

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



PROJECT 01 – INTRO XV6
Môn : Hệ Điều Hành

GIẢNG VIÊN HƯỚNG DẪN

Thầy Lê Giang Thanh
Thầy Nguyễn Thanh Quân
Thầy Lê Hà Minh

SINH VIÊN THỰC HIỆN

22127151 - Lâm Tiến Huy
22127290 - Nguyễn Thị Thu Ngân
22127408 – Kha Vĩnh Thuận

LỚP : 22CLC08

Thành Phố Hồ Chí Minh – 2024

Lời Cảm Tạ

Lời đầu tiên, cho phép chúng em được cảm ơn chân thành và sâu sắc nhất muôn gửi đến thầy Nguyễn Thanh Quân vì đã nhiệt tình hướng dẫn chúng em trong cả quá trình làm đồ án này. Ngoài ra, chúng em cũng muốn gửi lời tri ân đến thầy Lê Giang Thanh với công sức và thời gian thầy bỏ ra để phổ cập kiến thức bộ môn Hệ điều hành. Đây sẽ là tiền đề vững chắc cho những năm học sau trong ngành Công nghệ Thông tin.

Đồ án Intro Xv6 lần này là một cơ hội để cho các sinh viên làm quen với việc cài đặt và chạy các hệ điều hành, cụ thể là Linux hay Ubuntu và Xv6, đồng thời thực hiện các thao tác cơ bản và thêm những tính năng vào hệ điều hành. Với Xv6 là một hệ điều hành sử dụng hiệu quả để dạy học và tài liệu của trường đại học MIT, đây là một đồ án giúp cho các sinh viên hiểu rõ bản chất hơn.

Với quá trình làm việc, chúng em cam đoan rằng công việc được phân chia hợp lý, có minh chứng rõ ràng, quá trình làm việc ổn định và linh hoạt. Về kết quả làm việc nói chung, như bảng phân công công việc hay những điều đúc kết được sau đồ án sẽ được ghi chú vào chương cuối cùng của bản báo cáo.

Trong thời gian làm bản cáo cáo, sai sót là điều chúng em không thể tránh khỏi, vì thế chúng em rất mong các thầy xem xét và bỏ qua. Bản báo cáo dựa trên những kiến thức đã được dạy và tự tích luỹ nên cũng có phần thiếu kinh nghiệm thực tiễn, trình độ lí luận, chúng em rất mong nhận được những ý kiến, đóng góp của thầy, cô để chúng em có thể học hỏi và hoàn thiện hơn.

Nếu file gửi qua Moodle có lỗi, các thầy có thể truy cập GitHub của tụi em:
<https://github.com/tntstanHCMUS/xv6-labs-hcmus>

Chúng em xin chân thành cảm ơn.

Mục Lục

Lời Cảm Tạ.....	2
Mục Lục.....	3
Danh Mục Hình	5
Chương 1. Setup	6
1.1 xv6 trên Windows.....	6
1.1.1 Windows Subsystem For Linux	6
1.1.2 Cài đặt xv6 về máy ảo.....	7
1.1.3 Tương tác với Visual Studio Code.....	7
1.1.4 xv6 Booting.....	9
1.2 xv6 trên macOS.....	10
1.2.1 Cài đặt Developer Tools (Xcode)	10
1.2.2 Cài đặt Homebrew.....	10
1.2.3 Cài đặt RISC-V Toolchains.....	10
1.2.4 Cài đặt QEMU	11
1.2.5 xv6 Booting.....	11
Chương 2. Thực Hành	12
2.1 Tạo file và liên kết Makefile	12
2.1.1 Tạo file	12
2.1.2 Liên kết Makefile	12
2.1.3 Kiểm thử	13
2.2 Lập trình chức năng	13
2.2.1 Sleep.....	13
2.2.2 Pingpong.....	14
2.2.3 Primes	14
2.2.4 Find.....	15
2.2.5 Xargs.....	17
2.2.6 Time Spent.....	17
Chương 3. Hoàn Thành	18

3.1 Tổng hợp file.....	18
3.1.1 Tạo diff so sánh kết quả	18
3.1.2 Copy folder xv6 về Windows.....	18
3.2 Chấm điểm	18
Chương 4. Tổng Kết	20
Tài liệu tham khảo	21

Danh Mục Hình

Hình 1. Bật tính năng WSL.....	6
Hình 2. Ubuntu trên Microsoft Store.....	6
Hình 3. Connect to WSL trong Visual Studio Code	8
Hình 4. Kết nối máy ảo thành công.....	8
Hình 5. Thư mục xv6-labs-2023 trên máy Ubuntu.....	9
Hình 6. xv6 Booting trên Ubuntu trên Windows.....	9
Hình 7. Phiên bản đã cài đặt của Homebrew.....	10
Hình 8. Kiểm tra phiên bản của RISC-V Toolchains.....	10
Hình 9. Kiểm tra phiên bản của QEMU	11
Hình 10. xv6 Booting trên macOS.....	11
Hình 11. Tạo file C	12
Hình 12. Liên kết Makefile.....	12
Hình 13. Sleep.....	13
Hình 14. Pingpong.....	14
Hình 15. Primes	15
Hình 16. Pipeline sàng lọc số nguyên tố	15
Hình 17. Find.....	15
Hình 18. Xargs.....	17
Hình 19. Make grade.....	19
Hình 20. Bảng phân công công việc.....	20

Chương 1

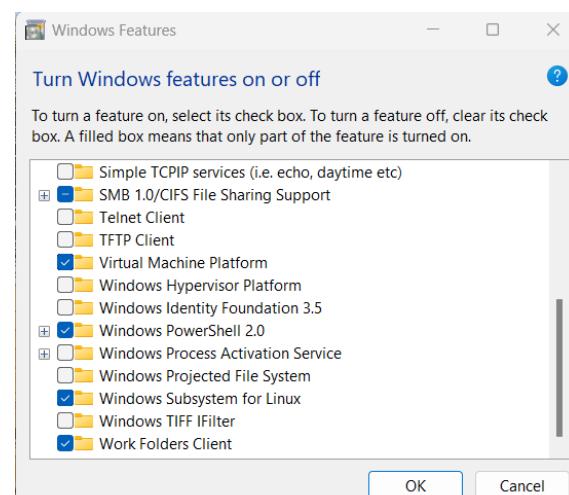
Setup

1.1 xv6 trên Windows

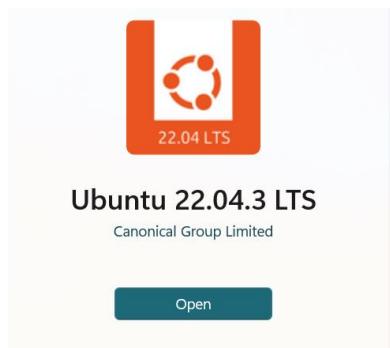
1.1.1 Windows Subsystem For Linux

Trước khi bước vào công việc chính thì việc đầu tiên cần làm đó chính là setup và tạo ra môi trường để có thể hoạt động. Với hai thành viên sử dụng Windows và một thành viên sử dụng macOS, sẽ có sự khác biệt trong việc cài đặt. Sau khi đã cài đặt ở trên máy của từng thành viên, trong chương này, nhóm sẽ chọn demo việc setup trên Windows ở phần này. Chúng ta sẽ sử dụng một số phần mềm được cài đặt sẵn, như Visual Studio Code và Git Bash.

Để enable Windows Subsystem For Linux hay WSL, chúng ta search trên thanh tìm kiếm của Windows là “Turn Windows features on or off” rồi đánh dấu tick vào Virtual Machine Platform và Windows Subsystem for Linux, rồi nhấn OK. Đợi một khoảng thời gian và khởi động lại máy. Điều này sẽ giúp cho chúng ta có thể sử dụng WSL, để bắt đầu cài máy ảo Linux vào máy tính Windows. Nếu WSL chưa có sẵn, sử dụng câu lệnh `wsl --install` hoặc truy cập vào trang web của Windows để hiểu rõ thêm.



Hình 1. Bật tính năng WSL



Hình 2. Ubuntu trên Microsoft Store

Đến lúc này, khi vào CMD, gõ **bash** thì chúng ta sẽ nhận được báo hiệu là chưa có sẵn phiên bản máy ảo nào. Chúng ta sẽ vào Microsoft Store để tiến hành cài đặt hệ điều hành Linux. Ở đây, Ubuntu là một hệ điều hành phát triển dựa trên Linux, và nhóm sẽ tiến hành sử dụng Ubuntu. Khi download Ubuntu về, nếu nhập vào mà chưa chạy được, hãy kiểm tra một số tính năng và chỉnh sửa theo mã lỗi đã báo.

Có một số xu hướng sẽ xảy ra, ví dụ điều chỉnh Virtualization trong BIOS, thì chúng ta sẽ tìm đúng tính năng trong BIOS của các loại máy. Ngoài ra, có thể việc cài đặt bị hoãn do phiên bản đã bị cũ, vì vậy gõ vào CMD câu lệnh **wsl --update**.

