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

Báo cáo bài tập

Đề tài: xv6

Môn học: Operating System

Sinh viên thực hiện:

Phạm Quốc Nam Anh	23120111
Nguyễn Xuân Duy	23120121
Nguyễn Minh Hiếu	23120124
Huỳnh Thái Toàn	23120175

Giáo viên hướng dẫn:

Lê Viết Long

Ngày 13 tháng 12 năm 2025

Mục lục

1 Tăng tốc syscall

1.1 Mô tả bài toán

Tối ưu hóa lệnh gọi hệ thống `getpid()` bằng cách chia sẻ một vùng nhớ chỉ đọc giữa kernel và user. Thay vì trap vào kernel mỗi lần gọi getpid, tiến trình user có thể đọc trực tiếp PID từ một trang bộ nhớ đặc biệt đã được ánh xạ sẵn. Trang này chứa một struct `usyscall` với trường `pid` lưu PID của tiến trình hiện tại.
Ngoài ra, viết chương trình kiểm tra `pidget()` để so sánh giá trị trả về từ getpid() (system call thông thường) và ugetpid() (đọc trực tiếp từ vùng chia sẻ). Bài tập đạt yêu cầu nếu hai giá trị này trùng nhau.

1.2 Phương pháp thực hiện

1. Trong `proc_pagetable()`, khởi tạo page table mới và ánh xạ các vùng bắt buộc (trampoline, trapframe).
2. Trong `allocproc()`, cấp phát và khởi tạo tiến trình, đồng thời xử lý cấp phát và ánh xạ trang `USYSCALL`.
3. Trong `freeproc()`, giải phóng tài nguyên tiến trình, bao gồm cả trang `USYSCALL` nếu tồn tại.

1.2.1 Khởi tạo page table

Trong hàm `proc_pagetable()`:

- Tạo một page table mới bằng `uvmcreate()`.
- Ánh xạ `trampoline` và `trapframe` vào không gian địa chỉ user.
- Đảm bảo các ánh xạ này có quyền truy cập phù hợp.

1.2.2 Cấp phát và khởi tạo tiến trình

Trong hàm `allocproc()`:

- Tìm một tiến trình rảnh trong mảng `proc[]` và khởi tạo các trường cơ bản (PID, trạng thái, trapframe).

- Cấp phát một trang cho `usyscall` bằng `kalloc()`.
- Khởi tạo trường `pid` trong `struct usyscall`.
- Ánh xạ địa chỉ ảo `USYSCALL` tới trang vật lý chứa `struct usyscall` bằng `mappages()` với quyền chỉ đọc cho user.

1.2.3 Giải phóng tiến trình

Trong hàm `freeproc()`:

- Giải phóng vùng nhớ được cấp cho `usyscall` nếu tồn tại.
- Đặt lại trạng thái tiến trình về `UNUSED`.

Trong hàm `proc_freepagetable()`:

- Bỏ ánh xạ vùng `USYSCALL` khỏi page table bằng `uvmunmap()`.
- Giải phóng toàn bộ page table của tiến trình.

2 In bảng trang

2.1 Mô tả bài toán

Viết hàm `vmprint()` để in ra nội dung của bảng trang (page table) trong hệ thống RISC-V. Hàm này giúp hình dung cấu trúc phân cấp của bảng trang và hỗ trợ gỡ lỗi trong tương lai.
Hàm `vmprint()` nhận vào một tham số kiểu `pagetable_t` và in bảng trang đó theo định dạng phân cấp. Mỗi mức của bảng trang được thể hiện bằng các dấu chấm (`..`), cho biết độ sâu của PTE trong cấu trúc cây. Chỉ các PTE hợp lệ mới được in ra, bao gồm chỉ số PTE, giá trị PTE (với các bit quyền truy cập), và địa chỉ vật lý tương ứng.
Ngoài ra, chèn lệnh gọi `vmprint()` trong `exec.c` để in bảng trang của tiến trình đầu tiên (PID = 1) ngay trước khi hàm `exec()` trả về. Bài tập đạt yêu cầu khi vượt qua bài kiểm tra "pte printout" của lệnh `make grade`.

2.2 Phương pháp thực hiện

1. Trong `kernel/vm.c`, viết hàm đệ quy `vmprintlevel()` để duyệt qua các cấp của bảng trang.
2. Viết hàm `vmprint()` để in địa chỉ bảng trang và gọi hàm đệ quy.
3. Trong `kernel/defs.h`, khai báo prototype của hàm `vmprint()`.
4. Trong `kernel/exec.c`, chèn lệnh gọi `vmprint()` cho tiến trình có PID = 1.

