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: System call

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, 11/2024

Mục lục

1 Giới thiệu

Trong hệ điều hành xv6, system call là phương thức mà các chương trình người dùng có thể yêu cầu các dịch vụ từ nhân hệ điều hành, cho phép tương tác trực tiếp với các tài nguyên phần cứng và thực hiện các thao tác quản lý hệ thống. Tương tự như Unix, xv6 cung cấp một loạt các system call cơ bản như quản lý tiến trình (với fork, exec, wait, exit), quản lý bộ nhớ (sbrk), và quản lý hệ thống tệp (open, read, write, close). Mỗi system call trong xv6 được định nghĩa trong file syscall.c, với các số hiệu syscall tương ứng được kernel ánh xạ qua một bảng hệ thống. Khi một tiến trình gọi system call, kernel sẽ xác định số hiệu, tìm hàm xử lý phù hợp, và thực hiện tác vụ theo yêu cầu của tiến trình. Điều này giúp xv6 đảm bảo tính cách ly và bảo mật cho các tài nguyên hệ thống, đồng thời cung cấp giao diện lập trình dễ sử dụng cho các nhà phát triển ứng dụng. Để hiểu sâu hơn về cách các system call hoạt động và quản lý tài nguyên trong xv6, báo cáo này chúng em sẽ đi vào tìm hiểu và phân tích hai tính năng quan trọng: trace và sysinfo. Đây là hai công cụ hữu ích giúp nhà phát triển và quản trị viên hệ thống kiểm soát, giám sát hoạt động của các tiến trình và nắm bắt tình trạng tài nguyên của hệ điều hành.

2 Phân công công việc

Bài tập	Họ và tên	Mức độ hoàn thành
Trace	Nguyễn Văn Hiến	100%
Cài đặt chương trình sysinfo	Trần Anh Tú	100%
Viết báo cáo Sysinfo	Nguyễn Anh Trí	100%
Tổng hợp báo cáo	Nguyễn Văn Hiến	100%

