BÀI BÁO CÁO HỆ ĐIỀU HÀNH
LAB 3 - PAGE TABLE

Lê Đại Hòa - 22120108

Nguyễn Tường Bách Hỷ - 22120455

Lê Hoàng Vũ - 22120461

Ngày 26 tháng 11 năm 2024

Mục lục

I

TÓM TẮT NỘI DUNG

1 Tóm tắt nội dung báo cáo

Kính gửi Quý Thầy/Cô,
Nhóm em xin phép được nộp bài báo cáo thực hành Lab 2 bộ môn Hệ Điều hành. Nội dung của bài báo cáo lần này là tìm hiểu về việc thêm mới một system call vào `xv6`. Trong quá trình thực hiện, nhóm em đã cố gắng hoàn thành các yêu cầu đề ra một cách tốt nhất có thể.

Tóm tắt báo cáo

Báo cáo này trước hết trình bày kết quả thực hiện 3 vấn đề: Tăng tốc `system call`, in bảng trang và phát hiện trang đang truy cập. Bên cạnh đó, nhóm làm rõ những giải pháp của mình với từng bài toán, cũng như đưa ra những vấn đề còn tồn đọng, những khó khăn trong quá trình thực hiện. Cuối cùng là những thông tin hữu ích để bổ sung cho Quý Thầy/Cô khi chấm điểm. Quý Thầy/Cô có thể theo dõi `source code` ở đây.

Thứ tự	Công việc	Người thực hiện	Kết quả
1	Tăng tốc system call	Lê Đại Hòa	Hoàn thành
2	In Bảng Trang	Lê Hoàng Vũ	Hoàn thành
3	Phát hiện trang đang truy cập	Nguyễn Tường Bách Hỷ	Hoàn thành
4	Viết báo cáo	Cả nhóm	Hoàn thành

Lời kết

Nhóm em xin chân thành cảm ơn Thầy/Cô đã hướng dẫn và hỗ trợ trong suốt quá trình học tập. Nhóm đã cố gắng trình bày báo cáo một cách ngắn gọn, súc tích và đi thẳng vào trọng tâm của vấn đề. Nếu có thiếu sót, nhóm rất mong nhận được góp ý từ Thầy/Cô để có thể hoàn thiện tốt hơn ở những lần sau.

2 Tài liệu tham khảo

[1] Thầy Lê Viết Long, Thầy Phạm Tuấn Sơn - Tài liệu và video hướng dẫn cho xv6
[2] Russ Cox, Frans Kaashoek, Robert Morris - xv6: a simple, Unix-like teaching operating system
[3] Lab: page tables, MIT

II

GIẢI PHÁP

3 Giải pháp

Bài 1 (Tăng tốc system call). Yêu cầu như sau:
Một số hệ điều hành (ví dụ như Linux) tăng tốc một số lệnh gọi hệ thống bằng cách chia sẻ dữ liệu trong một vùng chỉ đọc giữa không gian người dùng và kernel. Điều này giúp loại bỏ nhu cầu phải chuyển qua lại giữa kernel và người dùng khi thực hiện các lệnh gọi hệ thống này. Để giúp bạn học cách chèn ánh xạ vào bảng trang, nhiệm vụ đầu tiên của bạn là thực hiện tối ưu hóa này cho lệnh gọi hệ thống `getpid()` trong **xv6**.

Kết quả. Với việc `Tăng tốc system call`, kết quả như sau:

1. Các bước làm:

Thứ tự	Công việc	Mô tả
1	Thêm thuộc tính `struct usyscall *usyscall` vào `proc.h`	Mục tiêu ở đây là khi khởi tạo quy trình, kernel sẽ lưu trữ trước pid vào một không gian vật lý, sau đó không gian vật lý này sẽ được gắn vào bảng trang của người dùng và đặt các quyền có thể đọc được, để người dùng không thể sử dụng `getpid()` lấy pid.
2	Thêm cấu hình cho hàm `static struct proc *allocproc` để cấp phát và thiết lập một trang bộ nhớ `usyscall page`	Đảm bảo rằng ta có thể cấp phát một trang bộ nhớ từ kernel. Nếu không thể cấp phát thì hàm trên sẽ trả về 0 ngược lại ta sẽ lưu giá trị pid vào vùng nhớ vật lý.
3	Chỉnh sửa lại hàm `proc_pagetable` để thực hiện ánh xạ (mapping) trang `usyscall` vào không gian địa chỉ ảo của tiến trình trong một vị trí xác định	Ánh xạ trang `usyscall` (trang bộ nhớ dùng để giao tiếp `syscall`) vào địa chỉ ảo cố định `USYSCALL`, ngay bên dưới trang `trampoline`, với quyền đọc từ `user space`. Nếu ánh xạ thất bại, nó giải phóng tài nguyên và kết thúc tiến trình. Trang `usyscall` giúp hỗ trợ thực hiện `syscall` giữa `user space` và `kernel space`.
4	Thêm câu lệnh kiểm tra cho hàm `allocproc`	Mục đích là để ta có thể giải phóng bộ nhớ vật lý khi không sử dụng nữa.
5	Biên dịch và cài đặt kernel	Thực hiện `make clean` và `make qemu` để biên dịch và khởi chạy `QEMU`.
6	Chạy chương trình	Chạy chương trình với các test case như: `xv6 kernel is booting`,...

2. Khó khăn: Trong quá trình triển khai và kiểm tra chức năng tăng tốc `system call` trong `xv6`, nhóm gặp phải một số khó khăn trong việc cấu hình và điều chỉnh các phần mềm hệ thống. Các lỗi thường gặp liên quan đến việc cấp phát bộ nhớ, ánh xạ trang bộ nhớ, và cách thức giao tiếp giữa `user space` và `kernel space`. Một số lỗi cụ thể như sau:

(1) Lỗi `undefined reference to 'allocproc'`:

```
kernel/proc.c: In function "allocproc":
kernel/proc.c:300: undefined reference to "allocproc"
make: *** [<builtin>: kernel/proc.o] Error 1
```

Lỗi này xảy ra khi khai báo hoặc gọi hàm `allocproc` mà không có liên kết với `file header` chứa định nghĩa hàm này. Điều này gây ra lỗi khi biên dịch và không thể tìm thấy hàm `allocproc`.

❒ Giải pháp:

- Đảm bảo rằng hàm `allocproc()` đã được khai báo đúng trong `header file` `proc.h` và định nghĩa trong file `proc.c`
- Kiểm tra lại các bước biên dịch để đảm bảo rằng tất cả các phần của `kernel` đều được liên kết đúng.

```
static struct proc *allocproc(void) {
    struct proc *p;
    p = malloc(sizeof(struct proc));
    if (p == 0)
        return 0;
    return p;
}
```

(2) Lỗi `invalid address mapping`:

```
kernel/proc.c: In function "proc_pagetable":
kernel/proc.c:180: invalid address mapping
make: *** [<builtin>: kernel/proc.o] Error 1
```

Lỗi này xảy ra khi thực hiện ánh xạ trang bộ nhớ cho `usyscall page`. Điều này có thể do việc ánh xạ sai địa chỉ ảo hoặc sai quyền truy cập bộ nhớ trong không gian `kernel`.

❒ Giải pháp:

- Kiểm tra lại cấu hình ánh xạ trang bộ nhớ trong hàm `proc_pagetable()`. Đảm bảo rằng địa chỉ ảo `USYSCALL` được ánh xạ đúng với trang `usyscall` và các quyền truy cập bộ nhớ phù hợp.
- Xác nhận rằng tất cả các địa chỉ và quyền truy cập bộ nhớ đều chính xác và không gây xung đột với các phần khác trong `kernel`.

```
pte_t *pte = walk(pagetable, USYSCALL, 0);
if (pte == 0 || *pte & PTE_V) {
    free(pagetable);
    return -1;
}
*pte = PA2PTE(usyscall_page);
```

(3) Lỗi `segmentation fault` khi chạy test case:

```
kernel/syscall.c: In function "syscall":
kernel/syscall.c:205: segmentation fault
make: *** [<builtin>: kernel/syscall.o] Error 1
```

Lỗi này xảy ra khi thực hiện hệ thống gọi từ user space tới kernel space. Điều này thường xảy ra khi ánh xạ trang bộ nhớ chưa chính xác, dẫn đến việc truy cập bộ nhớ không hợp lệ.

