..) và máy ảo giả lập trong class machine với các thành phần Ram, thanh ghi,…

## Moodle 3: Cách viết và cài đặt các syscall trên nachos

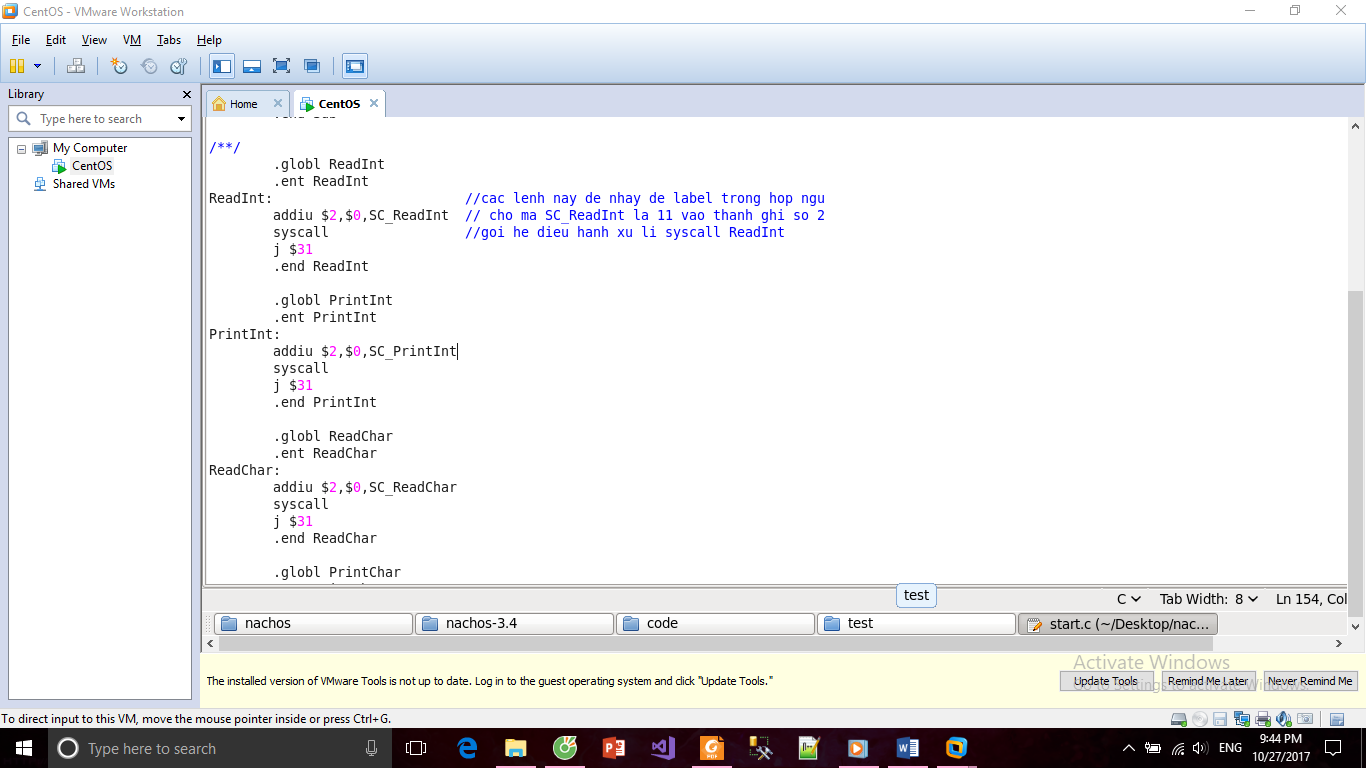
* Cách thêm 1 syscall vào nachos
* Để viết 1 syscall ta phải viết 2 thủ tục để copy vùng nhớ từ user space vào system space và ngược lại
* Những dữ liệu trong chương trình nằm ở USER MODE (vùng nhớ của chương trình chạy trên nachos) khi thực hiện syscall nó đọc lệnh từ thanh ghi R2 mà nó chỉ hiểu những biến nằm trong vùng nhớ của nó tức là ở KERNEL MODE(vùng nhớ của hệ điều hành Nachos), các tham số được truyền vào nằm từ thanh ghi số 4
* Ta dùng hàm *System2User* và *User2System* để chuyển dữ liệu từ KERNEL MODE sang USER MODE và ngược lại
* Bước 1: Đầu tiên để tạo ra 1 syscall bạn vào file syscall.h trong *userprog*/ để định nghĩa mã của syscall