Bảng 1 – Bảng phân công công việc

3 Thực hiện

3.1 Chương trình Trace

- **Mục tiêu:** Chương trình trace trong hệ điều hành xv6 có mục tiêu chính là giám sát và ghi lại các system call mà một tiến trình thực hiện trong quá trình chạy, từ đó hỗ trợ các nhà phát triển trong việc quản lý và tối ưu hóa hệ thống. Thông qua trace, các lập trình viên có thể theo dõi chi tiết cách tiến trình tương tác với kernel, các tài nguyên được sử dụng, và thời điểm thực thi từng lệnh gọi. Điều này giúp phát hiện và gỡ lỗi nhanh chóng, cải thiện độ ổn định và hiệu năng của chương trình. Ngoài ra, trace còn hỗ trợ đánh giá và tối ưu hóa cách sử dụng tài nguyên, đảm bảo tiến trình thực thi tuân thủ quy tắc hệ thống, đồng thời phát hiện các hành vi bất thường hoặc truy cập trái phép, góp phần tăng cường bảo mật hệ điều hành.
- **Các bước thực hiện:**
  - **Bước 1:** Thêm prototype cho system call vào file user/user.h. Thêm một stub vào file user/usys.pl và một syscall number vào file kernel/syscall.h
    - **Lý do:**
      - Việc thêm prototype trong user/user.h giúp hệ thống nhận diện hàm trace trong không gian người dùng, cho phép người dùng gọi lệnh trace từ các ứng dụng.
      - Stub trong user/usys.pl đảm bảo rằng khi gọi trace, kernel có thể chuyển lệnh gọi hệ thống từ không gian người dùng vào kernel.
      - Số syscall trong kernel/syscall.h là ID duy nhất để kernel xác định và ánh xạ đúng lệnh trace, giúp điều hướng lời gọi đến chức năng tương ứng.
  - **Bước 2:** Tạo 1 file trace.c trong user để lấy các tham số đầu vào từ terminal và gọi system call trace.
    - **Lý do:** File trace.c cung cấp giao diện dòng lệnh để người dùng tương tác, nhập tham số, và kích hoạt system call trace. Đây là bước quan trọng giúp kiểm thử và dễ dàng quản lý các tham số đầu vào.
  - **Bước 3:** Sau đó khai báo $U/_trace vào phần UPROGS trong Makefile
    - **Lý do:** Thêm _trace vào UPROGS đảm bảo chương trình trace được biên dịch và liên kết vào hệ điều hành xv6. Điều này giúp lệnh trace khả dụng như một tiện ích trong không gian người dùng.
  - **Bước 4:** Thêm hàm sys_trace trong kernel/sysproc.c để thực hiện lệnh gọi hệ thống mới bằng cách ghi nhớ đối số của nó trong một biến mới trong cấu trúc Proc trong kernel/proc.h
    - **Lý do:** sys_trace thực hiện chức năng hệ thống của trace, ghi nhớ các đối số (ví dụ: mask và pid) vào một biến trong cấu trúc Proc. Điều này giúp lưu trữ trạng thái và dữ liệu theo dõi cho từng tiến trình, hỗ trợ theo dõi chính xác system call của từng tiến trình cụ thể.
  - **Bước 5:** Cập nhật fork trong kernel/proc.c để lan truyền dấu vết từ tiến trình cha sang tiến trình con
    - **Lý do:** Khi một tiến trình cha đã kích hoạt trace, tính năng trace cũng cần có hiệu lực cho các tiến trình con được tạo ra từ nó. Việc cập nhật fork đảm bảo thông tin về trace được truyền từ tiến trình cha, giúp duy trì nhất quán khi theo dõi tiến trình và tất cả các tiến trình con của nó.

- **Bước 6:** Cập nhật hàm syscall trong kernel/syscall.c để in kết quả.
  - **Lý do:** Hàm syscall là nơi kernel thực thi tất cả các system call. Việc cập nhật hàm này để in thông tin về các system call (như tên và tham số) khi trace được kích hoạt, giúp ghi lại chi tiết các lệnh gọi của tiến trình, từ đó cung cấp dữ liệu cần thiết cho việc theo dõi và phân tích.

