TRƯỜNG ĐẠI HỌC KHOA HỌC TỰ NHIÊN,
ĐẠI HỌC QUỐC GIA TPHCM
KHOA CÔNG NGHỆ THÔNG TIN

BÁO CÁO ĐỒ ÁN SỐ 1:

Xv6 and Unix utilities

Môn học : Hệ điều hành
Giảng viên lý thuyết : Trần Trung Dũng
Giảng viên hướng dẫn thực hành : Lê Giang Thanh
Nguyễn Thanh Quân

TP Hồ Chí Minh 10/2023

MỤC LỤC

Chapter 1

Bảng đánh giá thành viên

MSSV	Họ và tên	% đóng góp
21120471	Phan Gia Huy	100%
21120075	Trần Minh Hoàng	100%

Chapter 2

Khởi động hệ điều hành xv6

2.1 Cài đặt môi trường

- Cài đặt Window Subsystem for Linux
- Tải ứng dụng Ubuntu từ Microsoft Store
- Trên Ubuntu, cài đặt các gói phụ thuộc

```
    $ sudo apt-get update && sudo apt-get upgrade
    $ sudo apt-get install git build-essential gdb-multiarch qemu-system-misc
gcc-riscv64-linux-gnu binutils-riscv64-linux-gnu
```

2.2 Khởi động xv6

- Clone source code của xv6 từ github bằng lệnh git.
- Tiến hành biên dịch xv6 bằng lệnh make qemu. Điều này sẽ build kernel và boot nó trong máy ảo QEMU.

```
    $ git clone git://g.csail.mit.edu/xv6-labs-2023
    $ cd xv6-labs-2023
    $ make qemu
```

Chapter 3

Cài đặt các chương trình

3.1 Chương trình sleep

- **Mục đích**:
  Chương trình sleep được sử dụng để ngừng thực thi tiến trình hiện tại trong một khoảng thời gian nhất định.

- **Cấu trúc chương trình**:
  - Kiểm tra độ dài mảng argv phải bằng 2, nếu không in thông báo lỗi và kết thúc chương trình.

  - Chuyển đổi tham số thứ 2 từ chuỗi sang số nguyên bằng hàm atoi, sau đó gán cho biến ticks để lưu thời gian ngừng.

  - Gọi hàm sleep để ngừng tiến trình hiện tại trong thời gian ticks.

  - Kết thúc chương trình bằng mã trả về thành công.

```
#include "kernel/types.h"
#include "user/user.h"

int main(int argc, char *argv[]) {
    if (argc != 2) {
        printf("Usage: sleep <ticks>\n");
        exit(1);
    }
    int ticks = atoi(argv[1]);
    sleep(ticks);
    exit(0);
}
```

3.2 Chương trình pingpong

- **Mục đích**:
  Chương trình pingpong dùng để thực hiện giao tiếp giữa hai tiến trình cha và con bằng cách sử dụng pipe.

- **Cấu trúc chương trình**:
  - Chương trình sử dụng 2 pipe để thực hiện trao đổi thông tin: + p2c : tiến trình cha gửi thông tin đến tiến trình con. + c2p : tiến trình con gửi ngược lại thông tin cho tiến trình cha.

  - Tiến trình cha fork tạo ra tiến trình con

  - Kiểm tra biến pid (= fork()), xác định đang chạy tiến trình con hay cha.

  - Nếu đang chạy tiến trình con, tiến trình con sẽ đóng các đầu không sử dụng của 2 pipe, sau đó đọc thông tin (“ping”) gửi từ p2c, in ra màn hình, trả lời bằng tin (“pong”) lên c2p.

  - Nếu là tiến trình cha, thực hiện tương tự đóng 2 đầu pipe. Sau đó gửi thông tin (“ping”) xuống p2c, nhận tin (“pong”) từ c2p và in kết quả ra màn hình.

