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


# BÁO CÁO ĐỒ ÁN

## Chủ đề: Xv6 and Unix Utilities

**Môn học: Hệ điều hành**

**Giảng viên hướng dẫn:** Lê Viết Long

**Lớp:** 22_1

| ***Họ và tên*** | ***MSSV*** |
|---|---|
| Nguyễn Văn Hiến | 22120101 |
| Nguyễn Anh Trí | 22120382 |
| Trần Anh Tú | 22120401 |

TP.Hồ Chí Minh, 10/2024

# Mục lục

**1 Giới thiệu** **2**

**2 Phân công công việc** **3**

**3 Thực hiện** **4**
3.1 Chương trình Sleep . . . . . 4
3.2 Chương trình Ping pong . . . . . 4
3.3 Chương trình Primes . . . . . 6
3.4 Chương trình Find . . . . . 6
3.5 Xargs . . . . . 8

# 1 Giới thiệu

Xv6 là một hệ điều hành đơn giản, mô phỏng Unix v6, được phát triển bởi Đại học MIT để giảng dạy các khái niệm cơ bản về hệ điều hành. Trong đồ án này, chúng em đã thực hành với Xv6, tập trung vào việc triển khai và mở rộng các tiện ích (utilities) như sleep, ping pong, primes, find, và xargs. Báo cáo sẽ trình bày cách tiếp cận và giải quyết từng bài tập, đồng thời phân tích lý do chọn các giải pháp đã áp dụng.

# 2 Phân công công việc

| Bài tập | Họ và tên | Mức độ hoàn thành |
|---|---|---|
| Sleep | Trần Anh Tú | 100% |
| Ping pong | Trần Anh Tú | 100% |
| Primes | Nguyễn Anh Trí | 100% |
| Find | Nguyễn Văn Hiến | 100% |
| Xargs | Trần Anh Tú | 100% |
| Tổng hợp báo cáo | Nguyễn Văn Hiến | 100% |

Bảng 1 – Bảng phân công công việc

Do các thành viên được chia ra để giải quyết từng bài nhỏ nên từng thành viên sẽ tự viết nội dung báo cho bài mà bản thân đảm nhiệm.

# 3 Thực hiện

## 3.1 Chương trình Sleep

- **Mục tiêu:** Chương trình sleep.c được viết để mô phỏng chức năng "sleep"trong hệ điều hành. Nó tạm dừng thực thi của quy trình hiện tại trong một khoảng thời gian nhất định được chỉ định bởi người dùng. Thời gian ngủ được tính theo đơn vị "ticks đơn vị thời gian cơ bản của hệ điều hành.
- **Các bước thực hiện:**
  - **Bước 1:** Kiểm tra số lượng đối số
    - Nếu người dùng không cung cấp bất kỳ đối số nào hoặc cung cấp không đúng định dạng, chương trình sẽ in ra thông báo hướng dẫn sử dụng và thoát với mã lỗi.
  - **Bước 2:** Chuyển đổi đối số từ chuỗi thành số nguyên
    - Sử dụng hàm atoi() để chuyển đổi chuỗi (string) đó thành số nguyên để có thể sử dụng được trong hàm sleep()
  - **Bước 3:** Kiểm tra tính hợp lệ của đối số
    - Sau khi chuyển đổi đối số thành số nguyên, chương trình kiểm tra tính hợp lệ của giá trị này: Nếu số lượng ticks nhỏ hơn hoặc bằng 0, chương trình sẽ in ra thông báo lỗi "Invalid argument"và thoát.
  - **Bước 4:** Thực hiện chức năng sleep
    - Nếu đối số hợp lệ, chương trình gọi hàm sleep() với tham số là số ticks mà người dùng đã chỉ định. Hệ điều hành sẽ tạm dừng quy trình trong khoảng thời gian này.
  - **Bước 5:** Thoát chương trình
    - Sau khi quy trình đã "ngủ"đủ thời gian yêu cầu, chương trình sẽ kết thúc bằng cách gọi exit(0).
- **Thuật toán sử dụng**
  - Kiểm tra đối số: Nếu số lượng đối số khác 2, chương trình kết thúc với mã lỗi (exit(1)). Chuyển đổi đối số: Lấy đối số thứ hai (thời gian sleep), chuyển thành số nguyên. Gọi sleep(): Truyền số đã chuyển đổi vào system call sleep() để tạm dừng chương trình trong thời gian tương ứng. Kết thúc chương trình: Sau khi ngủ xong, chương trình thoát thành công (exit(0)).