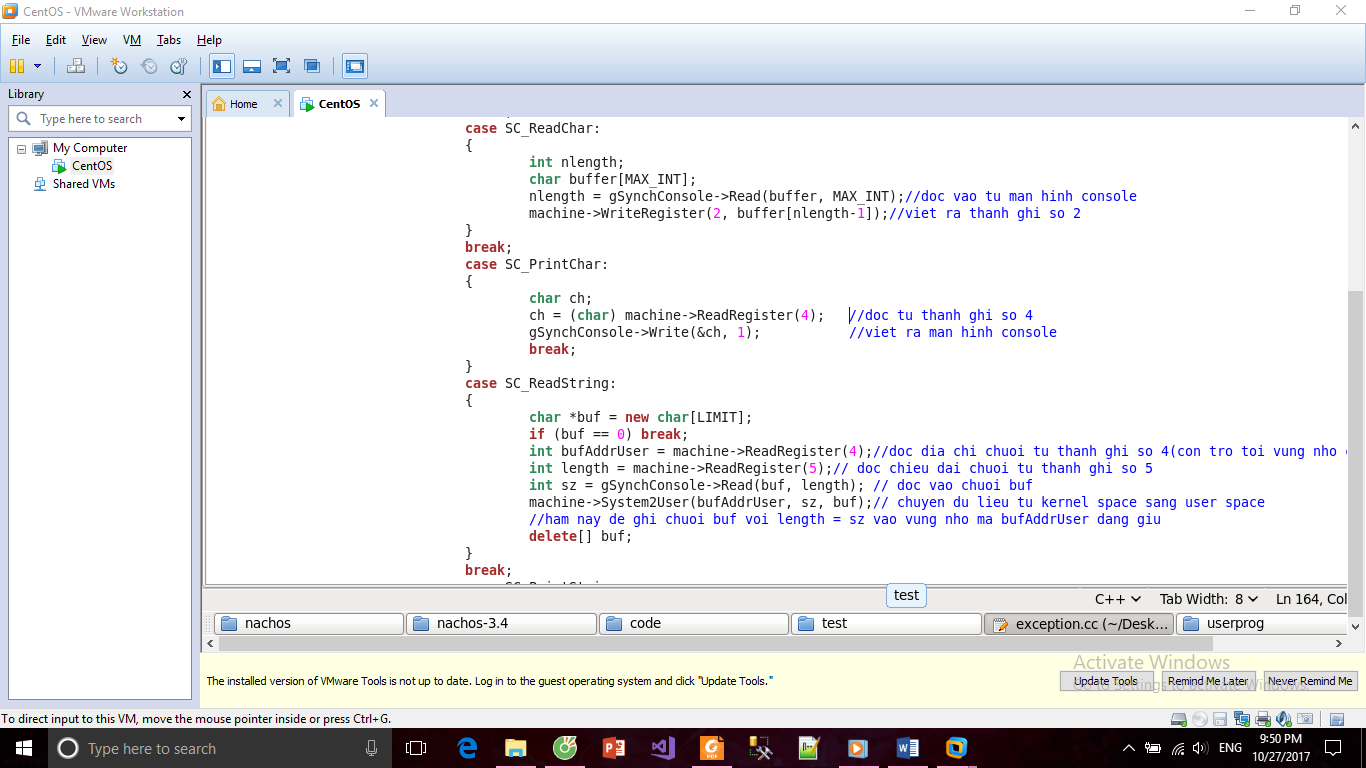
* Sau đó ta khai báo tên hàm của syscall đó



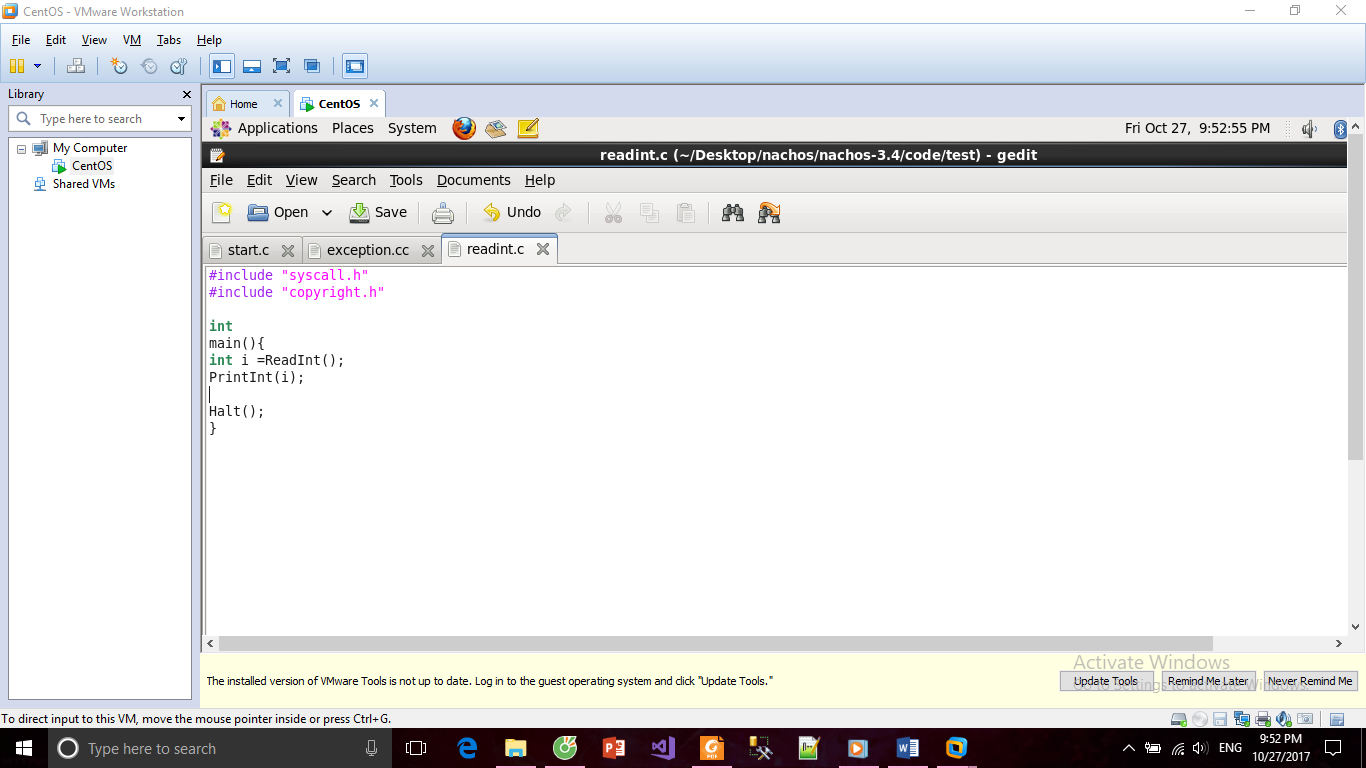
* Bước 2: Thêm vào start.c và start.s trong thư mục *code/test/* phần này để người dùng khi gọi hàm thì nó sẽ nhảy đến các syscall tương ứng



