TRƯỜNG ĐẠI HỌC KHOA HỌC TỰ NHIÊN, ĐHQG-HCM

Khoa Công nghệ Thông Tin

Môn học: Hệ Điều Hành

BÁO CÁO PROJECT 1 – XV6 LABS: UTILITIES

GV hướng dẫn: Lê Giang Thanh
Trần Trung Dũng

SV thực hiện:

MSSV	Họ và Tên
21120056	Nguyễn Đặng Tường Duy
21120462	Đỗ Khải Hưng

Mục lục

Thông tin lab và thông tin môn học

0.1 Thông tin môn học

- Tên môn học: Hệ điều hành.
- Khoa: Khoa Công nghệ Thông tin, Trường Đại học Khoa học Tự nhiên, ĐHQG-HCM.
- Giảng viên: Trần Trung Dũng, Lê Giang Thanh.
- Lớp: 21 Cử nhân Tài năng.

0.2 Thông tin lab

- Lab: Xv6 and Unix utilities.
- Người hướng dẫn: Trần Trung Dũng, Lê Giang Thanh.
- Hạn nộp: 22/10/2023.
- Nhóm thực hiện: Nhóm gồm các thành viên:
  1. Nguyễn Đặng Tường Duy - 21120056
  2. Đỗ Khải Hưng - 21120462

1 Introduction

1.1 xv6 là gì?

Xv6 là một hệ điều hành thời gian thực đơn giản được phát triển dựa trên Unix V6. Xv6 được sử dụng trong nhiều khóa học về hệ điều hành và hệ thống tại các trường đại học, bao gồm cả khóa học "6.828: Operating System Engineering" tại MIT.

Xv6 được viết bằng ngôn ngữ lập trình C và được thiết kế để hoạt động trên kiến trúc máy tính x86. Xv6 có cấu trúc đơn giản nhưng cung cấp một số tính năng quan trọng của một hệ điều hành như quản lý tiến trình, quản lý bộ nhớ ảo, quản lý tệp, và hỗ trợ mạng cơ bản.

Mục tiêu của xv6 là giúp sinh viên hiểu rõ cách hoạt động của một hệ điều hành, từ việc lên lịch thực thi tiến trình đến giao tiếp với các thiết bị và tệp tin. Xv6 cung cấp một môi trường thí nghiệm lý tưởng để nghiên cứu và thực hành các khái niệm hệ điều hành cơ bản.

Ngoài ra, xv6 cũng đi kèm với các bài thực hành, như lab "Xv6 and Unix utilities," để giúp sinh viên làm quen với hệ điều hành và phát triển kỹ năng lập trình hệ thống.

1.2 Lab: Xv6 and Unix utilities

1.2.1 Mục tiêu

Lab: Xv6 and Unix utilities là một bài thực hành của khóa học 6.828: Operating System Engineering tại MIT. Mục tiêu của bài thực hành này là giúp sinh viên làm quen với hệ điều hành xv6 và các lệnh hệ thống của nó.

1.2.2 Nội dung

Trong bài thực hành này, sinh viên sẽ được yêu cầu cài đặt một số chương trình người dùng cho xv6, như sleep, pingpong, find, xargs và bc. Những chương trình này sẽ giúp sinh viên tương tác với xv6 và thực hiện các tác vụ khác nhau. Sinh viên cũng sẽ được học cách sử dụng các công cụ như git, make và qemu để biên dịch và chạy xv6 trên máy ảo.

1.2.3 Lợi ích

Bài thực hành này sẽ giúp sinh viên hiểu rõ hơn về cấu trúc và hoạt động của một hệ điều hành đơn giản, cũng như kỹ năng lập trình hệ thống.Sinh viên sẽ có cơ hội khám phá các khái niệm như quá trình, lời gọi hệ thống, định tuyến, đồng bộ hóa, bộ nhớ ảo và hệ thống tập tin trong xv6. Sinh viên cũng sẽ có thể so sánh xv6 với các hệ điều hành khác, như Unix, Linux và Windows.

1.3 Cài đặt môi trường

Nhóm chúng em sử dụng máy ảo Linux với bản phân phối là Ubuntu, do đó, để cài đặt được hệ điều hành xv6, ta cần làm các bước như sau:

- Trong giao diện của Ubuntu, ta mở Terminal.
- Chạy lệnh: `sudo apt-get install git build-essential gdb-multiarch qemu-system-misc gcc-riscv64-linux-gnu binutils-riscv64-linux-gnu` để cài đặt các công cụ giúp hệ điều hành xv6 có thể chạy được.
- Ta cũng cần cài đặt Git, bằng cách chạy lệnh: `sudo apt-get install git`.

Chi tiết về cách thiết lập môi trường cho các hệ điều hành khác có thể tham khảo ở: `https://pdos.csail.mit.edu/6.828/2023/tools.html`.

