EEE3535-01 **Fall 2020**

Assignment 5: Paging

Due: Sunday, Dec. 13, 2020, 11:59PM

1 Introduction

- The goal of this assignment is to understand the paging scheme and address translation using xv6-riscv.
- In this assignment, you are asked to implement three system calls that expose the page table information of a process to a user program. The first system call named `phyaddr()` returns the physical address of a given virtual address, and the second system call is `ptidx()` that returns the page table (or directory) index at the specified level of page table for the virtual address. The last system call is `pgcnt()`, and it returns the total number of pages currently in use by the process.
- Before starting the assignment, go to the `xv6-riscv/` directory, download the following script file, and execute it to update xv6-riscv. The script makes some minor changes to xv6-riscv for this assignment.

```
$ cd xv6-riscv/
$ wget https://icsl.yonsei.ac.kr/wp-content/uploads/paging.sh
$ chmod +x paging.sh
$ ./paging.sh
```

- Launch xv6-riscv, and try executing the `pgtest` program. This sample program is to validate your three system call implementations. Details of the messages will be explained later.

```
$ make qemu
qemu-system-riscv64 -machine virt -bios none -kernel kernel/kernel -m 128M -smp 1
-nographic -drive file=fs.img,if=none,format=raw,id=x0 -device virtio-blk-device,
drive=x0,bus=virtio-mmio-bus.0

EEE3535 Operating Systems: booting xv6-riscv kernel
EEE3535 Operating Systems: starting sh
$ pgtest
&array[N-1]:
    Virtual address = 0x0000000001B4057C
    Physical address = 0x0000000000000000
    PT index at level 2: 0
    PT index at level 1: 0
    PT index at level 0: 0

...

&main:
    Virtual address = 0x000000000000009E
    Physical address = 0x0000000000000000
    PT index at level 2: 0
    PT index at level 1: 0
    PT index at level 0: 0

Total number of pages = 0
```

- If xv6-riscv runs fine, terminate it by pressing `Ctrl+a` and then `x`.

2 Paging in xv6-riscv

- The system calls to implement in this assignment need to access page tables and related information of a process. To help you understand how the paging scheme works in xv6-rixcv, the following describes its overall implementation.

- **Multi-level page tables**
  - An address in xv6-riscv is 39 bits long. Since there is no such a data type to express 39-bit-long variable, xv6-riscv uses `uint64`, a 64-bit unsigned integer, for the 39-bit address. Hence, the upper 25 bits of 64-bit address (from bit #39 to #63) are simply zeros.
  - Each page table entry (PTE) or page directory entry (PDE) is 64 bits long. PTEs and PDEs are technically the same except that a PTE points to *a page* containing data, and a PDE points to *a page* that includes a page table.
  - To avoid the confusion due to the use of PTE and/or PDE terms, let us simply call all of them as PTEs. In other words, we will no longer use the word "PDE" in this document. Instead we will differentiate them such as L0 PTE (or PTE at level 0), L1 PTE, etc.
  - xv6-riscv refers to a page table indexed by the lowest bits of a virtual address as level 0 (or L0), and its upper-level is called L1, and so on.
  - Since an address is 39 bits long and each PTE has 64 bits, multi-level pages are indexed by the virtual address as follows. The lowest 12 bits are page offset for 4KB page. The next 9 bits are used for indexing L0 page table, and the next 9 bits are for L1, and again the next 9 bits are for L2. The upper-most 25 bits are unused.

Virtual address

Unused (zeros)	L2 PTE	L1 PTE	L0 PTE	Page offset
63 39	38 30	29 21	20 12	11 0

- **Page table entry**
  - A PTE in xv6-riscv is organized as follows. Lowest 10 bits of PTE are used as control bits such as `V` (Valid), `R` (Readable), `W` (Writable), `X` (Executable), `U` (User), `G` (Global), `A` (Accessed), `D` (Dirty), `RSW` (Reserved for supervisor software). The middle 44 bits of PTE store a page frame number (PFN), and the upper-most 10 bits are reserved and filled with zeros.