Khi máy Ubuntu đã chạy thành công, chúng ta sẽ nhập username và mật khẩu, lưu ý rằng hãy đặt khác với máy Windows để tránh nhầm lẫn. Nếu máy ubuntu chạy thành công, cửa sổ terminal sẽ hiện lên: **<Tên máy ảo>:~\$**. Đến lúc này khi chúng ta cần tương tác với máy ảo, chúng ta có thể bấm thẳng vào ứng dụng **Ubuntu**, hoặc mở CMD và gõ **wsl**.

1.1.2 Cài đặt xv6 về máy ảo

- Từ cửa sổ của máy ảo Ubuntu đã cài đặt, gõ:

```
git clone git://g.csail.mit.edu/xv6-labs-2023
```

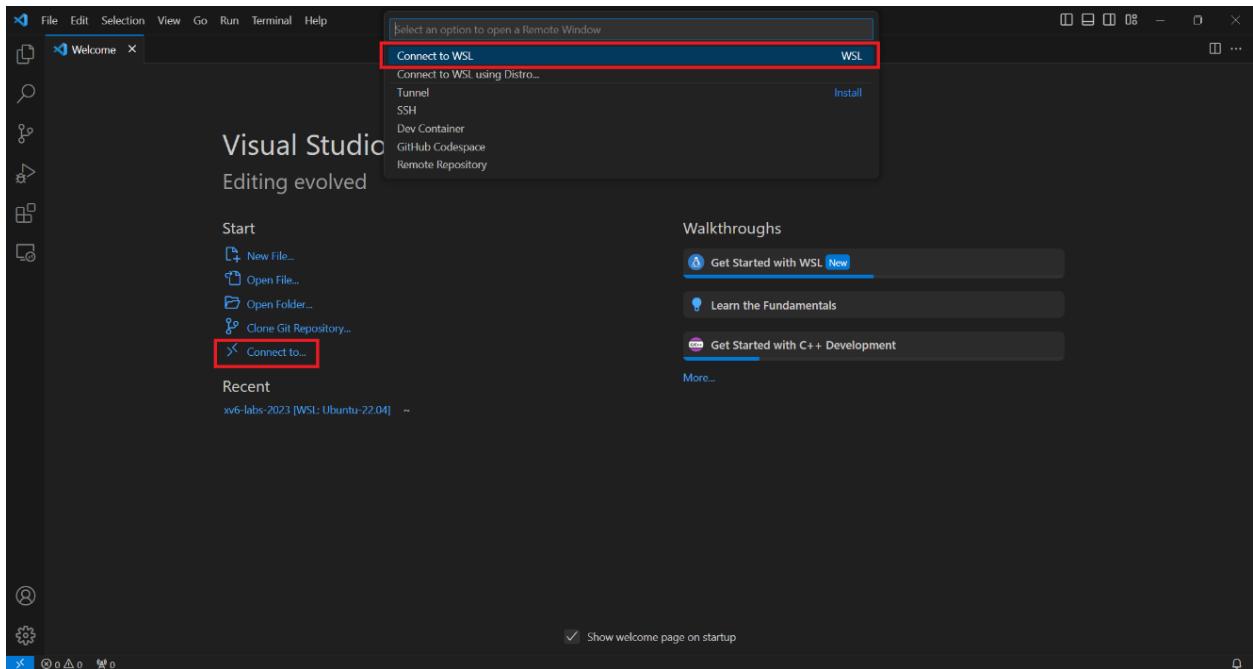
- Cửa sổ sẽ hiển thị : **Cloning into 'xv6-labs-2023'...**

Lúc này chúng ta sẽ có một thư mục **xv6-labs-2023** trong máy ảo. Chúng ta có thể kiểm tra các tập tin và thư mục dùng lệnh **ls**. Lệnh này sẽ hiển thị các tập tin và thư mục ở địa chỉ hiện tại. Lưu ý rằng khi muốn truy cập một thư mục, chúng ta sử dụng **cd <đường dẫn/tên thư mục>** và ở đây, chúng ta sẽ sử dụng thư mục gốc, chỉ cần gõ **cd** chúng ta sẽ về đúng với thư mục đó, đây sẽ là nơi chúng ta lưu trữ xv6.

Để đưa máy lên những tính năng mới nhất, chúng ta sử dụng lệnh **sudo apt-get update** và **sudo apt-get upgrade**. Để cài đặt những công cụ cần thiết, chúng ta sử dụng tiếp lệnh **sudo apt install git build-essential qemu-system**. Thao tác này thường sẽ cài sẵn compiler mà chúng ta sẽ sử dụng là gcc cho ngôn ngữ C. Nếu chưa có gcc, chúng ta có thể dùng lệnh **sudo apt-get install gcc**, kiểm tra phiên bản bằng **gcc -version**. Nếu báo thiếu command make, ta dùng lệnh **sudo apt-get install make**. Nếu trong quá trình làm có lỗi, chúng ta có thể xem sự hướng dẫn của chính terminal của Ubuntu.

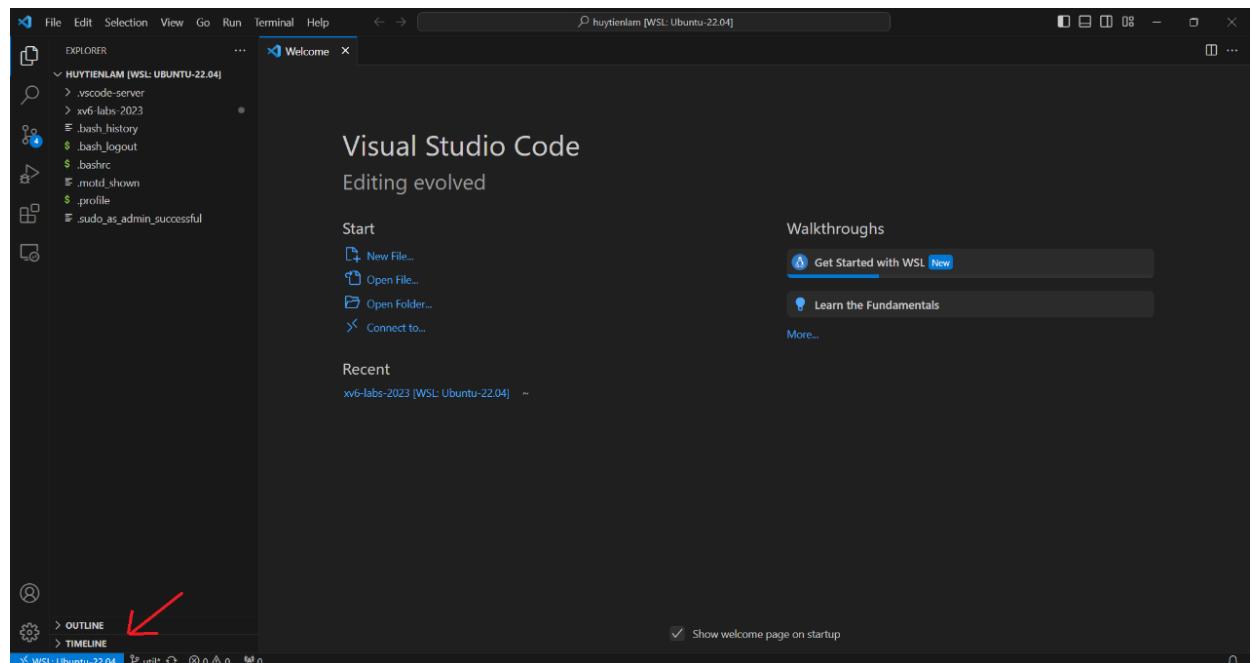
1.1.3 Tương tác với Visual Studio Code

Visual Studio Code cung cấp một môi trường để chúng ta có thể làm việc trên WSL. Chúng ta mở phần mềm, chọn **Connect to...** và chọn **Connect to WSL**. Phần mềm sẽ đưa chúng ta qua cửa sổ của máy ảo.



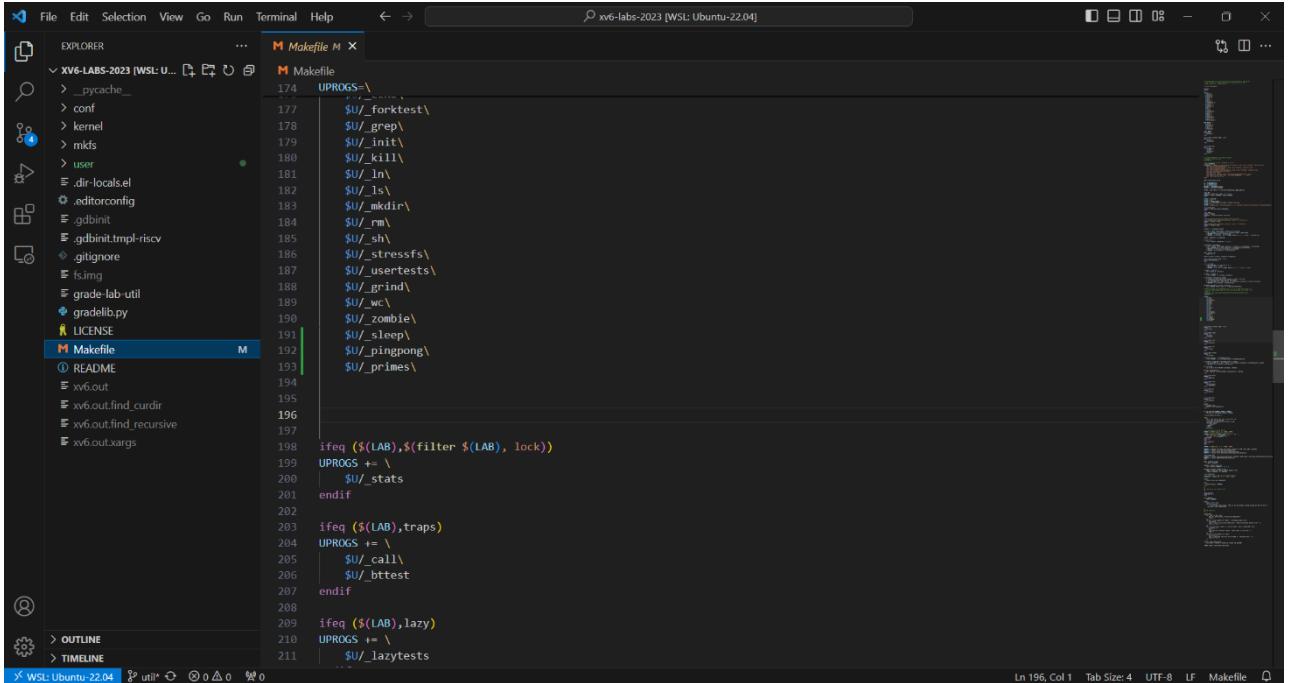
Hình 3. Connect to WSL trong Visual Studio Code

