TRƯỜNG ĐẠI HỌC KHOA HỌC TỰ NHIÊN, ĐHQG-HCM
Khoa Công nghệ Thông Tin

Môn học: Hệ Điều Hành

BÁO CÁO PROJECT 3 – XV6 LABS: PAGE TABLES

GV hướng dẫn: Lê Giang Thanh
Trần Trung Dũng

SV thực hiện:

MSSV	Họ và Tên
21120056	Nguyễn Đặng Tường Duy
21120462	Đỗ Khải Hưng

Mục lục

0.1 Thông tin môn học . . . 2
0.2 Thông tin lab . . . 2

1 Introduction **3**
1.1 xv6 là gì? . . . 3
1.2 Lab: Page tables . . . 3
1.2.1 Mục tiêu . . . 3
1.2.2 Nội dung . . . 3
1.3 Cài đặt môi trường . . . 3

2 Exercises **3**
2.0 Prerequisite . . . 3
2.1 Speed up system calls . . . 4
2.1.1 Mô tả đề bài và gợi ý . . . 4
2.1.2 Cách làm và Giải thích . . . 4
2.1.3 Lưu ý . . . 4
2.2 Print a page table . . . 4
2.2.1 Mô tả đề bài và gợi ý . . . 4
2.2.2 Cách làm và Giải thích . . . 5
2.3 Detect which pages have been accessed . . . 5
2.3.1 Mô tả đề bài và gợi ý . . . 5
2.3.2 Cách làm và Giải thích . . . 5

3 Kết quả đạt được **6**

Thông tin lab và thông tin môn học

0.1 Thông tin môn học

- Tên môn học: Hệ điều hành.
- Khoa: Khoa Công nghệ Thông tin, Trường Đại học Khoa học Tự nhiên, ĐHQG-HCM.
- Giảng viên: Trần Trung Dũng, Lê Giang Thanh.
- Lớp: 21 Cử nhân Tài năng.

0.2 Thông tin lab

- Lab: Page tables
- Người hướng dẫn: Trần Trung Dũng, Lê Giang Thanh.
- Hạn nộp: 24/12/2023.
- Nhóm thực hiện: Nhóm gồm các thành viên:
  1. Nguyễn Đặng Tường Duy – 21120056
  2. Đỗ Khải Hưng – 21120462

1 Introduction

1.1 xv6 là gì?

Xv6 là một hệ điều hành thời gian thực đơn giản được phát triển dựa trên Unix V6. Xv6 được sử dụng trong nhiều khóa học về hệ điều hành và hệ thống tại các trường đại học, bao gồm cả khóa học "6.828: Operating System Engineering" tại MIT.

Xv6 được viết bằng ngôn ngữ lập trình C và được thiết kế để hoạt động trên kiến trúc máy tính x86. Xv6 có cấu trúc đơn giản nhưng cung cấp một số tính năng quan trọng của một hệ điều hành như quản lý tiến trình, quản lý bộ nhớ ảo, quản lý tệp, và hỗ trợ mạng cơ bản.

Mục tiêu của xv6 là giúp sinh viên hiểu rõ cách hoạt động của một hệ điều hành, từ việc lên lịch thực thi tiến trình đến giao tiếp với các thiết bị và tệp tin. Xv6 cung cấp một môi trường thí nghiệm lý tưởng để nghiên cứu và thực hành các khái niệm hệ điều hành cơ bản.

Ngoài ra, xv6 cũng đi kèm với các bài thực hành, như lab "Lab: Page tables" để giúp sinh viên làm quen với hệ điều hành và phát triển kỹ năng lập trình hệ thống.

1.2 Lab: Page tables

1.2.1 Mục tiêu

Lab: Page tables là một bài thực hành của khóa học 6.828: Operating System Engineering tại MIT. Mục tiêu của bài thực hành này là giúp sinh viên hiểu rõ hơn về bảng trang hay page table, một cơ chế quan trọng của hệ điều hành để có thể cung cấp tài nguyên bộ nhớ đến từng tiến trình, nó định nghĩa về địa chỉ bộ nhớ và cách bộ nhớ vật lý được truy xuất như thế nào.

1.2.2 Nội dung

Trong bài thực hành này, sinh viên sẽ được yêu cầu:

- Tìm hiểu về bảng trang – page tables trong xv6.
- Tìm hiểu về cách cài đặt bảng trang để tăng tốc cho việc gọi system call.
- Tìm hiểu về cách cài đặt bảng trang để kiểm tra xem trang (page) nào đang được truy cập đến.

1.3 Cài đặt môi trường

Nhóm chúng em sử dụng máy ảo Linux với bản phân phối là Ubuntu, do đó, để cài đặt được hệ điều hành xv6, ta cần làm các bước như sau:

- Trong giao diện của Ubuntu, ta mở Terminal.
- Chạy lệnh: `sudo apt-get install git build-essential gdb-multiarch qemu-system-misc gcc-riscv64-linux-gnu binutils-riscv64-linux-gnu` để cài đặt các công cụ giúp hệ điều hành xv6 có thể chạy được.
- Ta cũng cần cài đặt Git, bằng cách chạy lệnh: `sudo apt-get install git`.