## 3.2 Chương trình Ping pong

- **Mục tiêu:** Chương trình Ping-Pong nhằm mô phỏng giao tiếp giữa hai quy trình cha và con, sử dụng cơ chế truyền dữ liệu liên quy trình (Inter-process Communication - IPC) thông qua pipe. Mục đích của chương trình là cho phép quy trình cha và quy trình con trao đổi dữ liệu với nhau theo kiểu yêu cầu-phản hồi (request-response).
- **Các bước thực hiện:**
  - **Bước 1:** Tạo các Pipe
    - Pipe p1: dùng để gửi dữ liệu từ cha đến con.

        - Pipe p2: dùng để gửi dữ liệu từ con về cha.
    - Mỗi pipe được tạo ra với hai đầu (là một mảng có 2 phần tử), một đầu để đọc và một đầu để ghi dữ liệu
- **Bước 2:** Tạo Quy Trình Con bằng fork()
    - Sử dụng hàm fork() để tạo ra quy trình con từ quy trình cha. Sau khi gọi fork(), hệ điều hành sẽ phân chia chương trình thành hai quy trình chạy song song:
        - Quy trình cha: Tiếp tục thực thi phần mã gốc.
        - Quy trình con: Bắt đầu thực thi từ vị trí gọi hàm fork(), nhưng với giá trị trả về là 0.
- **Bước 3:** Đóng Các Đầu Không Cần Thiết của Pipe
    - Quy trình cha
        - Đóng đầu đọc của pipe p1 (chỉ ghi dữ liệu vào p1).
        - Đóng đầu ghi của pipe p2 (chỉ đọc dữ liệu từ p2).
    - Quy trình con:
        - Đóng đầu ghi của pipe p1 (chỉ đọc dữ liệu từ p1).
        - Đóng đầu đọc của pipe p2 (chỉ ghi dữ liệu vào p2).
- **Bước 4:** Gửi và Nhận Dữ Liệu Giữa Quy Trình Cha và Con
    - Quy trình cha:
        - Gửi dữ liệu: Quy trình cha gửi một tín hiệu (cụ thể là một byte dữ liệu) qua pipe p1 đến quy trình con.
        - Nhận phản hồi: Sau khi quy trình con xử lý và gửi phản hồi, quy trình cha sẽ nhận lại dữ liệu qua pipe p2.
    - Quy trình con:
        - Nhận dữ liệu: Quy trình con đọc tín hiệu từ pipe p1 do quy trình cha gửi đến.
        - Xử lý và phản hồi: Sau khi nhận dữ liệu, quy trình con phản hồi bằng cách gửi một tín hiệu ngược lại cho quy trình cha qua pipe p2.
- **Bước 5:** Đóng Pipe Sau Khi Sử Dụng
    - Sau khi hoàn tất việc trao đổi dữ liệu, cả quy trình cha và con cần đóng các pipe mà họ đã sử dụng để giải phóng tài nguyên và tránh tình trạng rò rỉ tài nguyên trong hệ thống.
- **Bước 6:** Kiểm Thử Chương Trình
    - Chương trình sẽ được kiểm thử bằng cách chạy các lệnh trong project đã yêu cầu đảm bảo:
        - Xác nhận rằng quy trình con có thể nhận đúng tín hiệu từ cha và phản hồi lại.
        - Xác nhận rằng quy trình con có thể nhận đúng tín hiệu từ cha và phản hồi lại.