Kết nối thành công khi góc dưới bên trái màn hình hiển thị tên của hệ điều hành.



Hình 4. Kết nối máy ảo thành công

Sau đó, chọn Open Folder, chọn tên máy ảo và thư mục xv6-labs-2023. VSCode sẽ đưa đến thư mục nơi chúng ta lưu trữ xv6. Trong bài tập thực hành này, chúng ta sẽ mở folder user để xem các file và tạo thêm các file mới, là các tính năng mà đề bài yêu cầu. Ngoài ra, Makefile cũng là một file vô cùng quan trọng để liên kết các file trong hệ điều hành với nhau.



Hình 5. Thư mục xv6-labs-2023 trên máy Ubuntu

1.1.4 xv6 Booting

Khi đã hoàn thành, gõ câu lệnh **make qemu**. Nếu câu lệnh thành công, chúng ta sẽ hiện ra màn hình như bên dưới. Nếu chưa thành công, có thể một số thành phần đã bị thiếu, và chúng ta nên làm theo sự hướng dẫn của Ubuntu hoặc lên web để tra tìm các câu lệnh bị thiếu. Khi vào mode **qemu** thì gõ **ls** chúng ta sẽ hiện ra một loạt các dòng chữ. Đây chính là nền tảng để chúng ta chạy những câu lệnh trong bài thực hành này.

```
huytienlam@MSI:~/xv6-labs-2023$ make qemu
qemu-system-riscv64 -machine virt -bios none -kernel kernel/kernel -m 128M -smp 3 -nographic -global virtio-mmio.force-legacy=false -drive file=fs.img,if=none,format=rw,id=x0 -device virtio-blk-device,drive=x0,bus=virtio-mmio-bus.0
xv6 kernel is booting

hart 2 starting
hart 1 starting
init: starting sh
$ ls
.          1 1 1024
..         1 1 1824
README     2 2 2305
xargstest.sh 2 3 93
cat        2 4 32664
echo       2 5 31552
forktest   2 6 15448
grep       2 7 36000
init       2 8 31984
kill       2 9 31536
ln         2 10 31352
ls         2 11 34560
mkdir      2 12 31608
rm         2 13 31600
sh         2 14 53784
stressfs   2 15 32320
usertests  2 16 181712
grind      2 17 47528
wc         2 18 33656
zombie    2 19 38992
sleep     2 20 31352
pingpong   2 21 31696
primes    2 22 32896
console    3 23 0
kaU3MRZb   2 24 0
VPvL8qZf   1 25 64
dd3ET0am   1 29 48
a          1 31 48
c          1 33 48
b          2 35 6
$ |
```

Hình 6. xv6 Booting trên Ubuntu trên Windows

1.2 xv6 trên macOS

Mẫu MacBook sử dụng trong báo cáo lần này mang Chip M1. Chúng ta sẽ tiến hành mở terminal sau đó thực hiện lần lượt các bước cài đặt bên dưới.

1.2.1 Cài đặt Developer Tools (Xcode)

Xcode cung cấp môi trường dòng lệnh để compile chương trình, bỏ qua bước này nếu máy đã được cài sẵn Xcode.

Dùng lệnh: **xcode-select --install**

1.2.2 Cài đặt Homebrew

Homebrew là package manager, cho phép cài đặt, quản lý và cập nhật các package của phần mềm thông qua các dòng lệnh.

Dùng lệnh (trên một dòng): **/bin/bash -c "\$(curl -fsSL**

https://raw.githubusercontent.com/Homebrew/install/HEAD/install.sh)"

Sau đó, chúng ta thực hiện tiếp một số dòng lệnh theo hướng dẫn của nhà phát hành để thêm vào PATH để Homebrew có thể hoạt động bình thường trên shell của máy. Lưu ý kiểm tra đã cài đặt thành công Homebrew bằng lệnh **brew --version**.

```
Last login: Fri Feb 23 16:36:43 on ttys000
helenthungan@Helens-MacBook-Pro ~ % brew --version
Homebrew 4.2.8
helenthungan@Helens-MacBook-Pro ~ %
```

Hình 7. Phiên bản đã cài đặt của Homebrew

1.2.3 Cài đặt RISC-V Toolchains

RISC-V Toolchains cung cấp môi trường và các thư viện để phát triển phần mềm chạy trên kiến trúc RISC-V (như xv6).

Thao tác lần lượt các lệnh: **brew tap riscv/riscv**

brew install riscv-tools

Lưu ý kiểm tra đã cài đặt thành công bằng lệnh

riscv64-unknown-elf-gcc --version

```
Last login: Sat Feb 24 02:10:41 on ttys000
helenthungan@Helens-MacBook-Pro ~ % riscv64-unknown-elf-gcc --version
riscv64-unknown-elf-gcc (gc891d8dc2) 13.2.0
Copyright (C) 2023 Free Software Foundation, Inc.
This is free software; see the source for copying conditions. There is NO
warranty; not even for MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.
```

Hình 8. Kiểm tra phiên bản của RISC-V Toolchains

1.2.4 Cài đặt QEMU

QEMU là công cụ giả lập phần cứng cho phép chạy các chương trình hệ điều hành và phần mềm trên các kiến trúc máy tính khác nhau.

Dùng lệnh: **brew install qemu**

Lưu ý kiểm tra đã cài đặt thành công bằng lệnh **qemu-system-riscv64 --version**

```
Last login: Sat Feb 24 02:12:29 on ttys000
helenthungan@Helens-MacBook-Pro ~ % qemu-system-riscv64 --version
QEMU emulator version 8.2.1
Copyright (c) 2003-2023 Fabrice Bellard and the QEMU Project developers
helenthungan@Helens-MacBook-Pro ~ %
```