Chi tiết về cách thiết lập môi trường cho các hệ điều hành khác có thể tham khảo ở: `https://pdos.csail.mit.edu/6.828/2023/tools.html`.

2 Exercises

2.0 Prerequisite

Trước khi bắt đầu, ta bật Terminal trong folder `xv6-labs-2023` và thực hiện các bước sau:

```
$ git fetch

$ git checkout pgtbl

$ make clean
```

2.1 Speed up system calls

2.1.1 Mô tả đề bài và gợi ý

- **Mô tả đề bài:** Đề bài yêu cầu tối ưu hóa các system call bằng cách chia sẻ dữ liệu trong vùng chỉ đọc giữa user space và kernel. Ở đây yêu cầu tối ưu lời `getpid()` system call
- **Các gợi ý cho bài toán**:
  - Cấu trúc usyscall được lưu trữ trong file `memlayout.h`, hãy khởi tạo nó với `pid` của process hiện tại.
  - Chọn những bit mà người dùng được phép đọc trong trang.
  - Cần quản lý vòng đời của trang.

2.1.2 Cách làm và Giải thích

- Trang `usyscall` : Cấp phát một trang địa chỉ ảo bổ sung cho mỗi tiến trình. Vì vậy, ở đây lưu một cấu trúc `usyscall` trong `proc` struct.
- Khởi tạo trang `usyscall`: Thêm cấu trúc trang này trong `proc.c`. Trong allocproc(), cấp phát thêm cho `usyscall` bằng hàm `kalloc()`.
- Gán giá trị: Sau khi cấp phát, gán giá trị pid trong trang `usyscall` bằng giá trị id của process hiện tại.
- `usyscall` mapping: Vì việc địa chỉ ở cấp độ người dùng yêu cầu dịch thông qua phần cứng của bảng trang, nên `usyscall` cũng cần được ánh xạ trên bảng trang của tiến trình. Thêm logic mapping trong `proc_pagetable()` bằng hàm `mappages`.
- Giải phóng trang `usyscall`: Khi tiến trình được giải phóng, trang này cũng sẽ được giải phóng. Thêm logic giải phóng vùng nhớ vào `freeproc()`.
- Hủy mapping trang `usyscall`: Trong `proc_freepagetable()`, hủy ánh xạ trang `usyscall`.

2.1.3 Lưu ý

- Ở đây vì chỉ tối ưu system call `getpid()` nên trong struct `usyscall` chỉ lưu `pid`.
- Để có thể truy cập đến cấu trúc `usyscall` đã được định nghĩa cần thêm ở đầu file `proc.c`

```
#ifndef LAB_PGTBL
#define LAB_PGTBL
#endif
```

- Vùng địa chỉ của trang đã được định nghĩa trong `memlayout.h` với hằng số `USYSCALL`, dùng nó để quản lý vùng nhớ của trang.
- Vì vùng nhớ được cấp cho trang `usyscall` tương tự như vùng `trapframe`, do đó có thể tham khảo để viết tương tự.

2.2 Print a page table

2.2.1 Mô tả đề bài và gợi ý

- **Mô tả đề bài:** Hãy viết một hàm `vmprint()` nhận đối số là `pagetable_t` và in ra nội dung của bảng trang theo định dạng yêu cầu. Chèn đoạn mã

```
if(p->pid==1) vmprint(p->pagetable)
```

  vào `exec.c` trước câu lệnh `return argc` để in bảng trang của tiến trình đầu tiên.
- **Các gợi ý cho bài toán:**
  - Duyệt hết tất cả các PTEs, với mỗi PTE mà có chứa các entries con, duyệt hêt tất cả các entries con đó.
  - Đặt hàm `vmprint()` trong file `kernel/vm.c`.

- Sử dụng các macros được định nghĩa trong file `kernel/riscv.h` để truy cập các trường trong cấu trúc `pagetable_t`.
- Xem xét hàm `freewalk` để lấy ý tưởng cho cách duyệt qua các mức của bảng trang.
- Định nghĩa prototype cho `vmprint` trong tệp kernel/defs.h để có thể gọi nó từ exec.c.
- Khi sử dụng printf, sử dụng %p để in đầy đủ 64-bit hex PTEs và địa chỉ.
- In mỗi PTE một cách đẹp, với các dấu " .." cho biết sâu của nó trong cây. Không in các PTE không hợp lệ. Đối với mỗi trang lá trong output của `vmprint`, giải thích nó chứa những gì logic và giải thích các bit được phép truy cập của nó.