Page table entry

Reserved	Physical frame number (PFN)	RSW	D	A	G	U	X	W	R	V
63 54	53 10	9 8	7	6	5	4	3	2	1	0

  - In a PTE, what you need to extract from it is the PFN, and its validity must be checked by reading the bit poistion #0. If this bit is 0, it indicates the PTE points to an invalid page. Assume that no page swapping is used in xv6-riscv.
  - Other control bits are not used in this assignment, so you do not have to worry about their exact roles.
- **Address translation**
  - Most of paging-related implementations are written in the `kernel/vm.c` file, and it uses several macros in the `kernel/riscv.h` file for masking, shifting, etc.
  - Functions in `vm.c` such as `walk()` and `walkaddr()` show how to walk through multi-level page tables, and you will find them as very useful hints to do this assignment. Since these functions are written in straightforward manner, detailed explanations of them will be skipped.

3 Implementation

- In this assignment, you will have to implement three system calls that deal with page tables of a process. The skeleton codes of system calls are provided in the `kernel/sysproc.c` file, and all the necessary definitions to enable the system calls have been placed in appropriate header files.
- The following explains the expected operations of system calls to implement for this assignment.

- System call #1: **`void* phyaddr(void*);`**
  - This system call takes a memory address (e.g., pointer value) as it input argument. Since the data type of pointer is unknown, it takes the input as `void*`.
  - Obviously, the input argument from a user process should be a virtual address.
  - The system call is supposed to access the page tables of the process and return the translated *physical address* for the given virtual address.
- System call #2: **`int ptidx(void*, int);`**
  - This system call takes a memory address as its first argument, and the second argument indicates a page table level that should be 0, 1, or 2 representing L0, L1, or L2, respectively.
  - The system call returns the *PTE index of a page table at the specified level* (e.g., L0, L1, L2).
  - The return value of system call should obviously be between 0 and 511 for 512 PTEs in a page table.
  - This actually does not have to be a system call, since the PTE index at the specified level can be directly obtained from the given virtual address. Anyways, implement the system call that performs the explained operation.
- System call #3: **`int pgcnt(void);`**
  - This system call counts how many pages of the process are currently in use and returns the *total number of allocated pages*.
  - The system call takes no input arguments.
  - Counting the number of allocated pages requires walking through the entire multi-level page tables. However, the test program (i.e., `pgtest`) uses only a small number of pages, and thus most PTEs should have zero valid bits. If you encounter an invalid PTE at L2 page table, that saves a huge number of PTE checks along the page table hierarchy.

4 Submission

- In the `xv6-riscv/` directory, execute the `tar.sh` script. This script will compress the current `xv6-riscv/` directory into a tar file named after your student ID (e.g., `2020142020.tar`).
- Upload the created tar file on YSCEC. Do not rename the tar file by adding your name, project5, etc.

5 Grading Rules

- The following is a general guideline for grading the assignment. 30-point scale will be used for this assignment. The minimum score is zero, and negative scores will not be given. Grading rules are subject to change, and a grader may add a few extra rules for fair evaluation of students' efforts.

  -5 points: The tar file is renamed and includes some other tags such as a student name.

  -5 points: Program codes do not have sufficient comments. Comments in the baseline xv6-riscv do not count. You must make an effort to clearly explain what each part of your implementation intends to do.

  -10 points: The `phyaddr()` system call fails to give the correct physical address for a given virtual address.

  -10 points: The `ptidx()` system call does not give the right PTE index for a given virtual address.

  -10 points: The `pgcnt()` fails to return the correct number of pages in use.

  -30 points: No or late submission.

  F grade: A submitted code is copied from someone else. All students involved in the incident will be penalized and given F for final grades irrespective of assignments, attendance, etc.

- Your teaching assistant (TA) will grade your assignments. If you think your assignment score is incorrect for any reasons, discuss your concerns with the TA. Always be courteous when contacting the TA. In case no agreement is made between you and the TA, elevate the case to the instructor to review your assignment. Refer to the course website for the contact information of TA and instructor: `https://icsl.yonsei.ac.kr/eee3535`
- Begging for partial credits for no viable reasons will be treated as a cheating attempt, and thus such a student will lose all scores for the assignment.