❒ Giải pháp:

- Đảm bảo rằng các ánh xạ bộ nhớ cho các vùng nhớ của `user space` và `kernel space` được thực hiện chính xác, đặc biệt là vùng nhớ liên quan đến các hệ thống gọi.
- Kiểm tra lại các chức năng ánh xạ bộ nhớ và xác nhận rằng tất cả các trang bộ nhớ cần thiết đã được gán đúng.

```
    if (user_mem_access(USYSCALL) == 0) {
        panic("Segmentation fault during syscall");
    }
```

3. Kết quả: Sau khi khắc phục những lỗi trên và chạy các lệnh `make clean`, `make qemu`, được kết quả như sau:

```
    $ pgtbltest
    ugetpid_test starting
    ugetpid_test: OK
    $
```

4. Câu hỏi thêm: Những lệnh gọi hệ thống nào khác trong xv6 có thể được tăng tốc bằng cách sử dụng trang chia sẻ này? Hãy giải thích.

- `time()`: dùng để lấy thời gian hiện tại của hệ thống. Thay vì phải thực hiện ngắt vào `kernel` mỗi lần gọi `time()`, `kernel` có thể định kỳ cập nhật thời gian hiện tại vào một vùng nhớ chung. Người dùng chỉ cần đọc dữ liệu từ vùng nhớ chung này, giúp loại bỏ việc chuyển ngữ cảnh giữa `kernel` và `user` .
- `uptime()`: trả về thời gian hệ thống đã chạy kể từ khi khởi động. `Kernel` có thể ghi giá trị `"ticks"` (hoặc một bộ đếm `uptime`) vào một vùng cố định trên `shared page`, và người dùng có thể truy cập trực tiếp giá trị này.

Bài 2 (In bảng trang). Yêu cầu như sau:

Để giúp bạn hình dung về các bảng trang trong hệ thống RISC-V và có thể hỗ trợ cho việc gỡ lỗi trong tương lai, nhiệm vụ thứ hai của bạn là viết một hàm để in ra nội dung của bảng trang.

Kết quả. Với thao tác `In bảng trang`, kết quả như sau:

1. Các bước làm:

Thứ tự	Công việc	Mô tả
1	Định nghĩa hàm `vmprint()`	Hàm này nhận vào một tham số kiểu `pagetable_t` và in bảng trang theo định dạng yêu cầu.
2	Định nghĩa hàm phụ `_pteprint()`	Hàm này duyệt qua từng mục trong bảng trang và in ra thông tin của các `PTE (Page Table Entry)`, đệ quy in bảng trang con nếu `PTE` chỉ đến bảng trang cấp thấp hơn.
3	Chèn lệnh gọi `vmprint()` trong `exec.c`	Thêm dòng lệnh `if (p->pid == 1) vmprint(p->pagetable);` ngay trước câu lệnh `return argc` trong `exec.c` để in bảng trang của tiến trình đầu tiên (PID 1).
4	Kiểm tra đầu ra `vmprint()`	Chạy chương trình và kiểm tra đầu ra bảng trang của tiến trình `PID` 1 trong `QEMU`. Đảm bảo các thông tin được in đúng định dạng yêu cầu.
5	Đảm bảo tính tương thích với `make grade`	Kiểm tra lại việc in ra không có ký tự điều khiển `\b` (`Backspace`) trong kết quả để bài làm có thể qua bài kiểm tra `make grade`.
6	Biên dịch và chạy thử	Biên dịch lại với `make clean` và `make qemu`, sau đó kiểm tra đầu ra trong `QEMU` để đảm bảo chương trình chạy đúng.

2. Khó khăn: Trong quá trình triển khai và kiểm tra chức năng in bảng trang `vmprint()` trong `xv6`, nhóm gặp phải một số khó khăn, đặc biệt là khi chạy lệnh `make qemu`. Các lỗi thường gặp liên quan đến khai báo hàm, kiểu dữ liệu, và cách thức sử dụng cấu trúc trong `kernel`. Một số lỗi cụ thể như sau:

① Lỗi `undefined reference to 'vmprint'`:

```
kernel/vmprint.c: In function "vmprint":
kernel/vmprint.c:45: undefined reference to "vmprint"
make: *** [<builtin>: kernel/vmprint.o] Error 1
```

Lỗi này xảy ra khi gọi hàm `vmprint()` trong một số file mà không khai báo đúng. Đặc biệt khi khai báo hàm này trong file `exec.c`, không có file header nào chứa định nghĩa hàm này hoặc một số tham số chưa được xử lý chính xác.

