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

BÁO CÁO ĐỒ ÁN SỐ 3:

Page Tables

Môn học : Hệ điều hành
Giảng viên lý thuyết : Trần Trung Dũng
Giảng viên hướng dẫn thực hành : Lê Giang Thanh
Nguyễn Thanh Quân

TP Hồ Chí Minh 12/2023

MỤC LỤC

Chapter 1

Bảng đánh giá thành viên

MSSV	Họ và tên	% đóng góp
21120075	Trần Minh Hoàng	100%
21120471	Phan Gia Huy	100%

Chapter 2

Tổng quan đồ án

Đồ án thứ ba của môn học hệ điều hành phần thực hành là khám phá về bảng trang (page tables) và thực hiện các sửa đổi để tăng tốc một số system call cụ thể và để phát hiện xem trang nào đã được truy cập.

Page tables (Bảng trang) được sử dụng trong hệ thống máy tính để ánh xạ từ địa chỉ ảo của một tiến trình sang địa chỉ vật lý trong bộ nhớ. Điều này giúp quản lý bộ nhớ trở nên hiệu quả hơn. Trong đồ án, chúng em sẽ thực hiện 3 công việc: tăng tốc system call $getpid()$ trong xv6, viết hàm để in nội dung của một bảng trang và cuối cùng là in ra các trang đã được truy cập

Chapter 3

Cài đặt các chương trình

3.1 Speed up system calls

- **Mục đích**:
  Ta có thể tối ưu hóa việc gọi hệ thống trong hệ điều hành xv6 bằng cách chia sẻ dữ liệu trong vùng chỉ đọc giữa không gian người dùng và kernel.

- **Quá trình cài đặt**:

  – **Thêm usyscall**:Trong cấu trúc struct proc trong file *proc.h*,thêm một trường *usyscall* để kết nối hoặc theo dõi thông tin về các syscall từ user space tới kernel space.

```
struct proc {
    // ... Cac truong khac
     struct usyscall* usyscall; //process struct usyscall
};
```

  – **Xử lý proc_pagetable trong kernel/proc.c:**Mục đích chính của việc thêm này chính là thiết lập bảng trang cho một tiến trình trong hệ thống. Nó gán giá trị của **p->pid** vào một trường trong cấu trúc **usyscall** của tiến trình và ánh xạ cấu trúc **usyscall** đó vào không gian bộ nhớ của tiến trình thông qua bảng trang. Nếu quá trình ánh xạ không thành công, nó sẽ xử lý lỗi bằng cách gỡ bỏ ánh xạ và giải phóng bộ nhớ được cấp phát trước đó để tránh rò rỉ bộ nhớ.

```
//... Cai dat truoc
p->usyscall->pid = p->pid;
if(mappages(pagetable, USYSCALL, PGSIZE,(uint64)p->usyscall,
    PTE_R | PTE_U) < 0){
    uvmunmap(pagetable, USYSCALL, 1, 0);
    uvmfree(pagetable, 0);
}
```

  – **Xử lý allocproc trong kernel/proc.c:**Mục đích của việc thêm này chính là thực hiện việc khởi tạo một tiến trình mới trong hệ thống. Nó cấp phát bộ nhớ cho cấu trúc

usyscall của tiến trình, chứa thông tin về các cuộc gọi hệ thống. Nếu việc cấp phát bộ nhớ không thành công, nó giải phóng tài nguyên và trả về 0, báo hiệu rằng quá trình tạo tiến trình mới gặp lỗi vì không đủ tài nguyên. Điều này đảm bảo quá trình tạo tiến trình diễn ra một cách an toàn và tránh rủi ro của việc thiếu tài nguyên khi tạo các tiến trình mới.

```
// ... Cai dat truoc
if((p->usyscall = (struct usyscall *)kalloc()) == 0){
    freeproc(p);
    release(&p->lock);
    return 0;
}
```

- **Xử lý freeproc trong kernel/proc.c:**Mục đích của viejc thêm này chính là giải phóng bộ nhớ được cấp phát cho **p->usyscall**, đồng thời đặt con trỏ này về null để tránh lỗi trỏ đến vùng nhớ không hợp lệ.

```
//... Cai dat truoc
if(p->usyscall)
    kfree((void*)p->usyscall);
p->usyscall = 0;
```