3.2 Chương trình Sysinfo

- **Mục tiêu:** Phát triển system call sysinfo nhằm cung cấp thông tin về tình trạng hệ thống, bao gồm bộ nhớ trống (freemem) và số lượng tiến trình hiện tại (nproc). Điều này cho phép các ứng dụng người dùng truy vấn tài nguyên hệ thống, hỗ trợ giám sát và quản lý hiệu quả hơn.
- **Các bước thực hiện:**
  - **Bước 1:**Tạo file sysinfo.h trong thư mục kernel để định nghĩa cấu trúc sysinfo
    - **Lý do:** sysinfo.h giúp định nghĩa và chia sẻ cấu trúc sysinfo giữa các file trong kernel mà không cần phải định nghĩa lại, hỗ trợ việc tổ chức mã nguồn rõ ràng và dễ bảo trì khi cần mở rộng hoặc chỉnh sửa cấu trúc này.
  - **Bước 2:** Tạo file sysinfo.c và cài đặt hàm systeminfo
    - **Lý do:** Việc đặt logic cài đặt của hàm systeminfo trong sysinfo.c giúp giữ cho mã nguồn gọn gàng, theo chuẩn tách biệt giữa phần định nghĩa và phần cài đặt, đồng thời giúp các nhà phát triển dễ dàng duy trì và mở rộng khi cần thay đổi logic của systeminfo.
  - **Bước 3:** Thêm hàm sys_sysinfo vào sysproc.c làm hàm wrapper cho systeminfo
    - **Lý do:** sysproc.c là file chịu trách nhiệm chính cho các hàm system call trong kernel. Việc đặt sys_sysinfo tại đây giúp duy trì tính nhất quán của mã nguồn và tổ chức hợp lý giữa các lệnh gọi hệ thống.
  - **Bước 4:** Thêm hàm nproc vào proc.c để đếm số tiến trình
    - **Lý do:** proc.c là file chính quản lý tiến trình của kernel. Đặt hàm nproc() trong proc.c đảm bảo logic đếm tiến trình được quản lý tại một nơi duy nhất, giúp dễ dàng sửa đổi hoặc tối ưu khi cần thay đổi cách đếm tiến trình.
  - **Bước 5:** Thêm hàm freemem vào kalloc.c để tính bộ nhớ trống
    - **Lý do:** kalloc.c chịu trách nhiệm quản lý bộ nhớ, do đó đặt hàm freemem() ở đây cho phép tập trung các hàm liên quan đến bộ nhớ tại một file duy nhất. Nếu sau này có thay đổi về cách quản lý bộ nhớ, chỉ cần chỉnh sửa trong kalloc.c mà không ảnh hưởng đến các phần khác.
  - **Bước 6:** Khai báo hàm và cấu trúc mới trong defs.h
    - **Lý do:** defs.h là nơi tập trung khai báo các hàm và cấu trúc dùng chung trong kernel. Đưa các khai báo như freemem(), nproc(), và systeminfo(uint64) vào đây cho phép tất cả các file trong kernel dễ dàng gọi đến các hàm này, đồng thời tăng tính tiện lợi và đảm bảo mã nguồn nhất quán.
  - **Bước 7:** Thêm định nghĩa syscall trong syscall.h và ánh xạ vào syscall.c
    - **Lý do:** syscall.h định nghĩa mã số cho mỗi system call, còn syscall.c ánh xạ các mã này với các hàm tương ứng. Điều này cho phép kernel nhận diện và thực hiện system call chính xác khi được gọi từ không gian người dùng.

- **Bước 8:** Chuyển sang thư mục user để thực hiện các bước trong không gian người dùng
  - **Lý do:** Tách biệt mã nguồn giữa kernel và user space giúp dễ tổ chức và quản lý, đảm bảo các chương trình người dùng chỉ cần gọi system call mà không cần truy cập trực tiếp vào mã nguồn kernel.
- **Bước 9:** Định nghĩa struct và con trỏ struct sysinfo trong user.h
  - **Lý do:** Định nghĩa struct sysinfo và con trỏ của nó trong user.h giúp chương trình người dùng truyền con trỏ đến kernel để lấy thông tin hệ thống, tiết kiệm bộ nhớ và cho phép truyền dữ liệu dễ dàng giữa kernel và user space.
- **Bước 10:** Tạo file sysinfotest.c để kiểm tra tính đúng đắn của sysinfo
  - **Lý do:** Việc tạo một file kiểm tra riêng cho sysinfo giúp phát hiện lỗi sớm và đảm bảo chất lượng của system call, đồng thời không ảnh hưởng đến phần cài đặt chính trong kernel.
- **Bước 11:** Thêm entry vào usys.pl để kernel nhận diện system call sysinfo
  - **Lý do:** usys.pl tạo mã tự động để ánh xạ các system call từ không gian người dùng tới kernel, thêm entry("sysinfo") giúp việc gọi sysinfo từ chương trình người dùng đơn giản hơn.
- **Bước 12:** Cập nhật Makefile để biên dịch sysinfo và sysinfotest
  - **Lý do:** Thêm sysinfo.o và _sysinfotest vào Makefile giúp các file mới được biên dịch và liên kết vào kernel một cách tự động, đảm bảo khi build kernel, các tính năng mới sẵn sàng sử dụng.

Tài liệu

[1] Lab: system calls, MIT

[2] HƯỚNG DẪN LAB 02 SYSTEM CALL, GV: Lê Viết Long