  - Kết thúc chương trình bằng mã trả về thành công.

```
#include "kernel/types.h"
#include "kernel/stat.h"
#include "user/user.h"

int
main() {
  int p2c[2];
  int c2p[2];

  pipe(p2c);
  pipe(c2p);

  int pid = fork();

  if(pid == 0) { // child
    close(p2c[1]);
    close(c2p[0]);
    char buf[100];
    read(p2c[0], buf, sizeof(buf));
    printf("%d: received ping\n", getpid());
    write(c2p[1], "pong", strlen("pong"));
    exit(0);

  } else { // parent
    close(p2c[0]);
    close(c2p[1]);
    write(p2c[1], "ping", strlen("ping"));
    char buf[100];
```

```
        read(c2p[0], buf, sizeof(buf));
        printf("%d: received %s\n", getpid(), buf);
        exit(0);
      }
    }
```

3.3 Chương trình primes

- **Mục đích**:
  Chương trình primes tìm kiếm và hiển thị các số nguyên tố từ 2 đến một giới hạn nhất đinh N bằng phương pháp sàn Eratosthenes.

- **Cấu trúc chương trình**:
  - Tạo pipe: Chương trình tạo một đối tượng pipe bằng cách sử dụng pipe(primes_pipe). Pipe này sẽ được sử dụng để kết nối giữa tiến trình cha và tiến trình con.

  - Tạo tiến trình con: Sử dụng hàm fork(), tiến trình cha tạo một tiến trình con. Tiến trình cha và con sẽ cùng thực hiện việc tìm kiếm số nguyên tố, nhưng ở các bước khác nhau.

  - Giao tiếp qua pipe: Tiến trình cha đóng đầu đọc của pipe (primes_pipe[0]) và tiến trình con đóng đầu ghi của pipe (primes_pipe[1]). Tiến trình cha sau đó gửi các số từ 2 đến N vào pipe sử dụng hàm write(). Sau khi gửi xong, tiến trình cha đóng đầu ghi của pipe và sử dụng wait(0) để đợi tiến trình con kết thúc.

  - Hàm sieve: Hàm sieve là hàm chạy bởi tiến trình con để thực hiện thuật toán Sàng Eratosthenes để tìm kiếm số nguyên tố. Hàm này nhận một đối tượng pipe làm đối số, cho biết cách giao tiếp với tiến trình cha và con con.

  - Tạo pipe mới: Tiến trình con tạo một đối tượng pipe mới (next_pipe) và tiếp tục tạo một tiến trình con mới. Tiến trình con mới sẽ tiếp tục thực hiện thuật toán Sàng Eratosthenes cho các số tiếp theo.

  - Lọc số nguyên tố: Trong vòng lặp vô hạn, tiến trình con hiện số nguyên tố đã đọc ra màn hình và đọc các số từ pipe. Nếu một số không chia hết cho số nguyên tố hiện tại, tiến trình con ghi số đó vào pipe next_pipe để tiến trình con mới xử lý.

  - Kết thúc tiến trình Con: Tiến trình con đóng đầu ghi của pipe next_pipe và sau đó sử dụng wait(0) để đợi tiến trình con mới kết thúc trước khi thoát.

```
    int main() {
        int primes_pipe[2];
        pipe(primes_pipe);
        int p;
```

```
        if (fork() == 0) {
            close(primes_pipe[1]);
            sieve(primes_pipe[0]);
        } else {
            close(primes_pipe[0]);
            for (p = 2; p <= N; p++) {
                write(primes_pipe[1], &p, sizeof(p));
            }
            close(primes_pipe[1]);
            wait(0);
        }

        exit(0);
    }

    void sieve(int p) {
        int num;
        if (read(p, &num, sizeof(num)) <= 0) {
            close(p);
            exit(0);
        }

        printf("prime %d\n", num);

        int next_pipe[2];
        pipe(next_pipe);

        if (fork() == 0) {
            close(next_pipe[1]);
            close(p);
            sieve(next_pipe[0]);
        } else {
            close(next_pipe[0]);
        }

        while (1) {
            int n;
            if (read(p, &n, sizeof(n)) <= 0) {
                close(p);
                break;
            }
            if (n % num != 0) {
```

```
                write(next_pipe[1], &n, sizeof(n));
            }
        }

        close(next_pipe[1]);
        wait(0);
        exit(0);
    }
```

3.4 Chương trình find

- **Mục đích**:
  Chương trình find tìm kiếm một tệp hoặc thư mục cụ thể trong một cây thư mục bắt đầu từ một đường dẫn cơ sở.

- **Cấu trúc chương trình**:
  - Hàm find: Hàm find là hàm chính của chương trình để thực hiện tìm kiếm. Hàm này nhận vào hai đối số: path (đường dẫn cơ sở) và target (tên tệp/thư mục mục tiêu).