- **Xử lý proc_freepagetable trong kernel/proc.c:**Mục đích là hủy ánh xạ trang cho cấu trúc usyscall của tiến trình, giải phóng vùng nhớ tương ứng từ bảng trang.

```
// ... Cai dat truoc
uvmunmap(pagetable, USYSCALL, 1, 0);
```

3.2 Print a page table

- **Mục đích:**
  Việc in ra nội dung 1 bảng trang hỗ trợ trong việc debug và hiểu rõ cách quản lý bộ nhớ, bao gồm cả việc ánh xạ địa chỉ ảo sang địa chỉ vật lý, đồng thời giúp sinh viên làm quen với cấu trúc và cách thức hoạt động của bảng trang trong hệ thống xv6.

- **Quá trình cài đặt:**

  - **Xử lý exec trong kernel/exec.c:** Thêm đoạn mã kiểm tra xem tiến trình hiện tại có phải là tiến trình đầu tiên (init process) hay không.Nếu có sẽ gọi hàm **vmprint** để in

ra cấu trúc của bảng trang của tiến trình init.Mục đích chính là hiển thị cấu trúc bộ nhớ ảo của tiến trình đầu tiên, giúp trong việc debug, đánh giá và hiểu rõ hơn về cách thức quản lý bộ nhớ.

```
            // ... Cai dat truoc
            if(p->pid == 1){
                vmprint(p->pagetable);
            }
```

– **Thêm hàm vmprint trong kernel/vm.c:**

- Hàm này được thiết kế để hiển thị cấu trúc của bảng trang, tập hợp các thông tin quan trọng về quản lý bộ nhớ ảo. Quá trình này bao gồm việc duyệt qua các mục trong bảng trang, kiểm tra tính hợp lệ của chúng và in ra thông tin chi tiết về các PTE (Page Table Entry) cùng với địa chỉ vật lý tương ứng. Nếu các mục hợp lệ được tìm thấy, hàm sẽ tiếp tục xem xét các cấp độ của bảng trang và in ra thông tin tương ứng.
- Mục tiêu chính của hàm này là cung cấp cái nhìn tổng quan về cách mà hệ thống quản lý bộ nhớ ảo, bao gồm cả cấu trúc của bảng trang và cách các PTE ánh xạ địa chỉ ảo sang địa chỉ vật lý. Hàm vmprint() hữu ích trong việc debug, giúp người lập trình hiểu rõ hơn về cách thức hoạt động của hệ thống và cách quản lý bộ nhớ.

```
void vmprint(pagetable_t pagetable){
  printf("page table %p\n", pagetable);
  for(int i = 0; i < 512; i++){
    pte_t pte = pagetable[i];
    if(pte & PTE_V){
      printf("..%d: pte %p pa %p\n", i, pte, PTE2PA(pte));
      pagetable_t second = (pagetable_t)PTE2PA(pte);
      for(int j = 0; j < 512; j++){
        pte = second[j];
        if(pte & PTE_V){
          printf(".. ..%d: pte %p pa %p\n", j, pte, PTE2PA(pte));
          pagetable_t third = (pagetable_t)PTE2PA(pte);
          for(int k = 0; k < 512; k++){
            pte = third[k];
            if(pte & PTE_V){
              printf(".. .. ..%d: pte %p pa %p\n", k, pte, PTE2PA(pte));
            }
          }
        }
      }
```

```
    }
  }
}
```

– **Define hàm vmprint trong kernel/defs.h:**Định nghĩa hàm vmprint trong file defs.h để đại diện cho hàm

```
// File defs.h
void vmprint(pagetable_t pagetable);
```

3.3 Detect which pages have been accessed

- **Mục đích**:
  Việc xác định và báo cáo những trang bộ nhớ (pages) đã được truy cập (đọc hoặc ghi) trong quá trình thực thi chương trình rất hữu ích cho việc quản lý bộ nhớ, tối ưu hóa thao tác I/O, kiểm tra an ninh, và các mục đích khác nhau trong hệ thống máy tính.

- **Quá trình cài đặt**:

  – **Define access bit** Trong kernel/riscv.h, ta sẽ định nghĩa 1 macro PTE_A có giá trị 2^6 (64). Bit thứ 6 trong một PTE được đặt để chỉ ra rằng trang đã được truy cập.

