ĐẠI HỌC QUỐC GIA TPHCM

TRƯỜNG ĐẠI HỌC KHOA HỌC TỰ NHIÊN

KHOA CÔNG NGHỆ THÔNG TIN

BÁO CÁO ĐỒ ÁN

Chủ đề: Xv6 - Unix Utilities

Môn học: Hệ điều hành

Sinh viên thực hiện:

Vũ Hoàng Minh - 2317427

Nguyễn Đăng Phôn - 23127451

Nguyễn Nam Việt - 23127517

Giáo viên hướng dẫn:

Lê Viết Long

Ngày 1 tháng 3 năm 2025

Mục lục

1 Bảng phân công

STT	Yêu cầu	Phụ trách	Hoàn Thành
1	Sleep	Vũ Hoàng Minh	100%
2	Pingpong	Vũ Hoàng Minh	100%
3	Primes	Nguyễn Nam Việt	100%
4	Find	Nguễn Đăng Phôn	100%
5	Xargs	Nguễn Đăng Phôn	100%
6	Báo cáo	Vũ Hoàng Minh Nguyễn Nam Việt Nguyễn Đăng Phôn	100%

Bảng 1: Phân chia công việc và mức độ hoàn thành

2 Thực hành đồ án

2.1 Chương trình Sleep

2.1.1 Yêu cầu:

Triển khai chương trình **sleep** cấp độ người dùng cho xv6, dựa theo lệnh sleep của UNIX. Chương trình Sleep của bạn sẽ tạm dừng theo số ticks do người dùng chỉ định. Một tick là khái niệm về thời gian được định nghĩa bởi kernel xv6, cụ thể là thời gian giữa hai lần ngắt từ timer chip. Giải pháp của bạn sẽ nằm trong tệp user/sleep.c.

2.1.2 Cách hoạt động của chương trình

Hàm main: sẽ lấy đối số truyền vào để thực thi chương trình sleep.

2.1.3 Giải thích thuật toán

Hàm main:

- Kiểm tra số lượng đối số truyền vào. Nếu khác 1 thì chương trình không chạy.

- Tiếp theo sẽ kiểm tra đối số truyền vào. Nếu đối số truyền vào bé hơn hoặc bằng 0 thì sẽ báo lỗi và thoát chương trình. Nếu thỏa điều kiện thì sẽ gọi hàm sleep và chạy đúng số tick nhập vào.

2.2 Chương trình Ping pong

2.2.1 Yêu cầu:

"Ping-pong" một byte giữa hai tiến trình qua một cặp pipe (tương ứng với gởi và nhận). Tiến trình cha sẽ gửi một byte cho tiến trình con; tiến trình con sẽ in "**<pid>: received ping**", trong đó **<pid> là ID tiến trình của nó**, ghi byte vào pipe cho tiến trình cha và thoát; tiến trình cha sẽ đọc byte từ tiến trình con, in "**<pid>: received pong**" và thoát.

2.2.2 Cách hoạt động của chương trình

Hàm main: tạo 2 ống cha và con dùng để gửi byte. Gọi hàm fork, read, write, getpid, wait để thực thi các lệnh.

2.2.3 Giải thích thuật toán

Hàm main:

- Tạo 2 mảng có 2 phần tử tương ứng ống cha và con. Phần tử thứ nhất tương ứng đầu đọc, phần tử thứ hai tương ứng đầu ghi.
- Dùng hàm pipe để tạo 2 pipe nếu không được thì báo lỗi và thoát.
- Dùng hàm fork để tạo tiến trình con. Nếu giá trị trả về là 0 thì là tiến trình con còn lớn hơn 0 là tiến trình cha. Hai tiến trình cha và con sẽ chạy song song.
- Nếu là tiến trình con sẽ đóng đầu ghi của tiến trình cha và đầu đọc của tiến trình con (để hạn chế rõ rĩ thông tin). Sau đó đọc 1 byte của tiến trình cha và in ra thông báo. Tiếp theo ghi byte vào pipe, gửi lại cho tiến trình cha và đóng các đầu còn lại.
- Nếu là tiến trình cha sẽ đóng đầu ghi của tiến trình con và đầu đọc của tiến trình cha. Kiểm tra ghi một byte vào pipe và gửi cho tiến trình con được không, sử dụng hàm wait để đợi tiến trình con gửi lại. Kiểm tra tiến trình con có gửi lại không

bằng cách đọc 1 byte trả về. Nếu thành công sẽ xuất thông báo, đóng các đầu còn lại và thoát chương trình.

2.3 Chương trình Primes

2.3.1 Yêu cầu:

Bài tập yêu cầu viết một chương trình sàng nguyên tố trong **xv6** sử dụng **pipe** và **fork()** dựa theo thiết kế của Doug McIlroy. Chương trình này phải lọc các số nguyên tố từ 2 đến 280 và hiển thị chúng theo dòng.

2.3.2 Cách hoạt động của chương trình

Hàm main:

- **Tạo pipe** (p[2]) để truyền số từ tiến trình cha sang tiến trình con.
- **Gọi** fork():
  + Tiến trình con đọc số từ pipe và gọi **primes()**.
  + Tiến trình cha ghi các số từ 2 đến 280 vào pipe rồi đóng pipe.
- **Gọi** wait(0): Đợi tiến trình con kết thúc trước khi thoát.

Hàm primes:

- **Đọc số đầu tiên từ pipe** và coi đó là số nguyên tố.
- **In số nguyên tố** và tạo pipe mới (p[2]) để truyền số cho tiến trình con.
- **Gọi** fork():
  + **Tiến trình con:** Đọc từ pipe mới và gọi primes(p[0]).
  + **Tiến trình cha:** Lọc các số chia hết cho số nguyên tố, chỉ chuyển số không chia hết sang pipe mới.
- **Đóng tất cả pipe không dùng** để tránh bị rò rỉ file descriptor.
- **Chờ tiến trình con kết thúc** bằng wait(0), để tránh **zombie process**.

