Lab 5. xv6 page table

1. Mục đích

Thông qua bài thực hành này, sinh viên sẽ hiểu được cách hoạt động và cấu hình page table trong xv6 trên kiến trúc RISC-V

2. Tóm tắt lý thuyết

xv6 chạy trên kiến trúc Sv39 RISC-V, trong đó chỉ 39bit cuối của 64bit virtual address được sử dụng. Về mặt logic, **page table** của RISC-V là một mảng chứa 2^{27} **page table entry (PTE).** Mỗi PTE gồm 44bit **physical page number (PPN)** và 10bit **flags**.

Quá trình chuyển đổi từ **virtual address (VA)** sang **physical adresss (PA)** diễn ra như sau: 27bit cao của 39bit virtual address được dùng để xác định PTE tương ứng thông qua three-level (L2, L1, L0) page table (page directory). 44bit PPN tìm được trong PTE kết hợp với 12bit thấp của 39bit virtual address tạo thành 56bit physical adresss

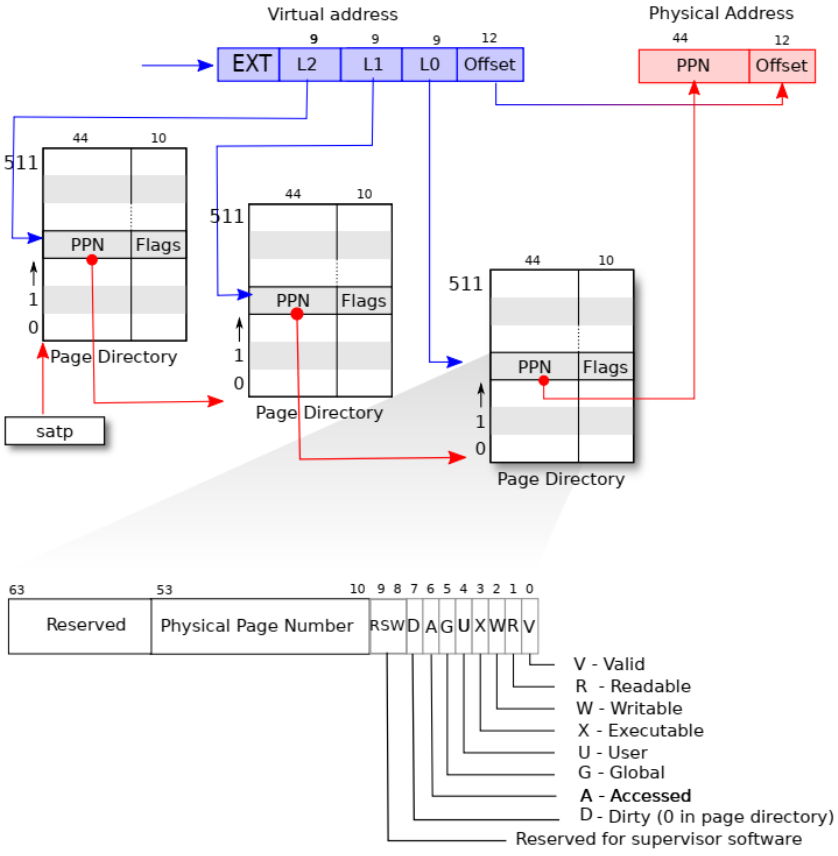


Figure 3.2: RISC-V address translation details.

Khi chuyển đổi từ VA sang PA, các flags bit của PTE được kiểm tra để xác định tính khả dụng của PTE:

- Bit V xác định tính hiện hữu của PTE
- Bit U cho biết PTE có thể truy cập trong user space
- Bảng bên dưới là các trường hợp sử dụng được quy định bởi bit RWX

X	W	R	Meaning
0	0	0	Pointer to next level of page table.
0	0	1	Read-only page.
0	1	0	*Reserved for future use.*
0	1	1	Read-write page.
1	0	0	Execute-only page.
1	0	1	Read-execute page.
1	1	0	*Reserved for future use.*
1	1	1	Read-write-execute page.