  - Mở tệp hoặc thư mục được chỉ định bởi đường dẫn path sử dụng hàm open và kiểm tra nếu việc mở tệp không thành công.

  - Sử dụng fstat để lấy thông tin về tệp hoặc thư mục đã mở. Thông tin này bao gồm kiểu (file type) của đối tượng (tệp hoặc thư mục). Dựa trên kiểu của đối tượng, chương trình sẽ thực hiện các công việc khác:

    + Nếu đối tượng là tệp (kiểu T_FILE), nó so sánh tên tệp với tên mục tiêu (target). Nếu chúng trùng khớp, nó in ra đường dẫn của tệp này.

    + Nếu đối tượng là thư mục (kiểu T_DIR), chương trình sẽ tiếp tục kiểm tra nội dung của thư mục và thực hiện đệ quy để tìm kiếm trong thư mục con.

  - Trong quá trình đệ quy, chương trình xây dựng lại đường dẫn buf cho mỗi thư mục con và thực hiện lại các bước tương tự. Hàm 'main' là hàm chính của chương trình, nơi thực hiện xử lý đối số dòng lệnh và gọi hàm find.

    a. Chương trình kiểm tra nếu số lượng đối số dòng lệnh không đúng (không phải chính xác là 3 đối số - tên chương trình, đường dẫn cơ sở, và tên mục tiêu), nó in ra thông báo sử dụng cách sử dụng đúng.

    b. Nếu số lượng đối số đúng, chương trình gọi hàm find với đường dẫn cơ sở và tên mục tiêu được truyền từ đối số dòng lệnh.

    c. Cuối cùng, chương trình kết thúc với mã thoát (exit code) 0 để thông báo rằng nó đã kết thúc thành công.

```
void find(char *path, char *target) {
    char buf[512];
    char *p;
    int fd;
    struct dirent de;
    struct stat st;

    if ((fd = open(path, 0)) < 0) {
        fprintf(2, "find: cannot open %s\n", path);
        return;
    }

    if (fstat(fd, &st) < 0) {
        fprintf(2, "find: cannot stat %s\n", path);
        close(fd);
        return;
    }

    switch (st.type) {
        case T_FILE:
            if (strcmp(path, target) == 0) {
                printf("%s\n", path);
            }
            break;

        case T_DIR:
            if (strlen(path) + 1 + DIRSIZ + 1 > sizeof buf) {
                fprintf(2, "find: path too long\n");
                break;
            }
            strcpy(buf, path);
            p = buf + strlen(buf);
            *p++ = '/';

            while (read(fd, &de, sizeof(de)) == sizeof(de)) {
                if (de.inum == 0)
                    continue;

                memmove(p, de.name, DIRSIZ);
                p[DIRSIZ] = 0;
```

```
                if (stat(buf, &st) < 0) {
                    fprintf(2, "find: cannot stat %s\n", buf);
                    continue;
                }

                if (st.type == T_DIR && strcmp(de.name, ".") != 0 &&
strcmp(de.name, "..") != 0) {
                    find(buf, target);
                }

                if (st.type == T_FILE && strcmp(de.name, target) == 0) {
                    printf("%s\n", buf);
                }
            }
            break;
    }
    close(fd);
}

int main(int argc, char *argv[]) {
    if (argc != 3) {
        fprintf(2, "Usage: find <path> <filename>\n");
        exit(1);
    }
    find(argv[1], argv[2]);
    exit(0);
}
```

3.5 Chương trình xargs

- **Mục đích**:
  Cho phép người dùng chạy một lệnh hoặc một chuỗi lệnh nhiều lần dựa trên dữ liệu đầu vào được cung cấp thông qua đầu vào chuẩn.

- **Cấu trúc chương trình**:
  - Chương trình bắt đầu bằng việc kiểm tra số lượng đối số từ dòng lệnh. Nếu số lượng đối số ít hơn 2 (bao gồm tên chương trình và ít nhất một lệnh), chương trình sẽ in ra thông báo sử dụng đúng và kết thúc với mã lỗi 1.

  - Chương trình sao chép các đối số từ dòng lệnh sau tên chương trình (‘xargs’) vào mảng

'cmd' để lưu trữ các đối số của lệnh cần thực thi. Nó cũng duyệt qua các đối số này và đếm số lượng đối số bằng cách tăng giá trị của biến 'cmd_argc'.