2.3.3 Giải thích thuật toán

- Chương trình sử dụng phương pháp **đệ quy** và **pipeline** để loại bỏ các bội số của số nguyên tố
- Mỗi số nguyên tố tìm được sẽ **tạo một tiến trình con** để xử lý các số tiếp theo.

- Quy trình **đọc, lọc, truyền** lặp lại cho đến khi không còn số nào trong pipeline.
- Hàm fork():
    + Tạo ra tiến trình con.
    + Tiến trình cha nhận **PID** dương của tiến trình con.
    + Tiến trình con nhận **PID = 0**.
    + Trả về -1 nếu không tạo được.
- Hàm pipe():
    + Tạo pipe giữa hai tiến trình.
    + p[0] dùng để **đọc**, p[1] dùng để **ghi**.
- Hàm read() và write():
    + read(): Đọc **4 byte** từ pipe fd vào buffer.
    + write(): Ghi **4 byte** từ buffer vào pipe fd.
- Hàm wait(0):
    + Chờ tiến trình con kết thúc, tránh **zombie process**.

2.3.4 Quản lý tài nguyên (file descriptor, tiến trình, bộ nhớ)

- **Tránh leak file descriptor** bằng cách **đóng pipe không dùng**.
- **Giới hạn pipe và tiến trình:** chỉ tạo tiến trình con khi cần.

2.4 Chương trình Find

2.4.1 Yêu cầu:

Viết chương trình tìm tất cả các tệp có trong cây thư mục với tên cụ thể.

2.4.2 Cách hoạt động của chương trình

- Hàm main: Kiểm tra đầu vào và gọi hàm find.
- Hàm find: Hàm này thực hiện tìm kiếm đệ quy trong thư mục để tìm tệp tin có tên trùng khớp với tên tệp cần tìm.
- Hàm fmtname: Hàm này trích xuất tệp tin từ đường dẫn đầy đủ và định dạng nó để có độ dài cố định (14 ký tự).

2.4.3 Giải thích thuật toán

- char* fmtname(char* path):
    + Hàm fmtname trả về kết quả là tên file cuối cùng của một đường dẫn (ví dụ đường dẫn là a/aa/b thì kết quả trả về sẽ là b).
    + Trước hết, tìm vị trí ký tự '/' cuối cùng trong đường dẫn.
    + Nếu tên tệp dài hơn DIRSIZ (14 ký tự, được định nghĩa trong kernel/fs.h), trả về tên gốc. Nếu ngắn hơn thì thêm ký tự trắng để đảm bảo độ dài.
    + Lệnh memmove(buf, p, strlen(p)): Hàm memmove sao chép khối bộ nhớ được trỏ bởi p sang vị trí được trỏ bởi buf.
- void find(char *path, char *filename):
    + Mở thư mục được chỉ định bởi **path**.
    + Kiểm tra loại của **path** là tệp tin hay thư mục.
    + Nếu là thư mục thì đọc từng thư mục/tệp tin trong thư mục:
        * Xây dựng đường dẫn đầy đủ mọi mục.
        * Nếu mục là tệp tin và cùng tên với **filename** thì in ra đường dẫn.
        * Nếu mục là thư mục thì tiếp tục gọi file để tìm kiêms trong các thư mục con.
- Hàm main:
    + Kiểm tra đối số đầu vào và gọi hàm find để thực hiện yêu cầu.
    + Nếu số lượng đối số đầu vào **argc** $<$ 3 trả về lỗi thiếu đối số đầu vào. Nếu số lượng đối số đầu vào **argc** $>=$ 3 thì thực hiện một vòng for tìm tất cả các tệp cần tìm đường dẫn với đường dẫn ban đầu là '.'.

2.5 Chương trình xargs

2.5.1 Yêu cầu:

Viết chương trình mô phỏng lệnh xargs trong UNIX/LINUX cho xv6: Các đối số của chương trình mô tả lệnh để chạy.

2.5.2 Cách hoạt động của chương trình

- Hàm main: Kiểm tra đầu vào, đọc dữ liệu và gọi hàm fork và exec để thực thi lệnh.

2.5.3 Giải thích thuật toán

- Hàm main:
  + Kiểm tra đầu vào, đọc dữ liệu từ stdin, và thực thi lệnh.
  + Kiểm tra số lượng đối số **argc**. Nếu không đủ đối số, in thông báo và hướng dẫn sử dụng rồi thoát.
  + Khởi tạo mảng args để lưu trữ các đối số từ dòng lệnh và stdin
  + Đọc từng dòng từ stdin, kết hợp với các đối số từ dòng lệnh, thực thi lệnh.
  + Sử dụng **while** để đọc từng ký tự stdin.
  + Khi gặp ký tự \n thì kết thúc chuỗi và xử lý dòng đã đọc.
  + Sử dụng **fork** để tạo tiến trình con.
  + Trong tiến trình con, sử dụng **exec** để thực thi lệnh với các đối số. Nếu **exec** thất bại, in thông báo lỗi và thoát.
  + Trong tiến trình cha, sử dụng **wait** để chờ tiến trình con kết thúc.

3 Tài liệu tham khảo

- Giáo viên hướng dẫn Lê Viết Long
- https://pdos.csail.mit.edu/6.1810/2024/labs/util.html
- https://swtch.com/ rsc/thread/
- https://pdos.csail.mit.edu/6.1810/2024/xv6/book-riscv-rev4.pdf
- https://github.com/mit-pdos/xv6-riscv
- ChatGPT