2.2.1 Duyệt qua bảng trang

Trong hàm `vmprintlevel()`:

- Duyệt qua 512 PTE trong bảng trang hiện tại.
- Chỉ xử lý các PTE hợp lệ (có bit `PTE_V` được set).
- In ra độ sâu (số lượng `..`), chỉ số PTE, giá trị PTE, và địa chỉ vật lý.
- Phân biệt PTE là bảng trang con (không có bit R|W|X) hay là trang lá (có ít nhất một bit R|W|X).
- Nếu PTE trỏ đến bảng trang con, đệ quy xuống cấp tiếp theo với độ sâu tăng thêm 1.

2.2.2 In bảng trang

Trong hàm `vmprint()`:

- In dòng đầu tiên chứa địa chỉ của bảng trang: `"page table 0x..."`.
- Gọi `vmprintlevel()` với tham số bảng trang và độ sâu ban đầu là 0.
- Sử dụng macro `PTE2PA()` từ `kernel/riscv.h` để chuyển đổi PTE thành địa chỉ vật lý.

2.2.3 Khai báo và sử dụng

Trong hàm `kernel/defs.h`:

- Thêm dòng khai báo `void vmprint(pagetable_t);` vào phần khai báo các hàm của `vm.c`.

Trong hàm `exec()` của `kernel/exec.c`:

- Sau khi gán `p->pagetable = pagetable` và trước câu lệnh `return argc`.
- Thêm điều kiện kiểm tra: `if(p->pid == 1) vmprint(p->pagetable)`.
- Điều này đảm bảo chỉ in bảng trang của tiến trình init (tiến trình đầu tiên).

3 Các trang đã được truy cập

3.1 Mô tả bài toán

Trong các hệ điều hành hiện đại, việc quản lý bộ nhớ hiệu quả đóng vai trò quan trọng đối với hiệu năng hệ thống. Một số bộ thu gom rác (garbage collector) và cơ chế quản lý bộ nhớ nâng cao có thể tận dụng thông tin về việc các trang bộ nhớ đã được truy cập (đọc hoặc ghi) để đưa ra các quyết định tối ưu, chẳng hạn như ưu tiên giữ lại các trang thường xuyên được sử dụng hoặc thu hồi các trang ít được truy cập.

Trên kiến trúc RISC-V, phần cứng hỗ trợ việc theo dõi truy cập bộ nhớ thông qua các bit trạng thái trong Page Table Entry (PTE). Cụ thể, bit `PTE_A` (Accessed bit) sẽ được phần cứng tự động thiết lập khi một trang bộ nhớ được truy cập và xảy ra TLB miss. Tuy nhiên, thông tin này không được cung cấp trực tiếp cho không gian người dùng.

Bài toán đặt ra là mở rộng hệ điều hành xv6 bằng cách cài đặt một lệnh gọi hệ thống mới, cho phép chương trình người dùng truy vấn các trang bộ nhớ đã được truy cập kể từ lần kiểm tra trước đó. Lệnh gọi hệ thống này phải đọc trạng thái bit truy cập trong bảng trang, tổng hợp kết quả và trả về cho không gian người dùng dưới dạng một bitmask.

3.2 Phương pháp thực hiện

3.2.1 Xử lý tham số hệ thống

Trong hàm `sys_pgaccess()`, các tham số được trích xuất từ không gian người dùng bằng các hàm hỗ trợ của xv6:

- `argaddr()` để lấy địa chỉ ảo bắt đầu và địa chỉ bộ đệm kết quả.
- `argint()` để lấy số lượng trang cần kiểm tra.

Để đảm bảo an toàn và tránh quét quá nhiều trang, số lượng trang được giới hạn ở một giá trị tối đa cho phép.

3.2.2 Duyệt bảng trang và kiểm tra bit truy cập

Hệ điều hành sử dụng hàm `walk()` trong `kernel/vm.c` để truy xuất Page Table Entry tương ứng với từng địa chỉ ảo của các trang cần kiểm tra. Với mỗi PTE hợp lệ:

- Kiểm tra bit `PTE_A` để xác định xem trang đã được truy cập hay chưa.
- Nếu bit `PTE_A` được thiết lập, bit tương ứng trong bitmask sẽ được đặt.

Bit `PTE_A` được định nghĩa trong `kernel/riscv.h` dựa trên đặc tả của kiến trúc RISC-V.

3.2.3 Xóa bit truy cập sau khi kiểm tra

Sau khi phát hiện một trang đã được truy cập, bit `PTE_A` trong PTE sẽ được xóa. Việc này đảm bảo rằng:

- Các lần gọi `pgaccess()` tiếp theo chỉ phản ánh các truy cập mới.
- Thông tin trả về luôn thể hiện chính xác các trang được truy cập kể từ lần kiểm tra gần nhất.

3.2.4 Trả kết quả về không gian người dùng

Bitmask kết quả được lưu tạm thời trong một biến của kernel. Sau khi hoàn tất quá trình quét:

- Hàm `copyout()` được sử dụng để sao chép bitmask từ không gian kernel sang bộ đệm trong không gian người dùng.
- Việc sao chép này đảm bảo tính an toàn bộ nhớ giữa kernel và user space.

—