3. Thực hành

Bài thực hành này sẽ in ra Terminal nội dung các page table (page directory) của process init có pid bằng 1, là process đầu tiên được khởi tạo bởi xv6. Từ đó quan sát và phân tích virtual address space của process init

Chuyển thư mục làm việc vào thư mục source code của xv6 và tạo branch mới lab5 từ branch origin/pgtbl, sau đó checkout branch lab5

Thêm đoạn code bên dưới vào cuối file kernel/vm.c. Đoạn code này định nghĩa function vmprint() nhận tham số là một root pagetable (level-2 pagetable) và in ra pagetable đó

```
1  void vmprintrec(pagetable_t pagetable, int level){
2    if(level == 2)  printf("page table %p\n", pagetable);
3    for(int i = 0; i < 512; i++){
4      pte_t pte = pagetable[i];
5      if(pte & PTE_V){
6        for(int i = 2-level; i > 0; i--){
7          printf(" ..");
8        }
9        printf(" ..%d: pte %p pa %p\n", i, pte, PTE2PA(pte));
10       if((pte & (PTE_R|PTE_W|PTE_X)) == 0){
11         // this PTE points to a lower-level page table.
12         uint64 child = PTE2PA(pte);
13         vmprintrec((pagetable_t)child, level-1);
14       }
15     }
16   }
17 }
18
19 void vmprint(pagetable_t pagetable){
20   vmprintrec(pagetable,  2);
21 }
```

Thêm prototype của function vmprint() vào file kernel/defs.h để có thể gọi nó từ các file source code khác

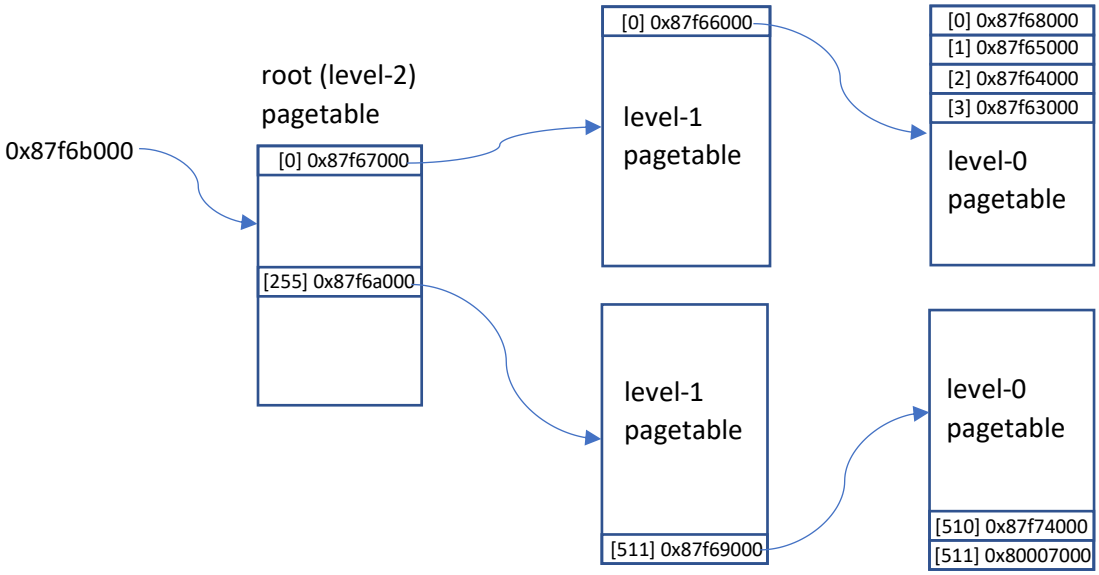
Thêm đoạn code bên dưới vào function exec() trong file kernel/exec.c, ngay trước dòng *return argc*