Hình 9. Kiểm tra phiên bản của QEMU

1.2.5 xv6 Booting

```
helenthungan@Helens-MacBook-Pro ~ % cd DESKTOP/Os/XV6
helenthungan@Helens-MacBook-Pro XV6 % cd xv6-labs-283
MacBook-Pro:~/xv6-labs-283 helenthungan % ./xv6
riscv64-unknown-elf-gcc -o kernel/entry.o kernel/entry.S
riscv64-unknown-elf-gcc -o kernel/entry,0 kernel/entry.S
riscv64-unknown-elf-gcc -Wall -Werror -O -fno-omit-frame-pointer -ggdb -gdwarf-2 -DSOL_UTIL -DLAB_UTIL -MD -mcmode=medany -ffreestanding -fno-common -nostdlib -mno-relax -I. -fno-stack-protector -fno-pie -no-pie
-c -O kernel/kalloc.o kernel/kalloc.c
riscv64-unknown-elf-gcc -Wall -Werror -O -fno-omit-frame-pointer -ggdb -gdwarf-2 -DSOL_UTIL -DLAB_UTIL -MD -mcmode=medany -ffreestanding -fno-common -nostdlib -mno-relax -I. -fno-stack-protector -fno-pie -no-pie
-o kernel/string.o kernel/string.c
riscv64-unknown-elf-gcc -Wall -Werror -O -fno-omit-frame-pointer -ggdb -gdwarf-2 -DSOL_UTIL -DLAB_UTIL -MD -mcmode=medany -ffreestanding -fno-common -nostdlib -mno-relax -I. -fno-stack-protector -fno-pie -no-pie
-o kernel/main.o kernel/main.c
riscv64-unknown-elf-gcc -Wall -Werror -O -fno-omit-frame-pointer -ggdb -gdwarf-2 -DSOL_UTIL -DLAB_UTIL -MD -mcmode=medany -ffreestanding -fno-common -nostdlib -mno-relax -I. -fno-stack-protector -fno-pie -no-pie
-c -O kernel/vm.o kernel/vm.c
riscv64-unknown-elf-gcc -Wall -Werror -O -fno-omit-frame-pointer -ggdb -gdwarf-2 -DSOL_UTIL -DLAB_UTIL -MD -mcmode=medany -ffreestanding -fno-common -nostdlib -mno-relax -I. -fno-stack-protector -fno-pie -no-pie
-c -O kernel/proc.o kernel/proc.c
riscv64-unknown-elf-gcc -c -O kernel/swtch.o kernel/swtch.S
riscv64-unknown-elf-gcc -c -O kernel/trampoline.o kernel/trampoline.S
riscv64-unknown-elf-gcc -Wall -Werror -O -fno-omit-frame-pointer -ggdb -gdwarf-2 -DSOL_UTIL -DLAB_UTIL -MD -mcmode=medany -ffreestanding -fno-common -nostdlib -mno-relax -I. -fno-stack-protector -fno-pie -no-pie
-o kernel/trap.o kernel/trap.c
riscv64-unknown-elf-gcc -Wall -Werror -O -fno-omit-frame-pointer -ggdb -gdwarf-2 -DSOL_UTIL -DLAB_UTIL -MD -mcmode=medany -ffreestanding -fno-common -nostdlib -mno-relax -I. -fno-stack-protector -fno-pie -no-pie
-c -O kernel/syscall.o kernel/syscall.c
riscv64-unknown-elf-gcc -Wall -Werror -O -fno-omit-frame-pointer -ggdb -gdwarf-2 -DSOL_UTIL -DLAB_UTIL -MD -mcmode=medany -ffreestanding -fno-common -nostdlib -mno-relax -I. -fno-stack-protector -fno-pie -no-pie
-c -O kernel/sysproc.o kernel/sysproc.c
riscv64-unknown-elf-gcc -Wall -Werror -O -fno-omit-frame-pointer -ggdb -gdwarf-2 -DSOL_UTIL -DLAB_UTIL -MD -mcmode=medany -ffreestanding -fno-common -nostdlib -mno-relax -I. -fno-stack-protector -fno-pie -no-pie
-o kernel/sleep.o kernel/sleep.c
riscv64-unknown-elf-gcc -Wall -Werror -O -fno-omit-frame-pointer -ggdb -gdwarf-2 -DSOL_UTIL -DLAB_UTIL -MD -mcmode=medany -ffreestanding -fno-common -nostdlib -mno-relax -I. -fno-stack-protector -fno-pie -no-pie
-o kernel/sleeplock.o kernel/sleeplock.c
riscv64-unknown-elf-gcc -Wall -Werror -O -fno-omit-frame-pointer -ggdb -gdwarf-2 -DSOL_UTIL -DLAB_UTIL -MD -mcmode=medany -ffreestanding -fno-common -nostdlib -mno-relax -I. -fno-stack-protector -fno-pie -no-pie
-o kernel/sleepfile.o kernel/sleepfile.c
riscv64-unknown-elf-gcc -Wall -Werror -O -fno-omit-frame-pointer -ggdb -gdwarf-2 -DSOL_UTIL -DLAB_UTIL -MD -mcmode=medany -ffreestanding -fno-common -nostdlib -mno-relax -I. -fno-stack-protector -fno-pie -no-pie
-c -O kernel/sleep.o kernel/sleep.c
riscv64-unknown-elf-gcc -Wall -Werror -O -fno-omit-frame-pointer -ggdb -gdwarf-2 -DSOL_UTIL -DLAB_UTIL -MD -mcmode=medany -ffreestanding -fno-common -nostdlib -mno-relax -I. -fno-stack-protector -fno-pie -no-pie
-o kernel/exec.o kernel/exec.c
riscv64-unknown-elf-objdump | sed '1,/SYMBOL TABLE/d; s/* / /; /*$/d' > user/stressfs.sym
riscv64-unknown-elf-gcc -Wall -Werror -O -fno-omit-frame-pointer -ggdb -gdwarf-2 -DSOL_UTIL -DLAB_UTIL -MD -mcmode=medany -ffreestanding -fno-common -nostdlib -mno-relax -I. -fno-stack-protector -fno-pie -no-pie
-c -O user/usertests.o user/usertests.c
riscv64-unknown-elf-objdump -z max-page-size=4096 -t user/user.Id > user/_usertests/user/usertests.o user/ulib.o user/usys.o user/printf.o user/umalloc.o
riscv64-unknown-elf-objdump -t user/_usertests | sed '1,/SYMBOL TABLE/d; s/* / /; /*$/d' > user/usertests.sym
riscv64-unknown-elf-gcc -Wall -Werror -O -fno-omit-frame-pointer -ggdb -gdwarf-2 -DSOL_UTIL -DLAB_UTIL -MD -mcmode=medany -ffreestanding -fno-common -nostdlib -mno-relax -I. -fno-stack-protector -fno-pie -no-pie
-c -O user/_usertests.o user/_usertests.c
riscv64-unknown-elf-objdump -t user/_usertests | sed '1,/SYMBOL TABLE/d; s/* / /; /*$/d' > user/_usertests.sym
riscv64-unknown-elf-gcc -Wall -Werror -O -fno-omit-frame-pointer -ggdb -gdwarf-2 -DSOL_UTIL -DLAB_UTIL -MD -mcmode=medany -ffreestanding -fno-common -nostdlib -mno-relax -I. -fno-stack-protector -fno-pie -no-pie
-c -O user/_usertests.o user/_usertests.c
riscv64-unknown-elf-objdump -t user/_usertests | sed '1,/SYMBOL TABLE/d; s/* / /; /*$/d' > user/_usertests.sym
riscv64-unknown-elf-gcc -Wall -Werror -O -fno-omit-frame-pointer -ggdb -gdwarf-2 -DSOL_UTIL -DLAB_UTIL -MD -mcmode=medany -ffreestanding -fno-common -nostdlib -mno-relax -I. -fno-stack-protector -fno-pie -no-pie
-c -O user/_usertests.o user/_usertests.c
riscv64-unknown-elf-objdump -t user/_wc > user/_wc.asm
riscv64-unknown-elf-objdump -t user/_wc | sed '1,/SYMBOL TABLE/d; s/* / /; /*$/d' > user/_wc.sym
riscv64-unknown-elf-gcc -Wall -Werror -O -fno-omit-frame-pointer -ggdb -gdwarf-2 -DSOL_UTIL -DLAB_UTIL -MD -mcmode=medany -ffreestanding -fno-common -nostdlib -mno-relax -I. -fno-stack-protector -fno-pie -no-pie
-c -O user/_wc.o user/_wc.c
riscv64-unknown-elf-objdump -t user/_wc | sed '1,/SYMBOL TABLE/d; s/* / /; /*$/d' > user/_wc.sym
riscv64-unknown-elf-gcc -Wall -Werror -O -fno-omit-frame-pointer -ggdb -gdwarf-2 -DSOL_UTIL -DLAB_UTIL -MD -mcmode=medany -ffreestanding -fno-common -nostdlib -mno-relax -I. -fno-stack-protector -fno-pie -no-pie
-c -O user/_wc.o user/_wc.c
riscv64-unknown-elf-objdump -t user/_zombie > user/_zombie.asm
riscv64-unknown-elf-objdump -S user/_zombie | sed '1,/SYMBOL TABLE/d; s/* / /; /*$/d' > user/_zombie.sym
riscv64-unknown-elf-gcc -Wall -Werror -O -fno-omit-frame-pointer -ggdb -gdwarf-2 -DSOL_UTIL -DLAB_UTIL -MD -mcmode=medany -ffreestanding -fno-common -nostdlib -mno-relax -I. -fno-stack-protector -fno-pie -no-pie
-c -O user/_zombie.o user/_zombie.c
riscv64-unknown-elf-objdump -t user/_forktest > user/_forktest.asm
riscv64-unknown-elf-objdump -S user/_forktest | sed '1,/SYMBOL TABLE/d; s/* / /; /*$/d' > user/_forktest.sym
mkfs/mkfs fs.img README user/xv6test.sh user/_cat user/_echo user/_forktest user/_grep user/_init user/_kill user/_ln user/_ls user/_mkdir user/_rm user/_sh user/_stressfs user/_usertests user/_grind user/_wc user/_zombie
netcat 46 (boot, super, log blocks 38 inode blocks 13, bitmap blocks 1) blocks 1954 total 2000
blocks 1954 bitmap blocks have been allocated
balloc: write bitmap block at sector 45
qemu-system-riscv64 -machine virt -bios none -kernel kernel/kernel -m 128M -smp 3 -nographic -global virtio-mmio.force-legacy=false -drive file=fs.img,if=none,format=raw,id=x0 -device virtio-blk-device,drive=x0,bus=virtio-mmio-bus.8

xv6 kernel booting
hart 2 starting
hart 1 starting
init: starting sh
$ ls
.. 1 1 1024
. 1 1 1024
README 2 2 2385
xv6test.sh 2 2 93
cat 2 4 33884
echo 2 5 32344
forktest 2 6 16640
grep 2 7 36944
init 2 8 32792
kill 2 9 32288
ln 2 10 32088
ls 2 11 36690
mkdir 2 12 32344
rm 2 13 32320
sh 2 14 55072
stressfs 2 15 33192
usertests 2 16 182600
grind 2 17 48750
wc 2 18 47472
zombie 2 19 31664
console 3 20 0
$
```