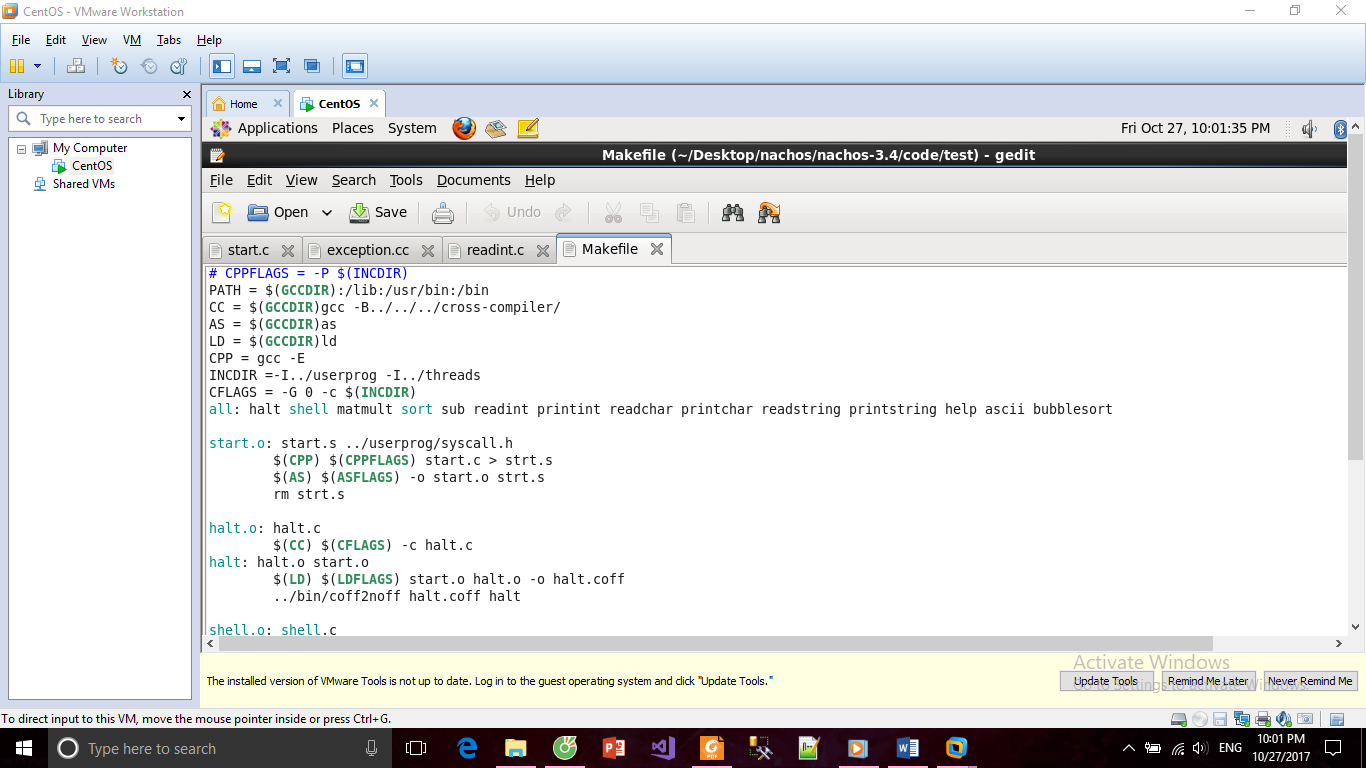
* Bước 3: xử lý syscall trong file *code/userprog/exception.cc*