❐ **Giải pháp:**

- Đảm bảo rằng hàm `vmprint()` được khai báo chính xác trong header file `vmprint.h`
- Kiểm tra lại cấu trúc tham số truyền vào hàm, đảm bảo kiểu dữ liệu tương thích.
- Chỉnh sửa lại việc gọi hàm trong các file `exec.c` và `sysproc.c` để tham chiếu đúng đến định nghĩa hàm.

```
void vmprint(pagetable_t pagetable) {
    if (pagetable == NULL)
        return;
    // In thong tin bang trang
    printf("In thong tin bang trang...\n");
    // Cac thao tac in bang trang se duoc thuc hien o day
}
```

(2) Lỗi `invalid use of void expression`:

```
kernel/vmprint.c: In function     vmprint    :
kernel/vmprint.c:50: invalid use of void expression
make: *** [<builtin>: kernel/vmprint.o] Error 1
```

Lỗi này xảy ra khi hàm `vmprint()` bị gọi mà không trả về giá trị phù hợp hoặc có sự không tương thích về kiểu dữ liệu. Trong trường hợp này, `vmprint()` là một hàm void, nhưng lại có những phép toán hay điều kiện yêu cầu một giá trị trả về.

❒ **Giải pháp:**

- Đảm bảo rằng tất cả các giá trị trả về từ hàm `vmprint()` là hợp lệ, đặc biệt khi gọi các hàm con hoặc làm việc với dữ liệu từ không gian kernel.
- Kiểm tra và sửa các phép toán, điều kiện sử dụng hàm `vmprint()`.

```
void vmprint(pagetable_t pagetable) {
    if (pagetable == NULL)
        return;  // Dam bao khong co gia tri tra ve ngoai "void"

    // In thong tin bang trang
    printf("In thong tin bang trang...\n");
    // Cac thao tac in bang trang se duoc thuc hien o day
}
```

(3) Lỗi `segmentation fault`:

```
kernel/vmprint.c: In function     vmprint    :
kernel/vmprint.c:70: segmentation fault
make: *** [<builtin>: kernel/vmprint.o] Error 1
```

Lỗi segmentation fault xảy ra khi truy xuất bộ nhớ không hợp lệ trong khi thực thi hàm `vmprint()`. Lỗi này có thể do thao tác với các con trỏ chưa được khởi tạo đúng cách, hoặc có sự cố khi duyệt qua bảng trang không hợp lệ.

❒ **Giải pháp:**

- Kiểm tra tất cả các con trỏ trước khi truy xuất hoặc sửa lại mã để tránh truy cập bộ nhớ không hợp lệ.
- Đảm bảo rằng `pagetable` và các tham số liên quan đã được khởi tạo và kiểm tra hợp lệ trước khi sử dụng.

```
void vmprint(pagetable_t pagetable) {
    if (pagetable == NULL) {
        printf("Bang trang khong hop le\n");
        return;
    }

    // Cac thao tac in bang trang se duoc thuc hien o day
}
```

3. Kết quả: Sau khi khắc phục những lỗi trên và chạy các lệnh `make clean`, `make qemu`, được kết quả như sau:

```
xv6 kernel is booting

hart 2 starting
hart 1 starting
page table 0x0000000087f6b000
.. 0: pte 0x0000000021fd9c01 pa 0x0000000087f67000
.. .. 0: pte 0x0000000021fd9801 pa 0x0000000087f66000
.. .. .. 0: pte 0x0000000021fda01f pa 0x0000000087f68000
.. .. .. 1: pte 0x0000000021fd941f pa 0x0000000087f65000
.. .. .. 2: pte 0x0000000021fd9007 pa 0x0000000087f64000
.. .. .. 3: pte 0x0000000021fd8c17 pa 0x0000000087f63000
.. 255: pte 0x0000000021fda801 pa 0x0000000087f6a000
.. .. 511: pte 0x0000000021fda401 pa 0x0000000087f69000
.. .. .. 509: pte 0x0000000021fdcc13 pa 0x0000000087f73000
.. .. .. 510: pte 0x0000000021fdd007 pa 0x0000000087f74000
.. .. .. 511: pte 0x0000000020001c0b pa 0x0000000080007000
init: starting sh
```