Hình 10. xv6 Booting trên macOS

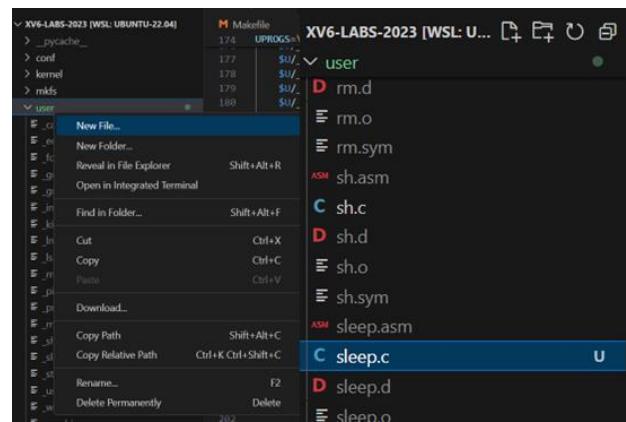
Chương 2

Thực Hành

2.1 Tạo file và liên kết Makefile

2.1.1 Tao file

Trong cửa sổ VSCode, thư mục user, chúng ta có thể thực hiện thủ công bằng cách chuột phải chọn New File. Đặt tên file dưới định dạng **<Tên file>.c** và bắt đầu thực hiện lập trình trong file. Lưu ý rằng sau mỗi lần chỉnh sửa thì phải Save lại để chương trình cập nhật phiên bản mới và có thể hoạt động bình thường.



Hình 11. Tao file C

2.1.2 Liên kết Makefile

```
File Edit Selection View Go Run Terminal Help ← → xv6-labs 2023 [WSL: Ubuntu-22.04] Makefile M x M Makefile 1/0 # details: 171 # http://www.gnu.org/software/make/manual/html_node/Chained-Rules.html 172 .PRECIOUS: %.* 173 174 UPROGS=\ 175 $U/_cat\ 176 $U/_echo\ 177 $U/_forktest\ 178 $U/_prep\ 179 $U/_init\ 180 $U/_kill\ 181 $U/_ln\ 182 $U/_ls\ 183 $U/_mkdir\ 184 $U/_rm\ 185 $U/_sh\ 186 $U/_stressfs\ 187 $U/_usertests\ 188 $U/_grind\ 189 $U/_wc\ 190 $U/_zombie\ 191 $U/_sleep\ 192 $U/_pingpong\ 193 $U/_primes\ 194 195 196 197 198 ifeq ($(LAB),$(filter $(LAB), lock)) 199 UPROGS += \ 200 $U/_stats 201 endif 202 203 ifeq ($(LAB),traps) 204 UPROGS += \ 205 $U/_call\ 206 $U/_bttest
```

Hình 12. Liên kết Makefile

Quay lại thư mục **xv6-labs-2023** và mở Makefile. Từ đó, kéo xuống tới dòng **UPROGS=\.**. Từ đó, chúng ta sẽ tuân thủ đúng định dạng của những file được cài sẵn ở phía trên, đúng định dạng tương ứng với các file C mang tên các chức năng, là **\$U/_<Tên file>\.**. Từ chức năng cuối cùng, thường là **zombie**, chúng ta sẽ thêm chức năng ngay phía dưới theo đúng định dạng như trên. Lưu ý rằng, chúng ta phải gõ chính xác tên file để có thể liên kết thành công. Sau đó, lưu lại Makefile và hoàn thành việc liên kết.

2.1.3 Kiểm thử

Sau mỗi quá trình tạo ra các chức năng mới, chúng ta sẽ tiến hành liên kết Makefile. Nếu các file được liên kết thành công và không có lỗi xảy ra trong quá trình lập trình, thì khi chúng ta thực hiện **make qemu** trong **xv6**, trình biên dịch sẽ không báo lỗi và chế độ **qemu** sẽ được thực hiện. Trong chế độ **qemu**, chúng ta sẽ gõ chính xác tên mà chúng ta đặt cho file C của tính năng, và chúng ta có thể kiểm tra được liệu tính năng đó có hoạt động đúng như mong muốn không. Nếu chưa chính xác, chúng ta quay lại file C để sửa lỗi và tiếp tục chạy lại quá trình kiểm thử.

Trong phần giải thích lập trình các chức năng, chúng ta sẽ chỉ đi qua cách hoạt động và logic lập trình, không trình bày lại code và giải thích từng dòng mà sẽ giải thích chung. Các tính năng cũng sẽ chạy dùng command-line.

2.2 Lập trình chức năng

2.2.1 Sleep

Sleep là chức năng dừng hoạt động hệ điều hành trong một số lượng **tích tắc** do người dùng chỉ định. **Tích tắc** là một khái niệm về thời gian được xác định bởi **xv6 kernel**, cụ thể là thời gian giữa hai lần ngắn từ chip hẹn giờ. Tệp ‘**sleep.c**’ sẽ được lưu trong folder **user**.

```
○ huytienlam@MSI:~/xv6-labs-2023$ make qemu
qemu-system-riscv64 -machine virt -bios none -kernel kernel/kernel -m 128M -smp 3 -nographic -global virtio-mmio.force-legacy=false -drive file=fs.img,if=none
,format=raw,id=x0 -device virtio-blk-device,drive=x0,bus=virtio-mmio-bus.0

xv6 kernel is booting

hart 2 starting
hart 1 starting
init: starting sh
$ sleep 10
Done!
$
```

Hình 13. Sleep

Chương trình sẽ thu thập số lượng **argument** từ **input** của người dùng. Nếu người dùng sai cú pháp, chương trình sẽ nhắc người dùng về cú pháp chuẩn, và **exit** với giá trị 1. Nếu đúng cú pháp, input con số của người dùng sẽ được chuyển thành dạng số nguyên và đưa vào hàm **sleep**, một hàm được cài sẵn trong **xv6 kernel** trong thư viện **user/user.h**. Hàm

sleep thực hiện đúng nhiệm vụ ban đầu đề ra, dừng trong một khoảng tích tắc nhất định. Sau khi hoàn thành, chương trình sẽ in tín hiệu hoàn thành và exit với giá trị 0.

2.2.2 Pingpong

Pingpong là chức năng dùng các lệnh system call của xv6 để “ping-pong” một byte giữa hai process thông qua một pipe có hai hướng. Tiến trình cha sẽ gửi một byte cho tiến trình con, và con phải in ra <pid>: received ping, với pid ở đây là process ID, ghi byte bên phía chuyển tới cha rồi thoát. Cha sẽ đọc byte mà con gửi rồi in <pid>: received pong, và thoát. ‘pingpong.c’ sẽ được lưu trong folder user.

```
huytienlam@MST:~/xv6-labs-2023$ make qemu
qemu-system-riscv64 -machine virt -bios none -kernel kernel/kernel -m 128M -smp 3 -nographic -global virtio-mmio.force-legacy=false -drive file=fs.img,if=none
,format=raw,id=x0 -device virtio-blk-device,drive=x0,bus=virtio-mmio-bus.0

xv6 kernel is booting

hart 2 starting
hart 1 starting
init: starting sh
$ pingpong
4: received ping
3: received pong
$
```

Hình 14. Pingpong

Hàm pipe được gọi để tạo một pipe, tạo ra một kênh truyền thông một chiều giữa hai tiến trình. Hai con số biểu thị đầu đọc và đầu ghi của ống lưu trữ trong mảng p. Hàm fork được gọi để tạo một tiến trình con. Giá trị trả về của fork được lưu trong biến pid. Nếu pid = 0, đó là tiến trình con. Nếu pid > 0, đó là tiến trình cha. Nếu pid < 0, đó là một lỗi trong quá trình tạo tiến trình.

Nếu pid = 0, đó là tiến trình con. Nó đóng đầu đọc của ống, ghi chuỗi ping vào đầu ghi của ống, và in ra một thông báo cho biết rằng nó đã nhận được ping. Nếu pid > 0, đó là tiến trình cha. Nó chờ đợi tiến trình con kết thúc, đóng đầu ghi của ống, đọc dữ liệu từ đầu đọc của ống vào bộ đệm buf, và in ra một thông báo cho biết rằng nó đã nhận được pong. Nếu fork trả về < 0, có nghĩa là có lỗi xảy ra trong quá trình tạo tiến trình, chương trình sẽ báo lỗi và exit với 1. Nếu quá trình diễn ra thành công, chương trình exit với 0.