- **Thuật toán sử dụng**
    - Thuật toán "Ping-Pong"sử dụng fork() để tách quy trình cha và con. Khi fork() trả về 0, quy trình đó là quy trình con; nếu trả về giá trị khác 0, đó là quy trình cha.
        - Quy trình cha ghi chữ "ping"vào pipe và đợi con thực hiện bằng wait(0). Do cả cha và con đều chia sẻ pipe, con có thể đọc dữ liệu cha ghi vào bằng read(), in thông báo nhận được "ping", và sau đó ghi lại "pong"vào pipe.

    * Quy trình con sau khi gửi "pong", đóng các đầu pipe để tránh treo pipe. Cha đọc thông điệp "pong"từ pipe và in thông báo, hoàn thành quá trình trao đổi.

## 3.3 Chương trình Primes

- **Yêu cầu bài tập:** Viết một chương trình sàng nguyên tố đồng thời cho xv6 sử dụng các pipe và thiết kế minh họa trong hình ở giữa trang này và nội dung xung quanh. Ý tưởng này là của Doug McIlroy, người phát minh ra Unix Pipes. Giải pháp của bạn phải nằm trong tệp user/primes.c.
- **Mục tiêu:** sử dụng pipe và fork để thiết lập pipeline, dùng thuật toán sàng Eratosthenes (the sieve of Eratosthenes) để tìm các số nguyên tố, qua đó hình dung được cơ chế liên lạc giữa các tiến trình (pipes) và tạo tiến trình con (fork).
- **Các bước thực hiện:**
  - **Bước 1:** Khai báo p[2] là mảng chứa 2 đầu đọc và ghi của pipe, num là biến lưu trữ số đọc từ pipe.
  - **Bước 2:** Tạo 1 pipe mới (pipe(p)), nếu thất bại thì báo lỗi và thoát chương trình.
  - **Bước 3:** Đọc 1 số từ pipe fd (file descriptor) và lưu vào biến num. Nếu đọc thành công, in ra số đó như là số nguyên tố.
  - **Bước 4:** Tạo tiến trình con fork(): nếu fork() = 0 thì ta đang ở tiến trình con.
  - **Bước 5:** Đóng đầu đọc của pipe; khai báo biến next_num để lưu số tiếp theo từ pipe, sau đó đọc các số tiếp theo từ fd, sử dụng thuật toán sàng Eratosthenes để sàng số nguyên tố ghi vào pipe p[1]. Đóng pipe và thoát tiến trình con.
  - **Bước 6:** Đóng đầu ghi của pipe p[1], đầu đọc của pipe fd để giải phóng tài nguyên, tránh rò rỉ và đảm bảo không đọc nhầm từ pipe cũ; gọi đệ quy hàm primes(p[0]) với đầu đọc của pipe mới; hàm wait(0) chờ tiến trình con hoàn thành, đảm bảo pipe không kết thúc trước khi fork hoàn thành; đóng file descriptor.
  - **Bước 7:** Hàm main dùng để chạy chương trình:
    * Tạo 1 pipe mới để giao tiếp giữa các tiến trình, nếu thất bại thì báo lỗi và thoát chương trình.
    * Tạo tiến trình con, trong đó, đóng đầu đọc của pipe p[0], ghi các số từ 2 đến 280 vào pipe p[1], đóng pipe và thoát tiến trình con.
    * Ở tiến trình cha, đóng đầu ghi của pipe p[1] để tránh rò rỉ tài nguyên, gọi đệ quy hàm primes(p[0]) để đọc số nguyên tố tiếp theo từ pipe và tạo ra tiến trình con mới để sàng lọc các số không phải số nguyên tố, hàm wait(0) chờ cho tiến trình con hoàn thành.
    * Kết thúc chương trình.