2.2.2 Cách làm và Giải thích

- Vì mỗi độ sâu cần in dấu ".." tương ứng nên cần cài đặt thêm 1 hàm `vmprintwalk` để lặp và in ra độ sâu tương ứng, đồng thời sử dụng đệ quy để in các entry con.
- Duyệt hết 512 entries của bảng trang, chỉ xét những entry hợp lệ qua macro `PTE_V`.
- In độ sâu và thông tin của entry gồm : index của entry trong bảng, bits của entry và địa chỉ vật lý của nó.
- Kiểm tra có tồn tại entry con hay không qua các macros.

```
if(pte & (PTE_R | PTE_W | PTE_X)) == 0)
```

  Nếu tồn tại, convert PTE sang địa chỉ vật lý và truyền tham số `pagetable_t` và độ sâu tương ứng để tiếp tục đệ quy.
- Ở lời gọi hàm `vmprint` gọi đến hàm `vmprintwalk` với độ sâu bằng 0 để duyệt.

2.3 Detect which pages have been accessed

2.3.1 Mô tả đề bài và gợi ý

- Trong bài tập này ta cần thêm vào cho xv6 một chức năng dùng để phát hiện việc trang (page) nào được truy cập đến và báo cáo thông tin về việc này đến không gian người dùng (userspace) thông qua việc kiểm tra các access bits (PTE_A) trong bảng trang của RISC-V.
  Ta cần cài đặt một system call để làm được chức năng này.
- Các gợi ý cho bài toán:
  - Hiểu cách system call `pgaccess` được sử dụng, trong file `user/pgtbltest.c`.
  - Ta cần dùng các hàm `argint()` và `argaddr()` để đọc các tham số như địa chỉ (virtual) bắt đầu ở trang đầu tiên, số trang cần kiểm tra và địa chỉ trong user space để lưu kết quả.
  - Ta có thể dùng hàm `copyout()` để sao chép bitmask kết quả từ kernel space sang user space.
  - Hàm `walk()` trong file `kernel/vm.c` giúp việc các PTEs trong bảng trang dễ dàng hơn.
  - Ta cũng cần phải định nghĩa giá trị cho PTE_A hay access bits trong file `kernel/riscv.h`, đồng thời ta cần reset access bit sau mỗi lần gọi system call `pgaccess` vì nếu không ta sẽ không thể biết được trang có được truy xuất đến trong lần gọi trước hay không.
  - Hàm `vmprint()` được cài đặt ở trên có thể hữu ích trong việc debug.

2.3.2 Cách làm và Giải thích

- Theo như các gợi ý được cho trước, ta có thể định nghĩa cho giá trị của PTE_A trong file `kernel/riscv.h` là:

```
#define PTE_A (1L << 6)
```

  theo như trong tài liệu **RISC-V privileged architecture manual** có chỉ ra.
- Ta dùng các hàm `argaddr()` và `argint()` để đọc các tham số (argument) từ stack của không gian địa chỉ của tiến trình.
- Ta được địa chỉ ảo bắt đầu của trang đầu tiên là `curAddr`, số trang cần duyệt là `numPages` và user address `userAddr` để lưu kết quả trong bảng trang.

- Ta cũng dùng `bitmask` như là một buffer trong kernel nhằm lưu kết quả và để trả về user bằng hàm `copyout`.
- Ta duyệt qua `numPages` trang và dùng hàm `walk()` để lấy địa chỉ của PTE cần tìm trong từng bảng trang đang xét.
- Khi đã có được địa chỉ của PTE nói trên, ta kiểm tra xem access bit của nó hay PTE_A có bằng 1 hay không bằng phép logic AND:

```
if ((*pte) & PTE_A)
```

- Nếu access bit là 1 thì ta tiến hành dịch trái `i` bits trong `bitmask`:

```
bitmask |= (1 << i);
```

  và reset access bit để cho lần duyệt tiếp theo:

```
*pte &= ~(PTE_A);
```

- Cuối cùng ta dùng hàm `copyout()` để gửi kết quả từ kernel về user, lưu vào `userAddr` đã nói ở trên.

3 Kết quả đạt được

Bằng các cách làm trên, chúng em đã triển khai các chương trình và dùng lệnh chấm điểm `make grade`.
Kết quả đạt được đều pass các test của hệ thống:

```
riscv64-linux-gnu-objdump -t kernel/kernel | sed '1,/SYMBOL TABLE/d; s/ .* / /;
/^$/d' > kernel/kernel.sym
make[1]: Leaving directory '/home/ndtduy/xv6-labs-2023'
== Test pgtbltest ==
$ make qemu-gdb
(5.5s)
== Test   pgtbltest: ugetpid ==
  pgtbltest: ugetpid: OK
== Test   pgtbltest: pgaccess ==
  pgtbltest: pgaccess: OK
== Test pte printout ==
$ make qemu-gdb
pte printout: OK (1.0s)
== Test answers-pgtbl.txt ==
answers-pgtbl.txt: FAIL
    Cannot read answers-pgtbl.txt
== Test usertests ==
$ make qemu-gdb
(239.3s)
== Test   usertests: all tests ==
  usertests: all tests: OK
== Test time ==
time: FAIL
    Cannot read time.txt
Score: 40/46
make: *** [Makefile:338: grade] Error 1
ndtduy@LENOVO:~/xv6-labs-2023$
```