2.2.3 Primes

Primes là chức năng sàng số nguyên tố đồng thời cho xv6 sử dụng các pipe và fork để thiết lập đường ống. Tiến trình đầu tiên sẽ đưa 2 đến 35 vào đường ống. Với mỗi số nguyên tố, chúng ta sẽ sắp xếp để tạo ra một tiến trình đọc từ tiến trình bên trái thông qua một đầu và ghi vào tiến trình bên phải thông qua đầu còn lại. Vì xv6 có số lượng hạn chế về các mô tả file và tiến trình, tiến trình đầu tiên có thể dừng lại ở số 35. ‘primes.c’ sẽ được lưu trong folder user.

```

huytienlam@MSI:~/xv6-labs-2023$ make qemu
qemu-system-riscv64 -machine virt -bios none -kernel kernel/kernel -m 128M -smp 3 -nographic -global virtio-mmio.force-legacy=false -drive file=fs.img,if=none
,format=raw,id=x0 -device virtio-blk-device,drive=x0,bus=virtio-mmio-bus.0

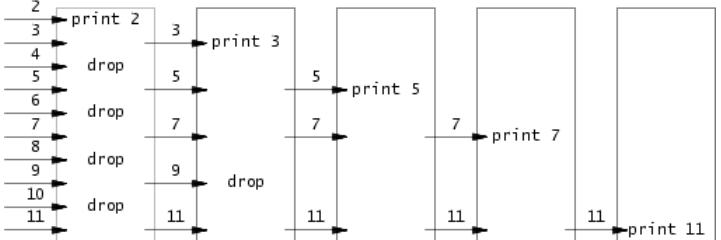
xv6 kernel is booting

hart 2 starting
hart 1 starting
init: starting sh
$ primes
prime 2
prime 3
prime 5
prime 7
prime 11
prime 13
prime 17
prime 19
prime 23
prime 29
prime 31
$ 

```

Hình 15. Primes

Hàm `exec_pipe` sử dụng để sàng lọc số nguyên tố và chuyển tiếp các số nguyên tố cho các tiến trình con để tiếp tục xử lý. Nó đọc một số nguyên từ pipe sử dụng hàm `read`. Số nguyên này sẽ là số nguyên tố đầu tiên được tìm thấy, lưu vào biến `num`. Sau đó, in ra màn hình thông báo rằng số đó là một số nguyên tố. Tiếp theo, dùng hàm `pipe` để tạo ra pipe mới để truyền tiếp các số thỏa điều kiện cho các tiến trình con. Hàm này sử dụng một vòng lặp vô hạn để đọc từng số từ ống tới khi không còn số nào. Nếu số được đọc từ ống không chia được cho số nguyên tố được lưu vào biến `num` trước đó, số đó được gửi qua ống mới bằng hàm `write`. Khi không còn số nào trong ống, giá trị biến tạm là `-1` biểu thị kết thúc, các ống và file descriptor được đóng và hàm kết thúc.



Hình 16. Pipeline sàng lọc số nguyên tố

Hàm `main` tạo ra một pipe để truyền dữ liệu giữa các tiến trình. Vòng lặp `for` được sử dụng để viết các số từ 2 đến 34 vào đầu ghi của ống sử dụng hàm `write`. Sau khi viết xong, đầu ghi của ống sẽ được đóng. Tiếp theo, `exec_pipe` sàng nguyên tố trên các số từ 2 đến 34. Cuối cùng, đầu đọc của ống cũng được đóng và chương trình exit với 0.

2.2.4 Find

`Find` là phiên bản đơn giản của chương trình UNIX `find` cho `xv6`, tìm tất cả các tệp trong cây thư mục có một tên cụ thể. ‘`find.c`’ sẽ được lưu trong folder `user`.

```

huytienlam@MSI:~/xv6-labs-2023$ make qemu
qemu-system-riscv64 -machine virt -bios none -kernel kernel/kernel -m 128M -smp 3 -nographic -global virtio-mmio.force-legacy=false -drive file=fs.img,if=none
,format=raw,id=x0 -device virtio-blk-device,drive=x0,bus=virtio-mmio-bus.0

xv6 kernel is booting

hart 1 starting
hart 2 starting
init: starting sh
$ echo > b
$ mkdir a
$ echo > a/b
$ mkdir a/aa
$ echo > a/aa/b
$ find . b
./b

```

Hình 17. Find

Hàm `fmtname` nhận một **đường dẫn** và trả về **tên tệp** (phần cuối cùng của đường dẫn). Biến `buf` là **mảng kí tự** có kích thước đủ để lưu tên tệp, còn con trỏ `p` là **kí tự** dùng duyệt qua đường dẫn. Vòng lặp `for` sẽ đưa `p` tìm tới kí tự đầu tiên sau dấu gạch chéo cuối cùng. Nếu tên tệp **dài hơn hoặc bằng DIRSIZ**, hàm sẽ trực tiếp trả về từ `p` đến khi không lưu trữ được thêm kí tự. Nếu tên tệp **ngắn hơn DIRSIZ**, hàm `memmove` sẽ dùng để sao chép tên tệp vào mảng `buf`. Hàm `memset` được sử dụng để điền kí tự trắng vào những vị trí còn trống của `buf` sau phần chứa tên tệp để cho đủ số lượng kí tự. Cuối cùng, hàm trả về một **pointer** đến `buf`, chứa tên của tệp đã được định dạng và **lấy kí tự trắng** nếu có.

Hàm `strcmp_blank` sử dụng để so sánh hai chuỗi kí tự `buf` và `file_name`. Hàm thực hiện so sánh **kí tự** của từng vị trí trong hai chuỗi kí tự. Nếu có bất kỳ sự khác biệt nào giữa các kí tự ở cùng một vị trí, hàm sẽ trả về giá trị 1, ngược ý hai chuỗi **không giống nhau**. Nếu các kí tự trong hai chuỗi đều giống nhau, hàm sẽ trả về 0, ngược ý rằng hai chuỗi **giống nhau**.

Hàm `find` này duyệt qua tất cả các tệp và thư mục trong đường dẫn được chỉ định và tìm kiếm các tệp có tên cụ thể. Như trước, `buf` là **mảng kí tự** dùng để lưu đường dẫn, `p` là **pointer** thao tác trên mảng `buf`, `fd` là **file descriptor** để mở và đọc từ thư mục. Biến `de` là `dirent` để **lưu thông tin** về mỗi tệp hoặc thư mục trong thư mục hiện tại, và biến `st` là `stat` để **lưu thông tin** về **trạng thái** của tệp hoặc thư mục. Hàm `open` **mở thư mục** được chỉ định bởi `path` để đọc. Nếu việc mở thư mục gặp lỗi và trả về -1, báo lỗi và **kết thúc** hàm. Hàm `fstat` lấy thông tin về **trạng thái** của thư mục và **lưu** vào biến `st`. Nếu việc lấy thông tin gặp lỗi và trả về -1, báo lỗi, **đóng** file descriptor và **kết thúc** hàm.

Tiếp theo, nếu `st.type` là `T_FILE`, tức là `path` trả tới **một tệp**, kiểm tra xem tên của tệp này có **trùng với** `file_name` hay không và **in đường dẫn** nếu có. Nếu `st.type` là `T_DIR`, tức là `path` trả tới một **thư mục** thì **tiếp tục** xử lý. Chúng ta sẽ tiến hành **kiểm tra** nếu độ dài của **đường dẫn**, **tên tệp** và **dấu /** **vượt quá** kích thước của `buf`. Nếu vượt quá, báo lỗi, **đóng** file descriptor và **kết thúc** hàm.

Tiếp theo, dùng vòng lặp `while` để **đọc thông tin** của từng tệp và thư mục trong thư mục hiện tại bằng cách sử dụng hàm `read`. Nếu `inum` của `de` bằng 0, tức là **tệp** hoặc **thư mục** này **không hợp lệ**, tiếp tục vòng lặp để xử lý tiếp. Chúng ta sao chép tên của tệp hoặc thư mục vào `buf` bằng cách sử dụng hàm `memmove`, sau đó đảm bảo **kết thúc** chuỗi bằng cách **gán** kí tự `NULL`. Nếu tên của tệp hoặc thư mục **trùng với** `file_name`, **in ra** **đường dẫn** của **tệp** hoặc **thư mục** đó. Nếu tên là `.` hoặc `..`, **tiếp tục** vòng lặp. Nếu `st.type` tiếp tục là `T_DIR` là một **thư mục**, thực hiện **đệ quy** gọi hàm `find` lại trên thư mục này để tìm kiếm các tệp trong đó. **Đóng** file descriptor sau khi đã hoàn thành xử lý các tệp và thư mục trong thư mục hiện tại.

Tới hàm **main**, nếu số argument < 3, chương trình sẽ báo lỗi cần thêm argument và thoát. Vòng lặp for duyệt qua từ đối số thứ hai đến đối số cuối cùng. Trong mỗi vòng lặp, hàm **find** được gọi với đối số đầu tiên là đường dẫn thư mục cần tìm kiếm và đối số thứ hai là tên tệp cần tìm trong thư mục. Chương trình hoàn thành và exit với 0.

2.2.5 Xargs

Xargs là một phiên bản đơn giản của chương trình UNIX **xargs** cho **xv6**, với các argument của nó mô tả một lệnh cần chạy, đọc các dòng từ **input** và chạy lệnh cho mỗi dòng, nối dòng vào các argument. ‘**xargs.c**’ sẽ được lưu trong folder user.

```
○ huytienlam@MSI-:~/xv6-labs-2023$ make qemu
qemu-system-riscv64 -machine virt -bios none -kernel kernel/kernel -m 128M -smp 3 -nographic -global virtio-mmio.force-legacy=false -drive file=fs.img,if=none
,format=raw,id=x0 -device virtio-blk-device,drive=x0,bus=virtio-mmio-bus.0

xv6 kernel is booting

hart 2 starting
hart 1 starting
init: starting sh
$ echo hello too | xargs echo bye
bye hello too
$ sh < xargstest.sh
$ $ $ $ $ hello
hello
hello
$ $ |
```

Hình 18. Xargs

Chương trình bắt đầu bằng việc khai báo mảng hai chiều **args** để lưu trữ các argument, mảng con trỏ ***temp_args** để truyền các argument cho lệnh **exec**, các biến **i**, **j**, **len** để điều khiển vòng lặp. Hàm **memset()** để khởi tạo các phần tử trong mảng **args** bằng 0. Vòng lặp **for** sao chép các argument từ dòng lệnh vào mảng **args**, đảm bảo các argument được cung cấp khi gọi chương trình sẽ được lưu trữ để sử dụng sau này.