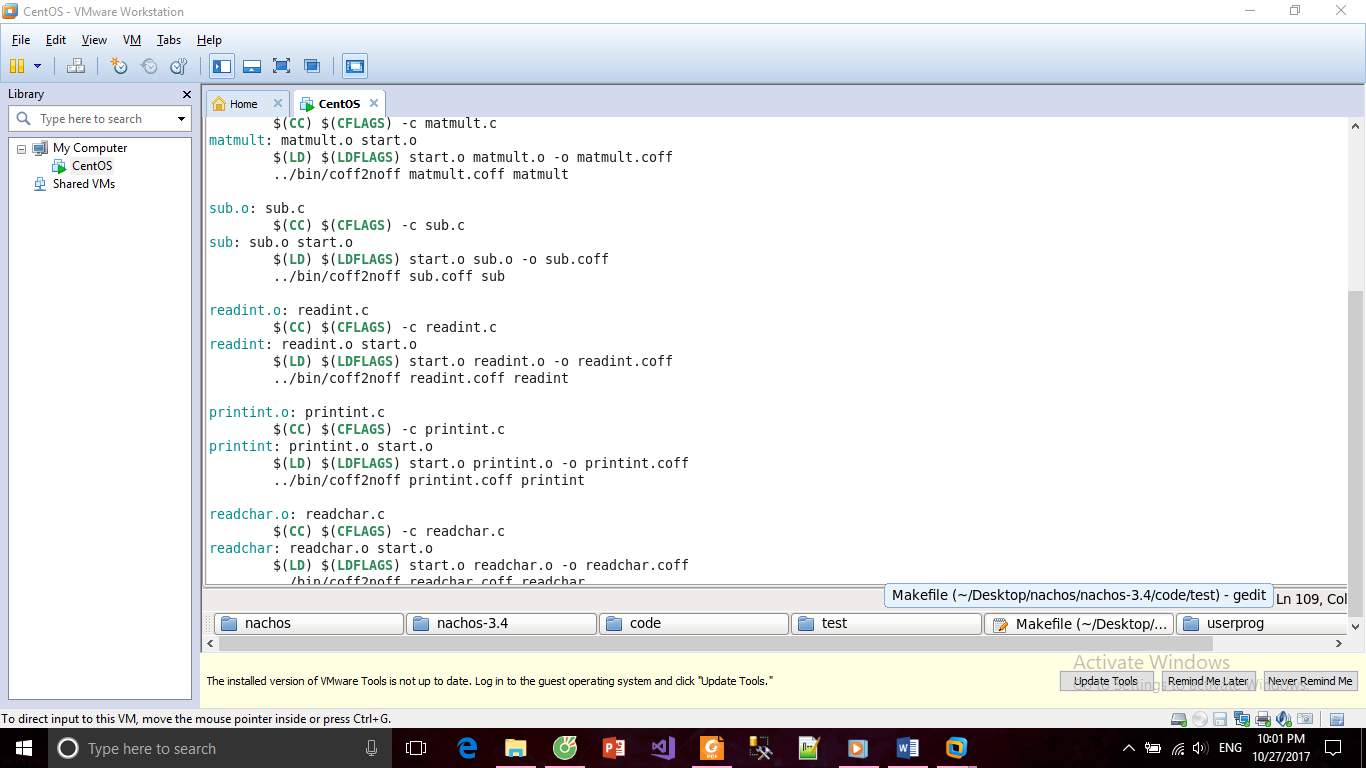
* Bước 4: ta viết chương trình để kiểm tra vd như readint.c



* Bước 5: thêm đoạn vào *Makefile* trong */code/test*/:



* Sau đó ta thêm readint vào dòng all và thêm phía sau đoạn matmult



* Bước 6: biên dịch lại Nachos
* Nhập xuất trên console với synchConsole
* Để hỗ trợ việc nhâp xuất trên console chúng ta sẽ dùng synchConsole bằng cách Copy 2 file sysconsole.c và sysconsole.h vào thư mục *code/threads*.
* Khai báo biến toàn cục gSynchConsole trong file Threads/system.h và cấp phát bộ nhớ, delete nó.
* Kế tiếp các bạn mở file: Makefile.common (trong nachos-3.4/code). Và cuối đoạn bắt
* đầu dòng USERPROG\_H = thêm vào ../threads/synchcons.h. Ví dụ như bên dưới:

USERPROG\_H = ../userprog/addrspace.h\

../userprog/bitmap.h\

../filesys/filesys.h\

../filesys/openfile.h\

../machine/console.h\

../machine/machine.h\

../machine/mipssim.h\

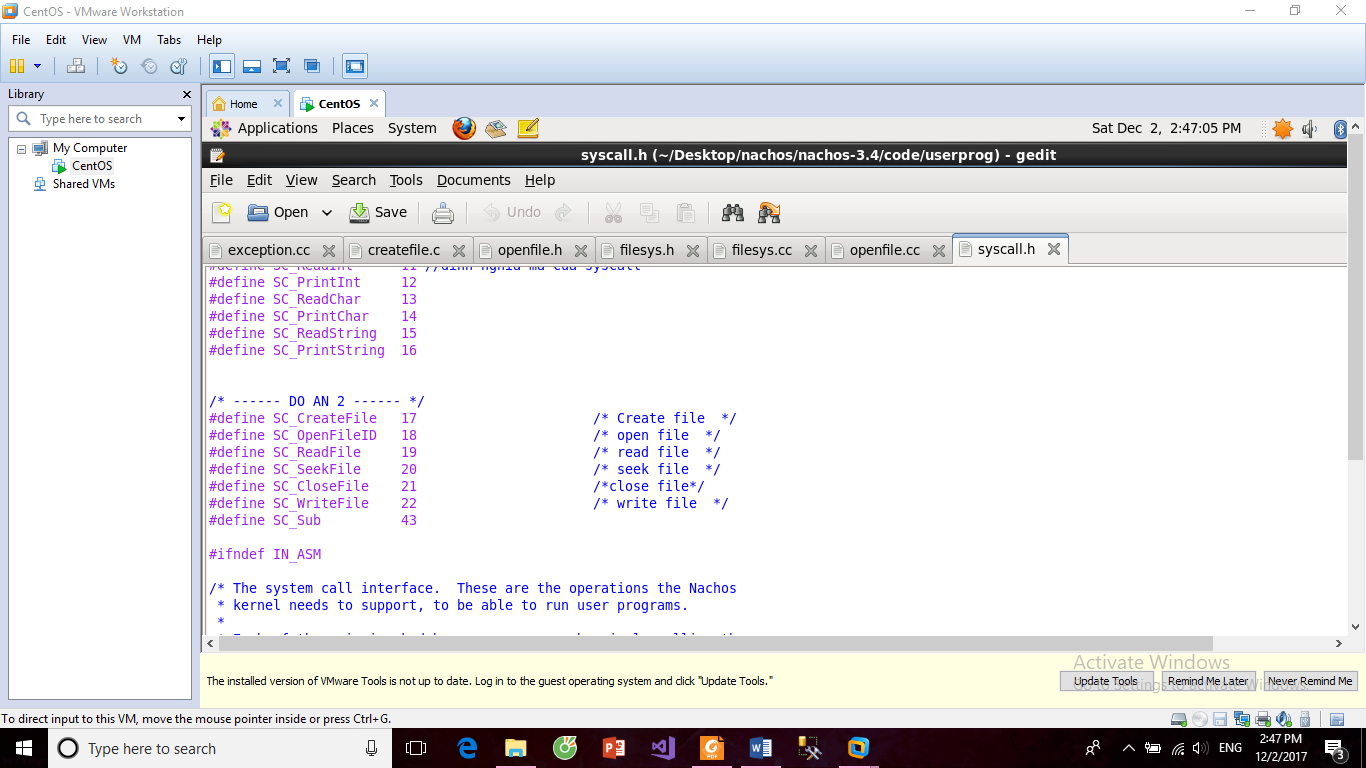
../machine/translate.h\

../threads/synchcons.h

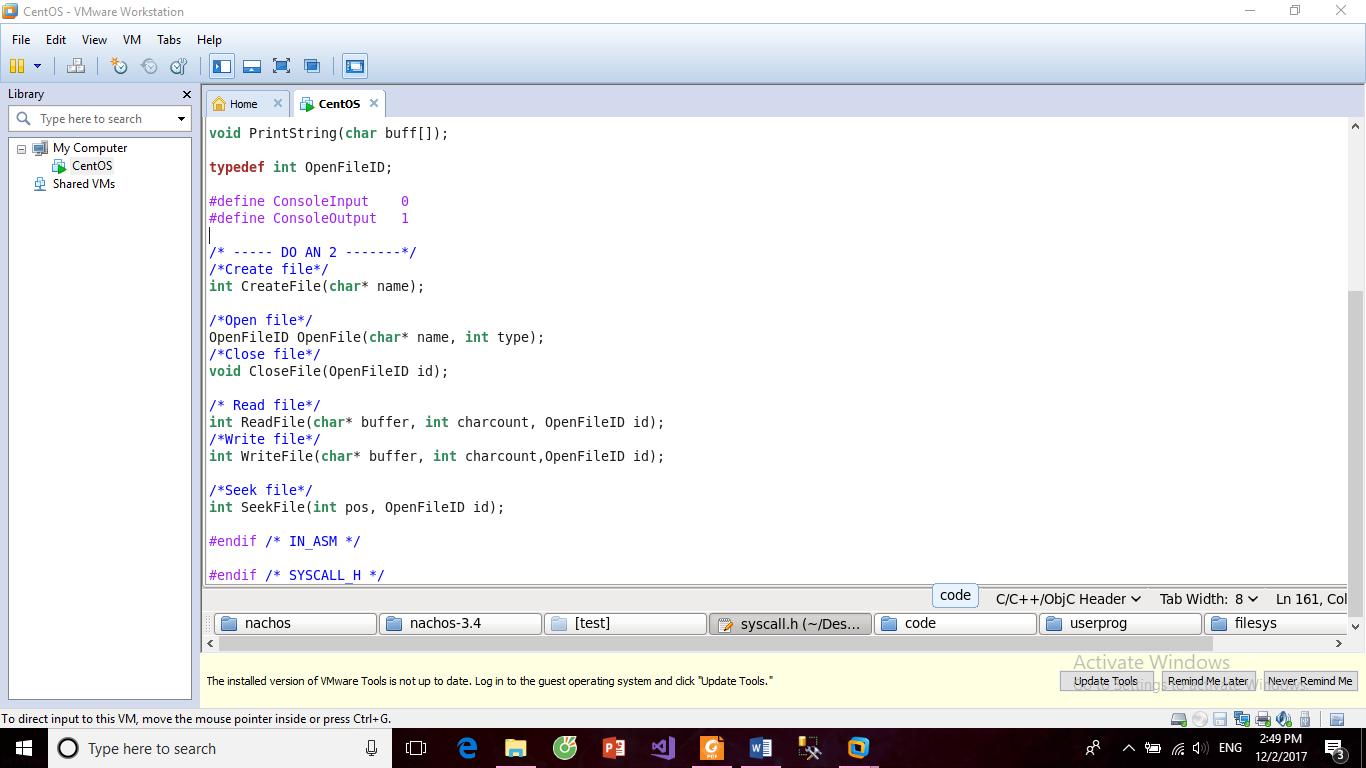
* Và cuối đoạn bắt đầu bằng USERPROG\_C = .. thêm vào ../threads/synchcons.cc
* Và cuối đoạn bắt đầu bằng USERPROG\_O = .. thêm vào synchcons.o
* Biên dịch lại Nachos.