2 Excercies

2.1 Boot xv6 (easy)

2.1.1 Mô tả đề bài và gợi ý

Ta cần tải mã nguồn và khởi chạy hệ điều hành xv6 lần đầu tiên. Đây là công việc khá đơn giản mà bất kì ai cũng cần phải làm được nếu muốn tiếp tục tìm hiểu về hệ điều hành xv6.

2.1.2 Cách làm và Giải thích

- Trên Terminal, chạy lệnh: `git clone git://g.csail.mit.edu/xv6-labs-2023` để tải về mã nguồn của hệ điều hành xv6.
- `cd xv6-labs-2023` để vào thư mục của xv6.
- Để chạy hệ điều hành xv6 thì ta cần chạy lệnh: `make qemu`.
- Để thoát xv6, ta nhấn tổ hợp phím `Ctrl + A`, sau đó nhấn `X`.

2.2 sleep (easy)

2.2.1 Mô tả đề bài và gợi ý

- **Mô tả đề bài:** Mục tiêu của bài tập này là viết một chương trình cho xv6 có tên là sleep, có thể tạm dừng một khoảng thời gian do người dùng chỉ định. Thời gian được đo bằng ticks, là đơn vị thời gian của xv6, tương ứng với khoảng thời gian giữa hai lần ngắt của bộ định thời.
- **Các gợi ý cho bài toán:**
  - Đọc chương 1 của sách xv6 để hiểu cơ bản về hệ điều hành và các hàm thư viện có sẵn.
  - Xem các chương trình mẫu trong thư mục user, như user/echo.c, user/grep.c và user/rm.c, để biết cách lấy các tham số dòng lệnh được truyền cho chương trình.
  - Kiểm tra xem người dùng có quên truyền tham số hay không, nếu có thì in ra một thông báo lỗi và thoát chương trình.
  - Sử dụng hàm atoi trong user/ulib.c để chuyển đổi tham số dòng lệnh từ chuỗi sang số nguyên.
  - Sử dụng hàm sleep trong user/user.h để gọi hệ thống ngủ. Hàm này sẽ gọi đến hàm sys_sleep trong kernel/sysproc.c, và sử dụng mã hợp dụng trong user/usys.S để chuyển sang chế độ nhân.
  - Đảm bảo gọi hàm exit() để kết thúc chương trình.
  - Thêm chương trình sleep vào danh sách UPROGS trong Makefile. Sau đó, sử dụng lệnh make qemu để biên dịch và chạy chương trình từ shell xv6.

2.2.2 Cách làm và Giải thích

1. Đầu tiên, xác định trong bài **sleep** này sẽ dùng lại system call đã được cài đặt sẵn, dựa vào các bài tương tự trong folder user, chúng em đã tìm thấy được folder user.h hỗ trợ các system call cũng như các hàm hỗ trợ để viết chương trình này.
2. Dựa vào file **kill.c** để xem cách truyền tham số dòng lệnh vào hàm.
3. Xử lý trường hợp ngoại lệ tham số không đúng số lượng.
4. Dùng atoi để chuyển thành số nguyên và gọi system call **sleep** và in lỗi nếu có ra màn hình.

2.3 pingpong (easy)

2.3.1 Mô tả đề bài và gợi ý

- **Mô tả đề bài:** Viết một chương trình cấp người dùng trong xv6 có tên là "pingpong" để thực hiện trò chơi "ping-pong" giữa hai tiến trình sử dụng `pipe`. Tiến trình cha gửi một byte cho tiến trình con, sau đó tiến trình con in ra "<pid>: received ping" và gửi byte đó lại cho tiến trình cha. Tiến trình cha sau đó in ra "<pid>: received pong" và kết thúc.
- **Các gợi ý cho bài toán:**
  - Sử dụng lời gọi `pipe` để tạo pipe cho việc truyền dữ liệu giữa hai tiến trình.
  - Sử dụng lời gọi `fork` để tạo một tiến trình con.
  - Sử dụng lời gọi `read` và `write` để đọc và ghi dữ liệu vào ống.
  - Sử dụng `getpid` để lấy ID của tiến trình hiện tại.

2.3.2 Cách làm và Giải thích

1. Đầu tiên, để hiểu rõ cách hoạt động của `pipe` và `fork` cần đọc qua `1.1 Processes and memory` và `1.3 Pipe` trong sách Xv6.
2. Import file `user.h` cho việc thao tác với các hàm có sẵn.
3. Khai báo mảng **p** chứa id của descriptors kiểu int, đồng thời biến buf kiểu char tương ứng 1 byte.
4. Tiến hành khởi tạo `pipe` và tạo tiến trình con bằng `fork`. Dùng giá trị trả về để phân biệt tiến trình con và cha
5. Trong tiến trình con dùng lệnh `read` lên file descriptor `p[0]` để in ra **ping** và `write` lên file descriptor `p[1]` để cho bên phát tín hiệu tiến trình cha
6. Tiến trình cha viết trước rồi `wait(0)` để chờ tiến trình con kết thúc và dọn dẹp bộ nhớ, đọc và in ra **pong**.
7. Đóng file và kết thúc.