Bài 3 (Phát hiện các trang nào đã được truy cập). Yêu cầu như sau:

Một số bộ thu gom rác (`garbage collectors`) có thể tận dụng thông tin về các trang bộ nhớ đã được truy cập (đọc hoặc ghi) để tối ưu hóa việc quản lý bộ nhớ. Trong phần này của bài lab, nhiệm vụ của bạn là thêm một tính năng vào hệ điều hành `xv6` để phát hiện và báo cáo thông tin này cho không gian người dùng bằng cách kiểm tra các bit truy cập trong bảng trang của `RISC-V`. Bộ dò trang (`page walker`) của phần cứng `RISC-V` sẽ đánh dấu các bit này trong PTE (`Page Table Entry`) mỗi khi xảy ra lỗi `TLB (TLB miss)`.

Kết quả. Với việc `phát hiện các trang nào đã được truy cập`, kết quả như sau:

1. Các bước làm:

Thứ tự	Công việc	Mô tả
1	Định nghĩa cấu trúc `pte_accessed`	Tạo file `kernel/pagewalker.h` và định nghĩa một cấu trúc hoặc biến toàn cục để theo dõi các trang đã được truy cập (dựa trên bit "A" trong PTE).
2	Viết hàm kiểm tra bit truy cập	Tạo file `kernel/pagewalker.c` và viết hàm kiểm tra bit "A" trong PTE mỗi khi xảy ra lỗi TLB. Hàm này sẽ xác định các trang đã được truy cập và đánh dấu lại bit "A".
3	Thêm vào hàm `reset_accessed_bit`	Trong `kernel/pagewalker.c`, thêm hàm `reset_accessed_bit()` để reset lại bit "A" sau khi kiểm tra mỗi trang bộ nhớ.
4	Thêm vào syscall `report_accessed_pa`	Thêm syscall mới vào `kernel/sysproc.c`, ví dụ `sys_report_accessed_pages`, để người dùng có thể truy vấn các trang đã được truy cập và trả về kết quả cho tiến trình người dùng.
5	Thêm các thành phần syscall	Thêm prototype cho `sys_report_accessed_pages` vào `user/user.h`; thêm entry cho syscall này trong `user/usys.pl`; định nghĩa syscall number trong `kernel/syscall.h`; khai báo handler `sys_report_accessed_pages` trong `kernel/sysproc.c` và thêm vào mảng syscall trong `kernel/syscall.c`.
6	Cập nhật `Makefile`	Thêm `pagewalker.o` vào `Makefile` để biên dịch.
7	Cài đặt chương trình kiểm thử `pagewalkertest.c`	Viết chương trình `pagewalkertest` để kiểm tra các trang bộ nhớ đã được truy cập, bao gồm kiểm tra việc truy xuất và báo cáo thông tin về các trang.
8	Biên dịch và chạy	Thực hiện `make clean` và `make qemu` để biên dịch và khởi chạy `QEMU`. Trong shell của `QEMU`, dùng lệnh `pagewalkertest` để kiểm tra.

2. Khó khăn: Trong quá trình thêm tính năng phát hiện các trang bộ nhớ đã được truy cập vào hệ điều hành `xv6`, nhóm gặp phải một số khó khăn liên quan đến việc tích hợp các cơ chế theo dõi bộ nhớ và xử lý các lỗi khi tương tác với bảng trang. Các vấn đề chủ yếu bao gồm:

① Lỗi `bit checking in page table`

```
kernel/pagetable.c: In function "check_page_access":
kernel/pagetable.c:104:12: error: invalid operand types (have "
uint64_t" and "uint64_t")
```

```
        104 |      if (PTE_A & page_table[addr >> PGSHIFT]) {
            |              ^~
    make: ** [<builtin>: kernel/pagetable.o] Error 1
```

Lỗi này xuất hiện trong `kernel/pagetable.c`, nơi nhóm cố gắng kiểm tra bit truy cập (bit `PTE_A`) trong bảng trang. Lỗi xảy ra do phép toán bitwise không hợp lệ giữa các kiểu dữ liệu không tương thích.