Trong vòng lặp **while**, chương trình đọc dữ liệu từ input chuẩn bằng cách sử dụng hàm **read()**, kết thúc khi không còn dữ liệu hoặc khi gặp kí tự xuống dòng. Khi gặp kí tự kết thúc dòng, chương trình kết thúc một argument bằng cách đặt kí tự kết thúc chuỗi **\0** vào vị trí cuối cùng trong mảng **args**. Sau đó, nó sao chép các argument từ **args** vào **temp_args** và thực thi hàm **exec**. Tiền trinh cha sẽ chờ tiền trinh con hoàn thành bằng hàm **wait()**. Nếu chương trình gặp kí tự **space**, nó kết thúc và chuyển sang argument tiếp theo. Nếu số lượng argument quá giới hạn, chương trình sẽ báo lỗi và kết thúc vòng lặp. Sau khi đã đọc hết dữ liệu từ luồng nhập chuẩn, chương trình **exit** với mã 0.

2.2.6 Time Spent

Đơn thuần là tạo một file ‘**time.txt**’ và gõ vào số giờ làm bài. ‘**time.txt**’ sẽ được lưu trong folder **xv6-labs-2023**.

Chương 3

Hoàn Thành

3.1 Tổng hợp file

3.1.1 Tạo diff so sánh kết quả

Từ folder xv6-labs-2023, gõ cú pháp: **git diff > Tên file.patch origin/util**. Cú pháp này sẽ tạo ra một file định dạng .patch ở trong folder xv6-labs-2023, với nội dung thể hiện sự khác biệt giữa lúc đầu và sau khi hoàn thành đồ án.

Lưu ý rằng, trước khi thực hiện **git diff**, các file chúng ta vừa tạo sẽ ở dạng untracked, tức những file chưa được thêm vào danh sách theo dõi của Git. Git sẽ không theo dõi mọi thay đổi của các file này, vì thế khi commit thay đổi, Git sẽ không lưu lại lịch sử thay đổi của các file này. Vì thế, trước đó hãy chạy câu lệnh **git add .** thì khi **git diff** thì các file sẽ được track đầy đủ và có thể so sánh với ban đầu.

3.1.2 Copy folder xv6 về Windows

Khi này, chúng ta sẽ quay trở về folder gốc của máy ảo. Lúc này, chúng ta sẽ gõ câu lệnh **cp -R xv6-labs-2023 /mnt/d**. Câu lệnh sẽ sao chép đệ quy tất cả các file và thư mục con trong xv6-labs-2023 và chuyển vào ổ đĩa D:/ trong máy Windows. Chúng ta hoàn toàn có thể áp dụng cho ổ đĩa C:/.

3.2 Chấm điểm

Có thể thấy rằng các tính năng đã gần như được hoàn thiện một cách chỉn chu nhất có thể, tất cả các test chạy ra đều đạt trạng thái OK và số điểm được chấm là 100/100.