# SYSCALL VỀ HỆ THỐNG QUẢN LÝ TẬP TIN

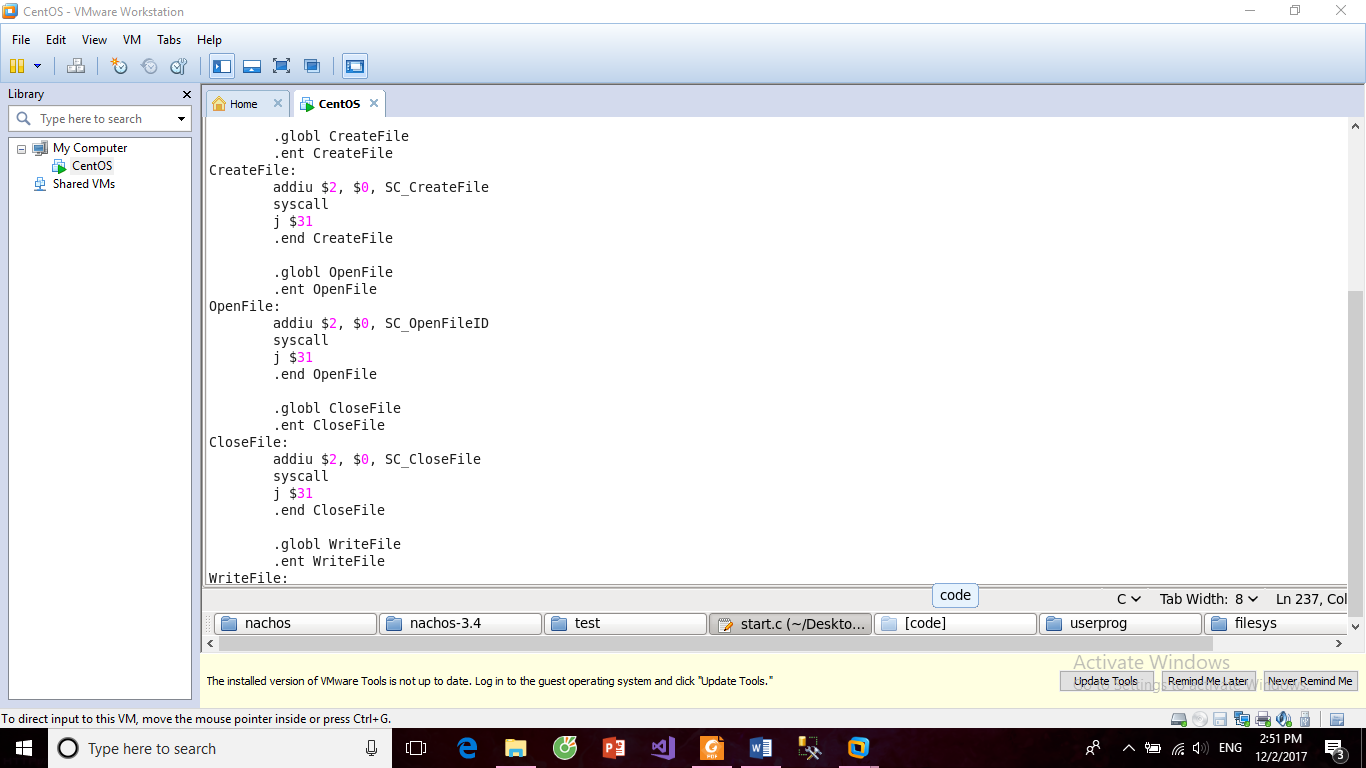
* Đầu tiên để cài đặt syscall CreateFile để tạo 1 file rỗng bạn phải buffer mà con trỏ trong user space sang system space vì file name lúc này đang ở trong user space
* Bước 1: để tạo ra 1 syscall bạn vào file syscall.h trong *userprog*/ để định nghĩa mã của syscall



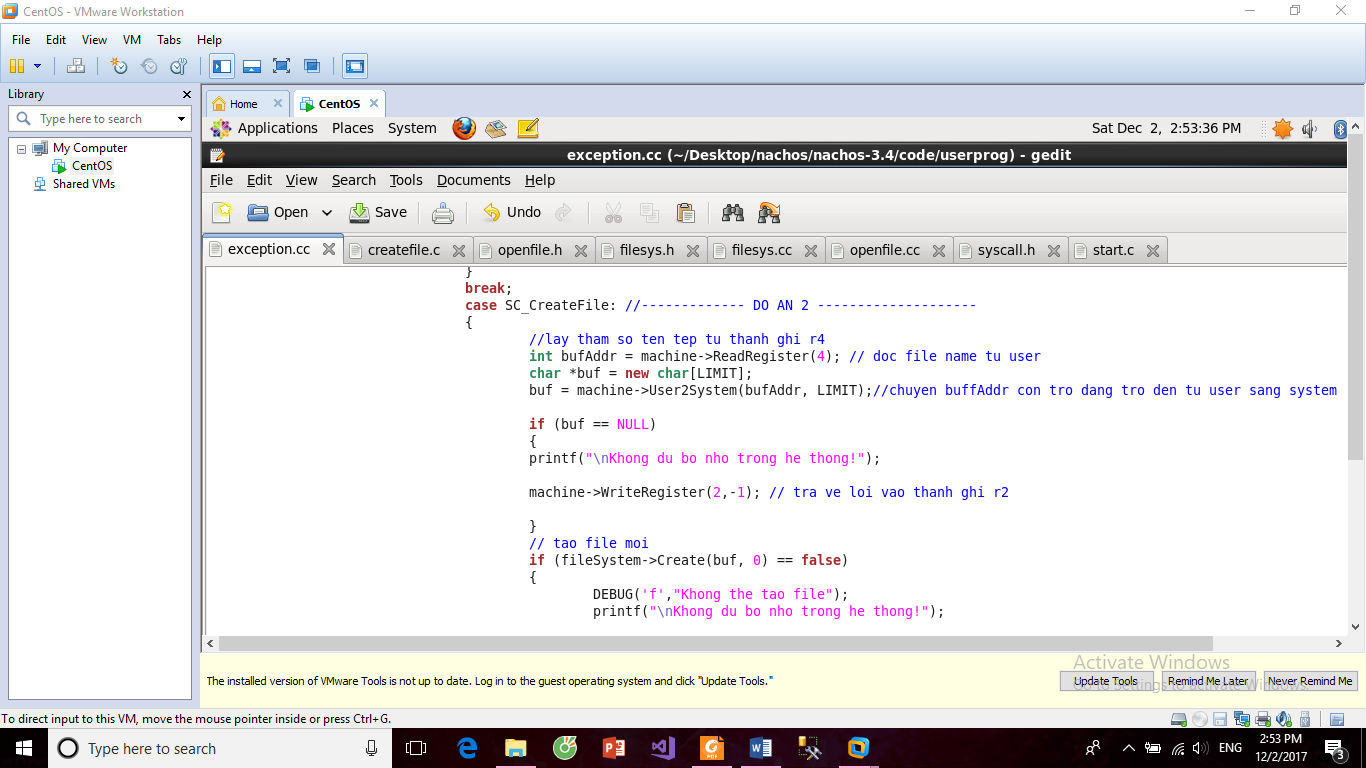
* Sau đó ta khai báo tên hàm của syscall đó