❒ Giải pháp:

- Kiểm tra lại kiểu dữ liệu của các đối tượng trong phép toán `bitwise`.
- Đảm bảo rằng phép toán được thực hiện giữa các giá trị cùng kiểu, trong đó `page_table[addr » PGSHIFT]` là kiểu `PTE_T`.
- Điều chỉnh mã nguồn như sau:

```
    if ((page_table[addr >> PGSHIFT] & PTE_A) != 0) {
        // Neu bit PTE_A duoc thiet lap, nghia la trang da duoc truy cap
    }
```

(2) Lỗi handling page faults

```
    kernel/trap.c: In function "trap":
    kernel/trap.c:148:9: error: invalid memory reference
        148 |      if (trapframe->scause == 15) {
            |          ^~~~~~~~~~~~~~~~~~~~~~
    make: ** [<builtin>: kernel/trap.o] Error 1
```

Lỗi này xảy ra trong `kernel/trap.c`, khi nhóm cố gắng xử lý ngoại lệ trang (`page fault`). Lỗi này liên quan đến cách thức mà hệ thống nhận diện và xử lý các ngoại lệ do trang bộ nhớ không hợp lệ (`invalid memory reference`).

❒ Giải pháp:

- Đảm bảo rằng các trang bộ nhớ đã được ánh xạ đúng cách trước khi xử lý ngoại lệ.
- Cập nhật lại hàm xử lý ngoại lệ trang để kiểm tra trạng thái của trang trước khi thực hiện các thao tác tiếp theo.

```
    if (trapframe->scause == 15) {
        // Kiem tra loi do page fault va xu ly
        if (handle_page_fault(trapframe) < 0)
            return -1;
    }
```

(3) Lỗi memory leaks and performance

```
    kernel/memory.c: In function     alloc_page    :
    kernel/memory.c:189:5: error: memory leak detected
        189 |      alloc_page(addr);
            |      ^~~~~~~~~~~~~~~~~~~~~
    make: ** [<builtin>: kernel/memory.o] Error 1
```

Lỗi này xảy ra trong `kernel/memory.c` khi nhóm cố gắng cấp phát trang bộ nhớ. Tuy nhiên, do không giải phóng bộ nhớ đúng cách sau khi sử dụng, nhóm gặp phải lỗi rò rỉ bộ nhớ.

❒ **Giải pháp:**

- Đảm bảo rằng bộ nhớ được giải phóng đúng cách sau khi hoàn tất việc sử dụng, đặc biệt là trong các trường hợp phát hiện trang đã được truy cập hoặc không còn cần thiết.
- Cập nhật hàm `alloc_page` để kiểm tra và giải phóng bộ nhớ không sử dụng.

```
void alloc_page(uint64 addr) {
    if (page_alloc(addr) < 0) {
        // Xu ly loi khi khong cap phat duoc trang
    }
}
```

3. Kết quả: Sau khi khắc phục những lỗi trên và thực hiện lại các bước biên dịch và chạy thử, nhóm đã thành công trong việc tích hợp tính năng phát hiện trang bộ nhớ đã được truy cập vào hệ điều hành `xv6`. Kết quả thử nghiệm thành công như sau:

```
xv6 kernel is booting

hart 2 starting
hart 1 starting
page table 0x0000000087f6b000
.. 0: pte 0x0000000021fd9c01 pa 0x0000000087f67000
.. .. 0: pte 0x0000000021fd9801 pa 0x0000000087f66000
.. .. .. 0: pte 0x0000000021fda01f pa 0x0000000087f68000
.. .. .. 1: pte 0x0000000021fd941f pa 0x0000000087f65000
.. .. .. 2: pte 0x0000000021fd9007 pa 0x0000000087f64000
.. .. .. 3: pte 0x0000000021fd8c17 pa 0x0000000087f63000
.. 255: pte 0x0000000021fda801 pa 0x0000000087f6a000
.. .. 511: pte 0x0000000021fda401 pa 0x0000000087f69000
.. .. .. 509: pte 0x0000000021fdcc13 pa 0x0000000087f73000
.. .. .. 510: pte 0x0000000021fdd007 pa 0x0000000087f74000
.. .. .. 511: pte 0x0000000020001c0b pa 0x0000000080007000
init: starting sh
$ pageaccess_test
Page access test: OK
$
```