## 3.4 Chương trình Find

- **Yêu cầu bài tập:** Viết một phiên bản đơn giản của chương trình find của UNIX cho xv6: tìm tất cả các tệp trong cây thư mục có tên cụ thể. Giải pháp của bạn phải nằm trong tệp user/find.c.
- **Giải thích thuật toán:**
  - **char* fmtname(char *path)**

* Hàm fmtname giúp ta tìm ra tên file cuối cùng trong một đường dẫn (vd: /a/c khi đưa vào hàm fmtname sẽ có ra kết quả là c)
* Hàm fmtname sử dụng một vòng lặp để đưa con trỏ p tới địa chỉ cuối cùng của path sau đó di chuyển con trỏ p trỏ đến ô nhớ chứa kí tự bắt đầu của tên file.
* Sau đó, kiểm tra độ dài của chuỗi p xem chuỗi p có lớn hơn hoặc bằng DIRSIZ hay không, nếu có trả về p vì buffer không thể chứa đủ p, nếu không thì copy tên file vào biến buffer và gán ký hiệu null cho biến buffer sau đó trả về biến buffer.

– **void find(char *path, char* name)**

* Hàm find nhận vào 2 đối số là đường dẫn và tên file cần tìm và trả ra tất cả các đường dẫn đến file cần tìm bắt đầu từ đường dẫn path.
* Đầu tiên sẽ kiểm tra xem có thể mở đường dẫn path và lấy thông tin từ đường dẫn path không. Nếu không thì dừng tìm kiếm.
* Sau khi lấy được thông tin từ đường dẫn path ta sẽ kiểm tra xem nó là folder hay file. Nếu là file thì kiểm tra xem tên có giống với tên file cần tìm không, nếu có thì in đường dẫn ra màn hình và kết thúc. Nếu là folder:
  · Đầu tiên kiểm tra xem độ dài của đường dẫn có vượt quá độ dài cho phép hay chưa. Nếu vượt qua thì dừng tìm kiếm.
  · Sao chép nội dung của chuỗi path sang chuỗi buffer, sau đó chuyển vị trí của con trỏ p đến vị trí cuối của chuỗi buffer, tiếp theo thêm ký hiệu '/' vào cuối chuỗi.
  · Tiếp đến, hàm bắt đầu một vòng lặp để đọc nội dung từ fd vào de với kích thước bằng kích thước của de và so sánh với kích thước của de xem có bằng nhau không nếu có tiếp tục vòng lặp. Đối với mỗi de được đọc, nó kiểm tra xem inum của de đó có bằng không không. Nếu bằng không, có nghĩa là bản ghi thư mục không hợp lệ và hàm tiếp tục với vòng lặp tiếp theo.
  · Sau đó, hàm sao chép tên của bản ghi thư mục (de.name) vào bộ đệm buffer, bắt đầu từ vị trí được trỏ bởi p.
  · Sử dụng stat để thu thập thông tin của tệp hoặc thư mục được chỉ định bởi đường dẫn trong buffer và lưu trữ thông tin trong cấu trúc st. Nếu $< 0$ có nghĩa rằng không thế lấy thông tin inode của file có tên chứa trong biến buf và tiếp tục một vòng lặp mới.
  · Nếu có thể lấy thông tin, thì kiểm tra xem nó là file hay folder. Nếu là file thì kiểm tra xem tên có giống với tên file cần tìm không và in ra đường dẫn nếu tên giống với tên file cần tìm, nếu không thì bỏ qua và tiếp tục vòng lặp mới.
  · Nếu là folder kiểm tra tên folder khác "."và ".."để tránh việc lặp vô. Sau đó gọi đệ quy lại hàm find với 2 đối số là buffer và name.

– **int main(int argc, char *argv[])**

* Hàm main dùng để kiểm tra đối số đầu vào và gọi các hàm để thực hiện yêu cầu.
* Đầu tiên hàm kiểm tra đối số argc (số lượng đối số đầu vào), nếu $< 2$ trả về lỗi thiếutên thư mục cần tìm, nếu $= 2$ gọi hàm find với đường dẫn ban đầu là ".", nếu kháchai điều kiện trên thì có nghĩa có nhiều hơn 1 thư mục cần tìm nên hàm thực hiện1 vòng lặp để tìm hết toàn bộ thư mục.