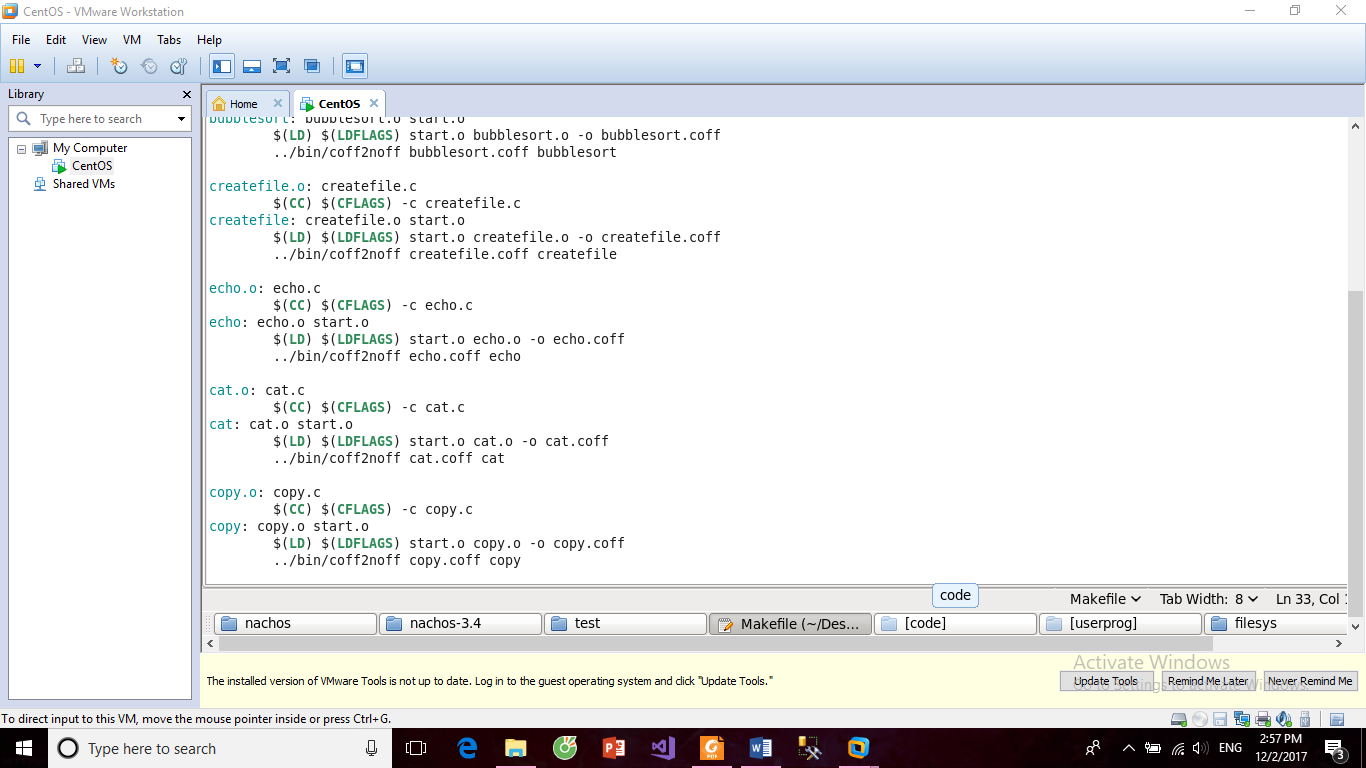
* Bước 2: Thêm vào start.c và start.s trong thư mục *code/test/* phần này để người dùng khi gọi hàm thì nó sẽ nhảy đến các syscall tương ứng