2.3.3 Lưu ý:

- Ở đây phải dùng lệnh `wait(0)` để cho hệ thống dọn dẹp bộ nhớ ở tiến trình con khi tiến trình con trở thành **zombie process**.
- Đóng file ở cả 2 tiến trình để giảm số con trỏ descriptor về 0 tránh rò rỉ bộ nhớ.

2.4 primes (moderate)/(hard)

2.4.1 Mô tả đề bài và gợi ý

- **Mô tả đề bài:** Bài tập **primes** yêu cầu sinh viên sàng và in ra các số nguyên tố, sử dụng **pipe** và thiết kế đa tiến trình.
- **Các gợi ý cho bài toán:**
  - Tiến trình đầu tiên lọc tới số 35 thì tất đợi tất cả các tiến trình con của nó kết thúc.
  - Hàm read sẽ dừng khi tiến trình cha đóng file
  - Viết các số kiểu *int* thay vì ASCII
  - Chỉ tạo những processes khi cần.

2.4.2 Cách làm và Giải thích

- Đầu tiên, hình dung bài toán sẽ được giải theo dạng đệ quy, trong các tiến trình con lại tạo thêm các tiến trình con để *filter* tất cả số nguyên không nguyên tố.
- Import file `user.h` cho việc thao tác với các hàm có sẵn
- Ở mỗi tiến trình, ta sẽ đọc các số đã có ở tiến trình trước, in số nguyên **n** là số nguyên tố nhận được, kiểm tra chia hết và viết vào tiến trình con. Vì vậy trong hàm *filter* cần dùng cả `pipe` và `fork`.
- Trong hàm *filter*, cần đóng file viết của `pipe` trước đó. Ở tiến trình cha sau khi viết cũng cần đóng các file descriptor còn lại.
- Dùng `wait(0)` đợi tiến trình con kết thúc ở tiến trình cha. và hệ điều hành giải phóng vùng nhớ của các `zombie process` tránh bị lỗi.
- Đóng tất cả các file descriptors và `exit(0)` để kết thúc các tiến trình và kết thúc chương trình.

2.4.3 Lưu ý:

- Ở process đầu tiên ta không cần đọc mà chỉ viết vào process con, ở process cuối cùng file sẽ không có kí tự và hàm `read` sẽ trả về `false` và tiến hành terminate process bằng `exit(0)`.
- Khi dùng hàm `fork` tiến trình sẽ copy reference đến các file descriptor, vì vậy `close` writing file về 0 để file viết được đóng hoàn toàn và tiến trình con có thể đọc dữ liệu không bị lỗi.
- Hàm `read` ở tiến trình sẽ tự động block khi ở tiến trình cha chưa ghi data vào file và nó chỉ kết thúc quá trình đọc khi tiến trình cha đóng file.

2.5 find (moderate)

2.5.1 Mô tả đề bài và gợi ý

- **Mô tả đề bài:** Viết một phiên bản đơn giản của chức năng tìm kiếm (find) như trong hệ điều hành UNIX. Ta cần tìm toàn bộ các tệp có tên cho trước trong cây thư mục. Lời giải phải được viết trong file `user/find.c`.
- **Các gợi ý cho bài toán:**
  - Tham khảo cách duyệt cây thư mục trong file `user/ls.c`.
  - Để duyệt vào thư mục con, ta có thể dùng kỹ thuật đệ quy.
  - Do ta cần tạo thư mục và tệp tạm để thử nghiệm nên ở mỗi lần chạy mới, ta cần sử dụng lệnh `make clean && make qemu`.
  - Bài tập này chủ yếu đòi hỏi khả năng xử lý chuỗi thành thạo trong ngôn ngữ C.

2.5.2 Cách làm và Giải thích

- Ta có thể tái sử dụng đa phần mã nguồn trong file `user/ls.c` vì về cơ bản, lệnh `ls` (lệnh liệt kê cây thư mục) có chức năng gần tương tự với lệnh `find` mà ta cần cài đặt.
- Ta mở đường dẫn (path) của cây thư mục được cho trước bằng hàm `open()` đồng thời ta cũng cần kiểm tra xem đường dẫn có hợp lệ hay không thông qua hàm `stat()`.
- Ta duyệt qua các thư mục con hoặc tệp có trong địa chỉ hiện tại, nếu đối tượng là tệp và có tên trùng với tệp ta cần tìm thì in địa chỉ kèm tên tệp đó ra. Ngược lại, nếu là thư mục con thì ta dùng đệ quy để duyệt tiếp vào thư mục con đó.