- Chương trình bắt đầu một vòng lặp vô hạn (while (1)) để thực hiện việc đọc dữ liệu từ đầu vào tiêu chuẩn (stdin). Trong vòng lặp, chương trình sử dụng hàm read để đọc từng ký tự từ đầu vào. Nó kiểm tra mỗi ký tự và xác định khi nào nên kết thúc đọc dựa trên ký tự xuống dòng (\n). Khi gặp ký tự xuống dòng, nó kết thúc việc đọc và tiến hành xử lý dòng dữ liệu đã đọc.

- Khi có dòng dữ liệu đầy đủ (kết thúc bởi xuống dòng), chương trình tạo một tiến trình con bằng cách sử dụng hàm fork. Trong tiến trình con, nó thực hiện các bước sau:

+ Xây dựng lại danh sách đối số của lệnh bằng cách thêm dòng dữ liệu vào cuối danh sách đối số.

+ Thực hiện lệnh cụ thể sử dụng hàm exec. Nếu lệnh không thực thi thành công, tiến trình con in ra thông báo lỗi và kết thúc với mã lỗi 1.

- Trong tiến trình cha, sau khi tiến trình con đã tạo, chương trình kiểm tra xem việc tạo tiến trình con có thành công hay không. Nếu không thành công, nó in ra thông báo lỗi.

- Tiến trình cha sử dụng hàm 'wait' để đợi cho đến khi tiến trình con kết thúc. Sau khi tiến trình con kết thúc, tiến trình cha kiểm tra kết quả của tiến trình con. Nếu có lỗi trong quá trình thực thi lệnh, nó in ra thông báo lỗi.

- Vòng lặp tiếp tục cho đến khi không còn dữ liệu từ đầu vào. Khi đó, chương trình thoát khỏi vòng lặp vô hạn và kết thúc.

```
int main(int argc, char *argv[]) {
  char line[512];
  char *cmd[MAX_ARGV];
  int cmd_argc = 0;
  int pid, status;

  if (argc < 2) {
    fprintf(2, "Usage: xargs command [args...]\n");
    exit(1);
  }
  for (int i = 1; i < argc; i++) {
    cmd[cmd_argc++] = argv[i];
  }

  while (1) {
    int i = 0;
    char c;
    int read_success = 0;
```

```
    while (read(0, &c, 1) > 0) {
      if (c == '\n') {
        line[i] = '\0';
        read_success = 1;
        break;
      }
      line[i++] = c;
    }

    if (!read_success) {
      break;
    }
    if ((pid = fork()) == 0) {
      cmd[cmd_argc] = line;
      cmd[cmd_argc + 1] = 0;
      exec(cmd[0], cmd);
      fprintf(2, "xargs: exec %s failed\n", cmd[0]);
      exit(1);
    } else if (pid < 0) {
      fprintf(2, "xargs: fork failed\n");
    }
    if (wait(&status) < 0) {
      fprintf(2, "xargs: wait failed\n");
    }
  }

  exit(0);
}
```

Chapter 4

Tổng kết đánh giá

Nhóm đã hoàn thành cài đặt các chương trình. Dưới đây là bảng kết quả sau khi dùng lệnh **make grade** :

```
== Test sleep, no arguments ==
$ make qemu-gdb
sleep, no arguments: OK (7.9s)
== Test sleep, returns ==
$ make qemu-gdb
sleep, returns: OK (1.2s)
== Test sleep, makes syscall ==
$ make qemu-gdb
sleep, makes syscall: OK (1.0s)
== Test pingpong ==
$ make qemu-gdb
pingpong: OK (1.2s)
== Test primes ==
$ make qemu-gdb
primes: OK (1.0s)
== Test find, in current directory ==
$ make qemu-gdb
find, in current directory: OK (1.1s)
== Test find, recursive ==
$ make qemu-gdb
find, recursive: OK (1.4s)
== Test xargs ==
$ make qemu-gdb
xargs: OK (2.0s)
== Test time ==
time: OK
Score: 100/100
```

Tham khảo

[1] Cài đặt môi trường : https://pdos.csail.mit.edu/6.1810/2023/tools.html

[2] Cài đặt WSL2 : https://pureinfotech.com/install-windows-subsystem-linux-2-windows-10/