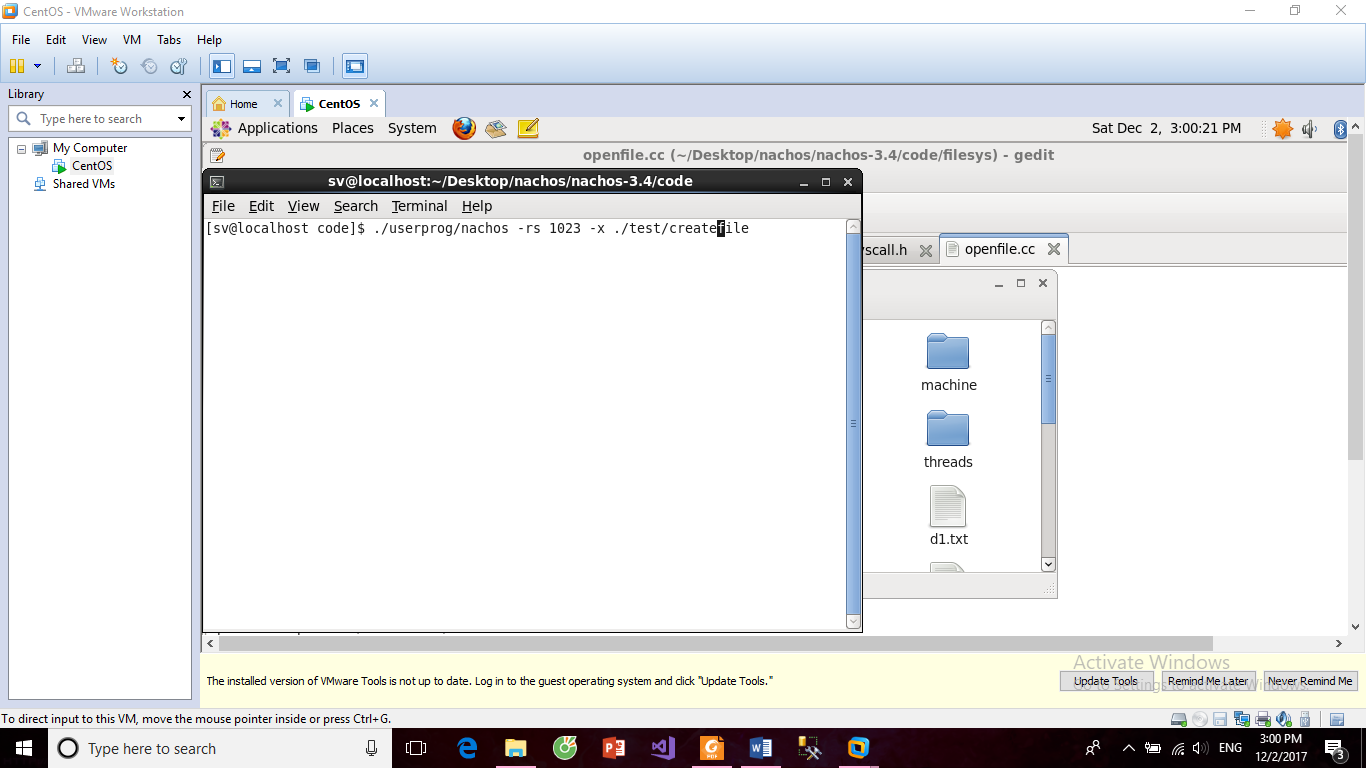
* Bước 3: xử lý syscall trong file *code/userprog/exception.cc*



* 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. Đồ án này, kích thước của bảng mô tả file là 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. Bạn sẽ phải dùng đối tượng filesystem trong thư mục *filesys*. System call Open sẽ trả về id của file (OpenFileID), hoặc là -1 nếu 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 cho mở file đọc và ghi, = 1 cho file chỉ đọc.
* ReadFile and WriteFile Cài đặt system call int ReadFile (char \*buffer, int charcount, OpenFileID id) và int WriteFile (char \*buffer, int charcount, OpenFileID id). Các system call đọc và ghi vào file với id cho trước. Bạn cần phải chuyển vùng nhớ giữa user space và system space, và cần phải phân biệt giữa Console IO (OpenFileID 0, 1) và File. Lệnh ReadFile và WriteFile sẽ làm việc như sau: Phần console read và write, bạn sẽ sử dụng lớp SynchConsole. Được khởi tạo qua biến toàn cục gSynchConsole(bạn phải khai báo biến này). Bạn sẽ sử dụng các hàm mặc định của SynchConsole để đọc và ghi, tuy nhiên bạn phải chịu trách nhiệm trả về đúng giá trị cho user. 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, bạn sẽ sử dụng các lớp được cung cấp trong file system. Sử dụng các hàm mặc định có sẵn của filesystem và thông số trả về cũng phải giống như việc trả về trong synchconsol
* Bước 4: ta viết chương trình để kiểm tra vd như createfile.c
* Bước 5: thêm đoạn vào *Makefile* trong */code/test*/:



* Bước 6: Biên dịch nachos bằng lệnh gmake all và chạy dòng lệnh:



* Cài đặt system call **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.
* Viết chương trình **createfile** để kiểm tra system call CreateFile. Bạn sẽ dùng tên file cố định, hoặc cho người dùng nhập vào từ console nếu như phần console IO của bạn là chạy được.
* Viết chương trình **echo**, mỗi khi nhập một dòng từ console thì console xuất lại dòng đó
* Viết chương trình **cat,** yêu cầu nhập vô filename, rồi hiển thị nội dung của file đó
* Viết chương trình **copy**, yêu cầu nhập tên file nguồn và file đích và thực hiện copy
* Viết chương trình **reverse**, yêu cần nhập tên file nguồn và file đích, đọc nội dung file nguồn, đảo nghịch nội dung và ghi vào file đích

# CÁC CHỨC NĂNG LÀM ĐƯỢC VÀ CHƯA LÀM ĐƯỢC

## Đã làm được

* Hiểu mã chương trình trong nachos
* Hiểu được cấu trúc hệ điều hành
* Cách cài 1 syscall và xử lí các trường hợp ngoại lệ, biết viết chương trình để kiểm tra syscall chạy có đúng hay không
* Thiết kế hệ thống quản lí tập tin

## Chưa làm được

# III. ĐÁNH GIÁ MỨC ĐỘ HOÀN THÀNH ĐỒ ÁN:100%