2.6 xargs (moderate)

2.6.1 Mô tả đề bài và gợi ý

- **Mô tả đề bài:** Viết một phiên bản đơn giản của chương trình UNIX `xargs`: các tham số số của chương trình này mô tả một lệnh để chạy, nó đọc các dòng từ standard input và chạy lệnh cho mỗi dòng đó, đồng thời thêm dòng đó vào tham số dòng lệnh. Lời giải phải được viết trong file `user/xargs.c`.
- **Các gợi ý cho bài toán:**
  - Dùng các system call `fork()` và `exec()` để thực thi câu lệnh trên mỗi dòng đầu vào.
  - Cần dùng system call `wait(0);` để chờ cho tiến trình con chạy xong một câu lệnh.
  - Các thay đổi trong hệ thống của xv6 vẫn tồn tại, như các tệp tạm được tạo ra ở mỗi lần chạy, do đó ta cần sử dụng lệnh `make clean && make qemu`.

2.6.2 Cách làm và Giải thích

- Để thực thi các lệnh như đã mô tả trong đề bài, ta cần tạo một mảng chứa các tham số của command-line arguments (`argc` & `argv`), ở đây đặt là `xargc` & `xargv`.
- Sau đó, ta tiến hành đọc từng dòng từ đầu vào chuẩn (standard input), ta có thể đọc từng kí tự một đến khi nào gặp kí tự xuống dòng (\n) thì ta được một câu lệnh.
- Khi đã có được một câu lệnh đọc từ đầu vào, ta tiến hành chạy lệnh đó, ta cũng cần dùng system call `wait()` để chờ cho tiến trình con thực hiện xong câu lệnh nói trên.

3 Kết luận

3.1 Kết quả đạt được

Bằng các cách làm trên, chúng em đã triển khai các chương trình và dùng lệnh chấm điểm **./xv6-./grade-lab-util *'excercise-name'***.
Kết quả đạt được đều pass các test của hệ thống:

- **sleep:**

```
make[1]: Leaving directory '/home/ndtduy/xv6-labs-2023'
== Test sleep, no arguments == sleep, no arguments: OK (3.2s)
== Test sleep, returns == sleep, returns: OK (0.9s)
== Test sleep, makes syscall == sleep, makes syscall: OK (1.0s)
ndtduy@LENOVO:~/xv6-labs-2023$
```

- **pingpong:**

```
make[1]: Leaving directory '/home/ndtduy/xv6-labs-2023'
== Test pingpong == pingpong: OK (3.7s)
ndtduy@LENOVO:~/xv6-labs-2023$
```

- **primes:**

```
make[1]: Leaving directory '/home/ndtduy/xv6-labs-2023'
== Test primes == primes: OK (3.0s)
ndtduy@LENOVO:~/xv6-labs-2023$
```

- **find:**

```
make[1]: Leaving directory '/home/ndtduy/xv6-labs-2023'
== Test find, in current directory == find, in current directory: OK (3.8s)
== Test find, recursive == find, recursive: OK (1.8s)
ndtduy@LENOVO:~/xv6-labs-2023$
```

- **xargs:**

```
make[1]: Leaving directory '/home/ndtduy/xv6-labs-2023'
== Test xargs == xargs: OK (4.2s)
ndtduy@LENOVO:~/xv6-labs-2023$ S
```

```
xv6 kernel is booting

hart 2 starting
hart 1 starting
init: starting sh
$ sh < xargstest.sh
$ $ $ $ $ $ hello
hello
hello
$ $
```

3.2 Lời kết

- Trong lab này, chúng em đã thực hiện được các bài tập về cách khởi động hệ điều hành xv6, cách sử dụng các hàm hệ thống như fork, wait, pipe, exec, exit, kill, sleep, và đã hiểu được cơ chế hoạt động của các tiến trình trong xv6, cách tạo ra các tiến trình con và đồng bộ hóa chúng với tiến trình cha, cách sử dụng các ống dẫn để truyền dữ liệu giữa các tiến trình, và cách xử lý các tín hiệu ngắt từ bàn phím.Chúng em cũng đã làm quen với các công cụ lập trình như make, gcc, gdb, qemu, và cách sử dụng chúng để biên dịch, chạy và gỡ lỗi chương trình trong môi trường xv6.
- Lab rất bổ ích và chúng em rất mong được tiếp tục học tập và thực hành các lab tiếp theo của môn học này để có thể nắm vững kiến thức về hệ điều hành và phát triển kỹ năng lập trình của mình.