Trong chương trình vẫn có thể còn những lỗi nhỏ nhặt, tuy nhiên tất cả những yếu tố trên đã thoả điều kiện của hàm **make grade** được cài sẵn trong xv6.

```

● huytienlam@MSI:/xv6-labs-2023$ make grade
make clean
make[1]: Entering directory '/home/huytienlam/xv6-labs-2023'
rm -f *.tex *.dvi *.idx *.aux *.log *.ind *.ilg \
*/*.o */*.d */*.asm */*.sym \
user/initcode user/initcode.out kernel/kernel fs.img \
mkfs/mkfs .gdbinit \
    user/usys.S \
user/_cat user/_echo user/_forktest user/_grep user/_init user/_kill user/_ln user/_ls user/_mkdir user/_rm user/_sh user/_stressfs user/_usertests user/_grin \
d user/_wc user/_zombie user/_sleep user/_pingpong user/_primes user/_find user/_xargs \
*.zip \
ph barrier
make[1]: Leaving directory '/home/huytienlam/xv6-labs-2023'
./grade-lab-util -v
make[1]: Entering directory '/home/huytienlam/xv6-labs-2023'
riscv64-unknown-elf-gcc -c -o kernel/entry.o kernel/entry.S
riscv64-unknown-elf-gcc -Wall -Werror -O -fno-omit-frame-pointer -ggdb -gdwarf-2 -DSOL_UTIL -DLAB_UTIL -MD -mcmode=medany -ffreestanding -fno-common -nostdlib \
b -mno-relax -I -fno-stack-protector -fno-pie -no-pie -c -o kernel/kalloc.o kernel/kalloc.c
riscv64-unknown-elf-gcc -Wall -Werror -O -fno-omit-frame-pointer -ggdb -gdwarf-2 -DSOL_UTIL -DLAB_UTIL -MD -mcmode=medany -ffreestanding -fno-common -nostdlib \
b -mno-relax -I -fno-stack-protector -fno-pie -no-pie -c -o kernel/string.o kernel/string.c
riscv64-unknown-elf-gcc -Wall -Werror -O -fno-omit-frame-pointer -ggdb -gdwarf-2 -DSOL_UTIL -DLAB_UTIL -MD -mcmode=medany -ffreestanding -fno-common -nostdlib \
b -mno-relax -I -fno-stack-protector -fno-pie -no-pie -c -o kernel/main.o kernel/main.c
riscv64-unknown-elf-gcc -Wall -Werror -O -fno-omit-frame-pointer -ggdb -gdwarf-2 -DSOL_UTIL -DLAB_UTIL -MD -mcmode=medany -ffreestanding -fno-common -nostdlib \
b -mno-relax -I -fno-stack-protector -fno-pie -no-pie -c -o kernel/vm.o kernel/vm.c
riscv64-unknown-elf-gcc -Wall -Werror -O -fno-omit-frame-pointer -ggdb -gdwarf-2 -DSOL_UTIL -DLAB_UTIL -MD -mcmode=medany -ffreestanding -fno-common -nostdlib \
b -mno-relax -I -fno-stack-protector -fno-pie -no-pie -c -o kernel/proc.o kernel/proc.c
riscv64-unknown-elf-gcc -c -o kernel/swtch.o kernel/swtch.S
riscv64-unknown-elf-gcc -c -o kernel/trampoline.o kernel/trampoline.S
riscv64-unknown-elf-gcc -Wall -Werror -O -fno-omit-frame-pointer -ggdb -gdwarf-2 -DSOL_UTIL -DLAB_UTIL -MD -mcmode=medany -ffreestanding -fno-common -nostdlib \
b -mno-relax -I -fno-stack-protector -fno-pie -no-pie -c -o kernel/trap.o kernel/trap.c
riscv64-unknown-elf-gcc -Wall -Werror -O -fno-omit-frame-pointer -ggdb -gdwarf-2 -DSOL_UTIL -DLAB_UTIL -MD -mcmode=medany -ffreestanding -fno-common -nostdlib \
b -mno-relax -I -fno-stack-protector -fno-pie -no-pie -c -o kernel/syscall.o kernel/syscall.c
riscv64-unknown-elf-gcc -Wall -Werror -O -fno-omit-frame-pointer -ggdb -gdwarf-2 -DSOL_UTIL -DLAB_UTIL -MD -mcmode=medany -ffreestanding -fno-common -nostdlib \
b -mno-relax -I -fno-stack-protector -fno-pie -no-pie -c -o kernel/sysproc.o kernel/sysproc.c
riscv64-unknown-elf-gcc -Wall -Werror -O -fno-omit-frame-pointer -ggdb -gdwarf-2 -DSOL_UTIL -DLAB_UTIL -MD -mcmode=medany -ffreestanding -fno-common -nostdlib \
b -mno-relax -I -fno-stack-protector -fno-pie -no-pie -c -o kernel/bio.o kernel/bio.c
riscv64-unknown-elf-gcc -Wall -Werror -O -fno-omit-frame-pointer -ggdb -gdwarf-2 -DSOL_UTIL -DLAB_UTIL -MD -mcmode=medany -ffreestanding -fno-common -nostdlib \
b -mno-relax -I -fno-stack-protector -fno-pie -no-pie -c -o kernel/fs.o fs.c
riscv64-unknown-elf-gcc -Wall -Werror -O -fno-omit-frame-pointer -ggdb -gdwarf-2 -DSOL_UTIL -DLAB_UTIL -MD -mcmode=medany -ffreestanding -fno-common -nostdlib \
b -mno-relax -I -fno-stack-protector -fno-pie -no-pie -c -o kernel/log.o kernel/log.c
riscv64-unknown-elf-gcc -Wall -Werror -O -fno-omit-frame-pointer -ggdb -gdwarf-2 -DSOL_UTIL -DLAB_UTIL -MD -mcmode=medany -ffreestanding -fno-common -nostdlib \
b -mno-relax -I -fno-stack-protector -fno-pie -no-pie -c -o kernel/sleeplock.o kernel/sleeplock.c
riscv64-unknown-elf-gcc -Wall -Werror -O -fno-omit-frame-pointer -ggdb -gdwarf-2 -DSOL_UTIL -DLAB_UTIL -MD -mcmode=medany -ffreestanding -fno-common -nostdlib \
b -mno-relax -I -fno-stack-protector -fno-pie -no-pie -c -o kernel/file.o kernel/file.c
riscv64-unknown-elf-gcc -Wall -Werror -O -fno-omit-frame-pointer -ggdb -gdwarf-2 -DSOL_UTIL -DLAB_UTIL -MD -mcmode=medany -ffreestanding -fno-common -nostdlib \
b -mno-relax -I -fno-stack-protector -fno-pie -no-pie -c -o kernel/pipe.o kernel/pipe.c
riscv64-unknown-elf-gcc -Wall -Werror -O -fno-omit-frame-pointer -ggdb -gdwarf-2 -DSOL_UTIL -DLAB_UTIL -MD -mcmode=medany -ffreestanding -fno-common -nostdlib \
b -mno-relax -I -fno-stack-protector -fno-pie -no-pie -c -o kernel/exec.o kernel/exec.c
riscv64-unknown-elf-gcc -Wall -Werror -O -fno-omit-frame-pointer -ggdb -gdwarf-2 -DSOL_UTIL -DLAB_UTIL -MD -mcmode=medany -ffreestanding -fno-common -nostdlib \
b -mno-relax -I -fno-stack-protector -fno-pie -no-pie -c -o kernel/sysfile.o kernel/sysfile.c
riscv64-unknown-elf-gcc -c -o kernel/kernelvec.o kernel/kernelvec.S
riscv64-unknown-elf-gcc -Wall -Werror -O -fno-omit-frame-pointer -ggdb -gdwarf-2 -DSOL_UTIL -DLAB_UTIL -MD -mcmode=medany -ffreestanding -fno-common -nostdlib \
b -mno-relax -I -fno-stack-protector -fno-pie -no-pie -c -o kernel/plic.o kernel/plic.c
riscv64-unknown-elf-gcc -Wall -Werror -O -fno-omit-frame-pointer -ggdb -gdwarf-2 -DSOL_UTIL -DLAB_UTIL -MD -mcmode=medany -ffreestanding -fno-common -nostdlib \
b -mno-relax -I -fno-stack-protector -fno-pie -no-pie -c -o kernel/virtio_disk.o kernel/virtio_disk.c
riscv64-unknown-elf-gcc -Wall -Werror -O -fno-omit-frame-pointer -ggdb -gdwarf-2 -DSOL_UTIL -DLAB_UTIL -MD -mcmode=medany -ffreestanding -fno-common -nostdlib \
b -mno-relax -I -fno-stack-protector -fno-pie -no-pie -c -o kernel/start.o kernel/start.c
riscv64-unknown-elf-gcc -Wall -Werror -O -fno-omit-frame-pointer -ggdb -gdwarf-2 -DSOL_UTIL -DLAB_UTIL -MD -mcmode=medany -ffreestanding -fno-common -nostdlib \
b -mno-relax -I -fno-stack-protector -fno-pie -no-pie -c -o kernel/console.o kernel/console.c
riscv64-unknown-elf-gcc -Wall -Werror -O -fno-omit-frame-pointer -ggdb -gdwarf-2 -DSOL_UTIL -DLAB_UTIL -MD -mcmode=medany -ffreestanding -fno-common -nostdlib \
b -mno-relax -I -fno-stack-protector -fno-pie -no-pie -c -o kernel/printf.o kernel/printf.c
riscv64-unknown-elf-gcc -Wall -Werror -O -fno-omit-frame-pointer -ggdb -gdwarf-2 -DSOL_UTIL -DLAB_UTIL -MD -mcmode=medany -ffreestanding -fno-common -nostdlib \
b -mno-relax -I -fno-stack-protector -fno-pie -no-pie -c -o kernel/uart.o kernel/uart.c
riscv64-unknown-elf-gcc -Wall -Werror -O -fno-omit-frame-pointer -ggdb -gdwarf-2 -DSOL_UTIL -DLAB_UTIL -MD -mcmode=medany -ffreestanding -fno-common -nostdlib \
b -mno-relax -I -fno-stack-protector -fno-pie -no-pie -c -o kernel/spinlock.o kernel/spinlock.c
riscv64-unknown-elf-gcc -Wall -Werror -O -fno-omit-frame-pointer -ggdb -gdwarf-2 -DSOL_UTIL -DLAB_UTIL -MD -mcmode=medany -ffreestanding -fno-common -nostdlib \
b -mno-relax -I -fno-stack-protector -fno-pie -no-pie -march=rv64g -nostdinc -I -Ikernel -c user/initcode.S -o user/initcode.o
riscv64-unknown-elf-ld -z max-page-size=4096 -N -e start -Ttext 0 -o user/initcode.out user/initcode.o
riscv64-unknown-elf-objcopy -S -O binary user/initcode.out user/initcode
riscv64-unknown-elf-objdump -S user/initcode.o > user/initcode.asm
riscv64-unknown-elf-ld -z max-page-size=4096 -T kernel/kernel.ld -o kernel/kernel kernel/kernel/entry.o kernel/kalloc.o kernel/string.o kernel/main.o kernel/vm.o ker \
n/kernel/proc.o kernel/swtch.o kernel/trampoline.o kernel/trap.o kernel/syscall.o kernel/sysproc.o kernel/bio.o kernel/fs.o kernel/log.o kernel/sleeplock.o kernel/ \
file.o kernel/pipe.o kernel/exec.o kernel/sysfile.o kernel/kernelvec.o kernel/plic.o kernel/virtio_disk.o kernel/start.o kernel/console.o kernel/printf.o kern \
el/uart.o kernel/spinlock.o
riscv64-unknown-elf-objdump -S kernel/kernel > kernel/kernel.asm
riscv64-unknown-elf-objdump -t kernel/kernel | sed '1,/SYMBOL TABLE/d; s/ .* / /; /^$/d' > kernel/kernel.sym
make[1]: Leaving directory '/home/huytienlam/xv6-labs-2023'
== Test sleep, no arguments ==
$ make qemu-gdb
sleep, no arguments: OK (5.1s)
== Test sleep, returns ==
$ make qemu-gdb
sleep, returns: OK (0.7s)
== Test sleep, makes syscall ==
$ make qemu-gdb
sleep, makes syscall: OK (0.7s)
== Test pingpong ==
$ make qemu-gdb
pingpong: OK (0.9s)
== Test primes ==
$ make qemu-gdb
primes: OK (1.1s)
== Test find, in current directory ==
$ make qemu-gdb
find, in current directory: OK (1.0s)
== Test find, recursive ==
$ make qemu-gdb
find, recursive: OK (1.3s)
== Test xargs ==
$ make qemu-gdb
xargs: OK (1.2s)
== Test time ==
time: OK
Score: 100/100

```

Hình 19. Make grade

Chương 4

Tổng Kết

Qua đồ án **Intro Xv6**, chúng em được hiểu rõ hơn về cách sử dụng dòng lệnh, cài đặt hệ điều hành và code thêm tính năng bằng ngôn ngữ C. Quá trình làm việc của nhóm chúng em khá rõ ràng, rành mạch, và có logic, phân chia công việc đều đặn. Chúng em xin chân thành cảm ơn!

MSSV	Họ và tên	Công việc	Hoàn thành
22127151	Lâm Tiến Huy	Cài đặt xv6 trên Ubuntu trên máy Windows. Lập trình sleep, pingpong. Làm báo cáo chính.	100%
22127290	Nguyễn Thị Thu Ngân	Cài đặt xv6 trên máy macOS. Lập trình primes, find. Làm báo cáo phụ.	100%
22127408	Kha Vĩnh Thuận	Cài đặt xv6 trên Ubuntu trên máy Windows. Lập trình find, xargs. Tester chính và gom file.	100%

Hình 20. Bảng phân công công việc

Tài liệu tham khảo

Lab: Xv6 and Unix utilities. (n.d.). <https://pdos.csail.mit.edu/6.1810/2023/labs/util.html>

Cox, R., Kaashoek, F., & Morris, R. (2022). *xv6: a simple, Unix-like teaching operating system.*

<https://pdos.csail.mit.edu/6.1810/2023/xv6/book-riscv-rev3.pdf>

Lab 1 Xv6 and Unix utilities - build a OS. (n.d.). https://xiayingp.gitbook.io/build_a_os/labs/lab-1-xv6-and-unix-utilities?fbclid=IwAR3jv_ztCju_jaB6Tjg6P_X3z7-D9yDniPo4fY_GGLcVEfOMuNH7qqNZcmA

Craigloewen-Msft. (2023, August 28). *Install WSL.* Microsoft Learn. <https://learn.microsoft.com/en-us/windows/wsl/install>

Ogany Supreme. (2022, December 28). *XV6 Install and add commands* [Video]. YouTube.

https://www.youtube.com/watch?v=yHD_FNIXiJo