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

  – **Cài đặt system call pgaccess()**: Hàm này được thiết kế trong kernel/sysproc.c để báo cáo những trang nào đã được truy cập bằng cách kiểm tra các bit truy cập trong bảng trang RISC-V.

    * Hàm sẽ nhận địa chỉ bắt đầu $startaddr$, số lượng trang $npage$, và địa chỉ buffer $useraddr$ từ user space thông qua system call arguments. Biến $bitmask$ lưu trữ trạng thái trang đã được truy cập, $complement$ chứa bổ sung của bit PTE_A.
    * Tiếp theo đó, ta sử dụng vòng lặp để kiểm tra từng trang. Kiểm tra xem bit PTE_A có được set trong PTE không. Nếu có, đặt bit tương ứng trong bitmask và clear bit PTE_A trong PTE.
    * Sau cùng, ta copy giá trị của bitmask từ kernel space vào user space thông qua hàm copyout.

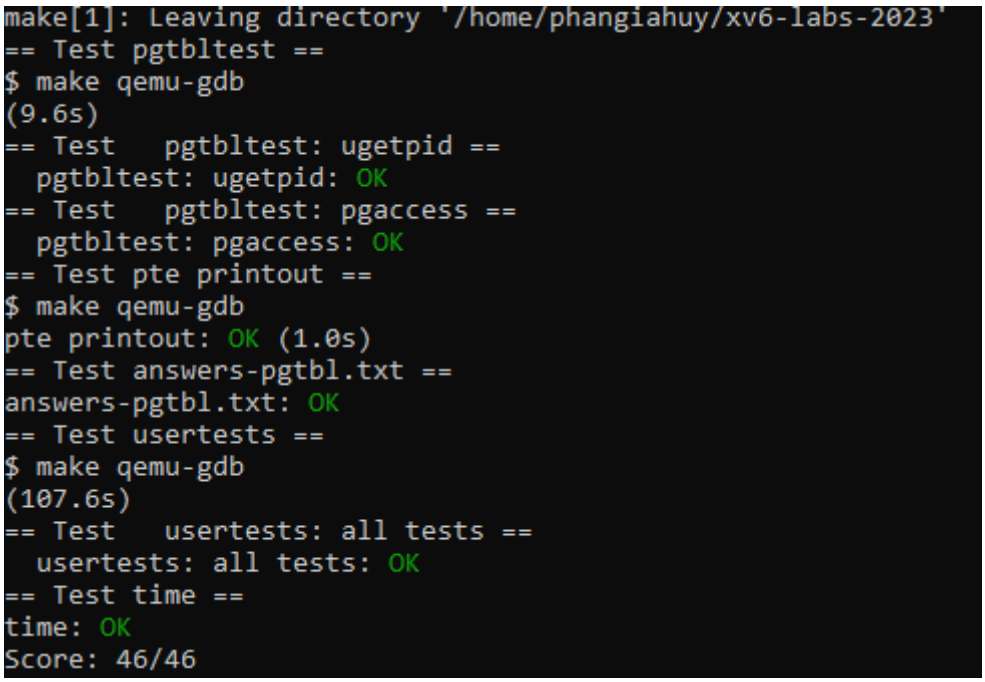
```
int
```

```
sys_pgaccess(void)
{
  uint64 vaddr;
  int num;
  uint64 res_addr;
  argaddr(0, &vaddr);
  argint(1, &num);
  argaddr(2, &res_addr);

  struct proc *p = myproc();
  pagetable_t pagetable = p->pagetable;
  uint64 res = 0;

  for(int i = 0; i < num; i++){
    pte_t* pte = walk(pagetable, vaddr + PGSIZE * i, 0);
    if(*pte & PTE_A){
      *pte &= (~PTE_A);
      res |= (1L << i);
    }
  }

  copyout(pagetable, res_addr, (char*)&res, sizeof(uint64));
  return 0;
}
```

Chapter 4

Tổng kết đánh giá

Nhóm đã hoàn thành cài đặt các chương trình. Dưới đây là bảng kết quả sau khi dùng lệnh **make grade** :



Tham khảo

[1] Thông tin về pages table :
https://pdos.csail.mit.edu/6.S081/2020/xv6/book-riscv-rev1.pdf