## 3.5 Xargs

- **Mục tiêu:** Chương trình xargs trong hệ điều hành xv6 cho phép thực thi một lệnh với các đối số được đọc từ đầu vào chuẩn (stdin). Chương trình sẽ nhận đầu vào từng dòng, mỗi dòng sẽ được sử dụng như một đối số cho lệnh. Nó cũng hỗ trợ tùy chọn -n, cho phép giới hạn số lượng đối số được truyền cho mỗi lần thực thi lệnh.
- **Các bước thực hiện:**
  - **Bước 1:** Kiểm tra số lượng đối số
    - Chương trình kiểm tra xem người dùng có cung cấp đủ số lượng đối số hay không.
    - Nếu có tùy chọn -n, chương trình sẽ lấy giá trị số lượng đối số mà người dùng chỉ định.
  - **Bước 2:** Sao chép lệnh vào mảng đối số
    - Lệnh và các đối số được chỉ định bởi người dùng sẽ được sao chép vào mảng nargv, một mảng chứa các đối số cho lệnh thực thi.
  - **Bước 3:** Đọc đầu vào chuẩn
    - Chương trình đọc từng dòng từ đầu vào chuẩn (stdin) và lưu vào một bộ đệm (buf).
    - Sau mỗi dòng đầu vào, dòng này được thêm vào mảng nargv như một đối số cho lệnh
  - **Bước 4:** Thực thi
    - Khi đủ số lượng đối số (theo giá trị maxargs), chương trình sẽ thực thi lệnh với các đối số đã gom được. Quá trình thực thi được thực hiện bằng cách:
      - Tạo một tiến trình con bằng fork().
      - Thực thi lệnh trong tiến trình con bằng exec().
      - Tiến trình cha sẽ đợi tiến trình con hoàn thành với wait().
    - Nếu đầu vào kết thúc mà vẫn còn đối số chưa được thực thi, chương trình sẽ thực thi lệnh với các đối số này một lần cuối.
  - **Bước 5:** Tích hợp vào file Makefile
    - Chương trình xargs được thêm vào danh sách chương trình người dùng trong xv6 bằng cách chỉnh sửa Makefile. Chúng ta phải thêm tên chương trình xargs vào biến UPROGS trong Makefile để chương trình có thể được biên dịch và tích hợp vào hệ điều hành.
  - **Bước 6:** kiểm thử
    - Chương trình xargs được kiểm thử với các trường hợp như trong project yêu cầu
- **Thuật toán sử dụng**
  - Chương trình xargs sử dụng các thuật toán và kỹ thuật sau:
    - Thuật toán duyệt đầu vào:
      - Chương trình sử dụng vòng lặp để đọc từng ký tự từ stdin cho đến khi gặp dấu xuống dòng () hoặc EOF. Mỗi dòng đầu vào được coi là một đối số cho lệnh.
    - Fork-Exec-Wait:

- Đây là thuật toán phổ biến trong các hệ điều hành để tạo và quản lý tiến trình.
- Chương trình tạo một tiến trình con bằng fork(), sau đó tiến trình con thực thi lệnh bằng exec(), và tiến trình cha chờ con hoàn thành bằng wait()

* Tùy chọn -n:
  - Sử dụng điều kiện để kiểm soát số lượng đối số được truyền vào mỗi lần thực thi lệnh (maxargs). Điều này giúp tối ưu hóa số lượng đối số được truyền đi mà không làm tràn bộ nhớ

# Tài liệu

[1] Lab: Xv6 and Unix utilities, MIT

[2] nathan-chin/xv6-riscv-os/os/user/find.c

[3] Lab 1 Xv6 and Unix utilities | build a OS