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

return argc; // this ends up in a0, the first argument to main(argc, argv)
```

Save các file đã thay đổi. Thực thi command make qemu để khởi động xv6. Pagetable của process init được in ra Terminal như bên dưới

```
1  page table 0x0000000087f6b000
2   ..0: pte 0x0000000021fd9c01 pa 0x0000000087f67000
3   .. ..0: pte 0x0000000021fd9801 pa 0x0000000087f66000
4   .. .. ..0: pte 0x0000000021fda01b pa 0x0000000087f68000
5   .. .. ..1: pte 0x0000000021fd9417 pa 0x0000000087f65000
6   .. .. ..2: pte 0x0000000021fd9007 pa 0x0000000087f64000
7   .. .. ..3: pte 0x0000000021fd8c17 pa 0x0000000087f63000
8   ..255: pte 0x0000000021fda801 pa 0x0000000087f6a000
9   .. ..511: pte 0x0000000021fda401 pa 0x0000000087f69000
10  .. .. ..510: pte 0x0000000021fdd007 pa 0x0000000087f74000
11  .. .. ..511: pte 0x0000000020001c0b pa 0x0000000080007000
12 init: starting sh
```



Dòng 1 là tham số truyền vào function printvm(), là PA của level-2 pagetable của process init. Các dòng bên dưới là các PTE. Số lượng dấu ".." ở đầu dòng cho biết level của PTE, theo sau là số index (vị trí của PTE trong pagetable) và giá trị của PTE. Cuối cùng là PA trích xuất từ PTE.

Dòng 2 và 8 là hai PTE của level-2 pagetable, mỗi PTE tương ứng với một PA của level-1 pagetable. Dòng 3 và 9 là hai PTE của hai level-1 pagetable, mỗi PTE tương ứng với một PA của level-0 pagetable. Dòng 4, 5, 6, 7, 10, 11 là các PTE của hai level-0 pagetable, mỗi PTE tương ứng với một PA của một page

BÁO CÁO THỰC HÀNH

Sinh viên: .....

MSSV: ..... Nhóm: .....

Bài 1: Trả lời các câu hỏi sau

- Có bao nhiêu pages được mapped vào virtual address space của process init? Các page này chứa nội dung gì?
- Process init có thể read và write vào page nào trong số các pages được mapped?
- Process init có thể execute code ở page nào trong số các pages được mapped?

Bài 2: Viết user program **myspace** trên xv6 tăng kích thước address space của nó lên 1 byte bằng cách gọi đến system call sbrk(1). Chạy program myspace và so sánh pagetable của nó trước và sau khi gọi đến sbrk(). Kernel đã cấp phát cho program myspace thêm bao nhiêu bộ nhớ?

Gợi ý: Dùng function printvm() để xem pagetable của process. System call sbrk() được thực hiện trong function sys_sbrk() trong file kernel/sysproc.c

Bài 3*: Viết function vmprintmap() nhận tham số là một root pagetable và in ra Terminal mapping của các pages và flags của chúng như ví dụ bên dưới:

```
1 va:0x0000000000000000 pa:0x0000000087f68000 (VR-XU)
2 va:0x0000000000001000 pa:0x0000000087f65000 (VRW-U)
3 va:0x0000000000002000 pa:0x0000000087f64000 (VRW--)
4 va:0x0000000000003000 pa:0x0000000087f63000 (VRW-U)
5 va:0x0000003fffffd000 pa:0x0000000087f73000 (VR--U)
6 va:0x0000003fffffe000 pa:0x0000000087f74000 (VRW--)
7 va:0x0000003ffffff000 pa:0x0000000080007000 (VR-X-)
```

Có 7 pages được mapped trong ví dụ này. Page ở dòng 5 được mapped từ VA 0x0000003fffffd000 sang PA 0x0000000087f73000. Process chỉ có quyền read nhưng không có quyền write hay execute với page này

Gợi ý:

- Xem function vmprint() để lấy ý tưởng
- VA được tạo thành qua từng level của pagetable còn PA được trích xuất từ PTE của level-0 pagetable
- Function vmprintmap() nên được định nghĩa trong file kernel/vm.c với prototype khai báo trong file kernel/defs.h
- Kiểm tra function vmprintmap() với pagetable của process init, sau đó đối chiếu kết quả của nó với vmprint()

Bài 4*: Viết các function sau:

- vmprintva2pa() nhận tham số là một root pagetable và một VA, và in ra Terminal PA được mapped với VA đó trong pagetable. In ra thông báo lỗi nếu mapping không tồn tại

- vmprintpa2va() nhận tham số là một root pagetable và một PA, và in ra Terminal **các** VA được mapped với PA đó trong pagetable. In ra thông báo lỗi nếu mapping không tồn tại

Để kiểm tra chức năng của hai function trên, thêm đoạn code bên dưới vào function kvminit() trong file kernel/vm.c

```
void
kvminit(void)
{
  kernel_pagetable = kvmmake();

  vmprintva2pa(kernel_pagetable, 0);
  vmprintva2pa(kernel_pagetable, UART0);
  vmprintva2pa(kernel_pagetable, UART0 + 100);
  vmprintva2pa(kernel_pagetable, TRAMPOLINE + 1);

  vmprintpa2va(kernel_pagetable, 0);
  vmprintpa2va(kernel_pagetable, UART0);
  vmprintpa2va(kernel_pagetable, UART0 + 100);
  vmprintpa2va(kernel_pagetable, (uint64)trampoline + 1);
}
```

Khi xv6 chạy, kết quả bên dưới được in ra Terminal:

```
Error: there is no mapping for va 0x0000000000000000
va 0x0000000010000000 mapped to pa 0x0000000010000000
va 0x0000000010000064 mapped to pa 0x0000000010000064
va 0x0000003ffffff001 mapped to pa 0x0000000080007001
Error: there is no mapping for pa 0x0000000000000000
pa 0x0000000010000000 mapped to following va:
    0x0000000010000000
pa 0x0000000010000064 mapped to following va:
    0x0000000010000064
pa 0x0000000080007001 mapped to following va:
    0x0000000080007001
    0x0000003ffffff001
```

Gợi ý:

- Tham khảo các function walk(), walkaddr(), vmprint() trong kernel/vm.c
- VA được tạo thành qua từng level của pagetable còn PA được trích xuất từ PTE của level-0 pagetable
- Function vmprintva2pa() và vmprintpa2va() nên được định nghĩa trong file kernel/vm.c với prototype khai báo trong file kernel/defs.h

Bài 5: Thực hiện mapping một page chứa pid của process vào process's virtual address space. Process có thể đọc pid của mình từ page này thay vì gọi đến system call getpid(), nhờ đó tăng tốc độ xử lý. Nhiều hệ điều hành hiện nay cũng dùng phương pháp tương tự để tăng tốc độ xử lý cho các system call

Gợi ý:

- Tham khảo cách xv6 thực hiện mapping page TRAPFRAME cho mỗi process
- Khi một process được tạo ra, map một page read-only tại virtual address USYSCALL. Sau đó lưu trữ struct usyscall vào đầu page và khởi tạo struct bằng giá trị pid của process
    - USYSCALL và struct usyscall được định nghĩa trong file kernel/memlayout.h

    - Cấp phát và khởi tạo page trong function allocproc() (file kernel/proc.c)
    - Địa chỉ của page đã cấp phát có thể được lưu trong struct proc (file kernel/proc.h)
    - Mapping page trong function proc_pagetable() dùng function mappages() (file kernel/proc.c)
    - Giải phóng page đã cấp phát trong function freeproc() (file kernel/proc.c)
- Function ugetpid() (fiel user/ulib.c) đọc pid của process từ virtual address space. User program có thể dùng function này thay vì system call getpid()