

## CS 370 – Project #4

Purpose: Become familiar with operating virtual memory concepts.

Points: 125 Project #4 – Code (50 pts)  
Project #4 – Write-up (75 pts)

### Introduction:

Page tables are the mechanism through which the operating system provides each process with its own private address space and memory. Page tables determine what memory addresses mean, and what parts of physical memory can be accessed. They allow **xv6** to isolate different process's address spaces and to multiplex them onto a single physical memory.

We will create a new system call to print the page table for the process calling the system call.



### Resources:

The following resources provide more in depth information regarding **xv6**. They include the **xv6** reference book, tools for installing, and guidance to better understand **xv6**. Specifically, refer to chapter 3, Page Tables.

1. **xv6** Reference Book: <https://pdos.csail.mit.edu/6.828/2023/xv6/book-riscv-rev3.pdf>

### Project:

Complete the following actions.

- Implement **vmprint** system call.
- Implement **showTable** user level program (that uses the new **vmrpint** system call).

### Submission

When complete, submit:

- (50 points) An analysis PDF placed in the **xv6** folder
- (75 points) A copy of the zipped **xv6** folder via the class web page by class time.

Submissions received after the due date/time will not be accepted.

### Print Page Table (vmprint) System Call:

The print page table (**vmprint**) system call will be used to print the page table for the process that calls the system call. Your implementation of the **vmprint** system call will be located in **kernel/vm.c**. Since this function requires access to a protected memory and number of kernel functions, we will print from within the kernel. This is not good practice, but we will overlook it just this once. :-)

The system call will display a header line (see example) and print the valid entries in the current process pagetable at the specific instant the system service is called. Of course, these can change over time. The print page table routine is provided below. Make sure to review and understand what is being done here as it will be part of the final write-up.

```
#define PXROOTLEVEL      2

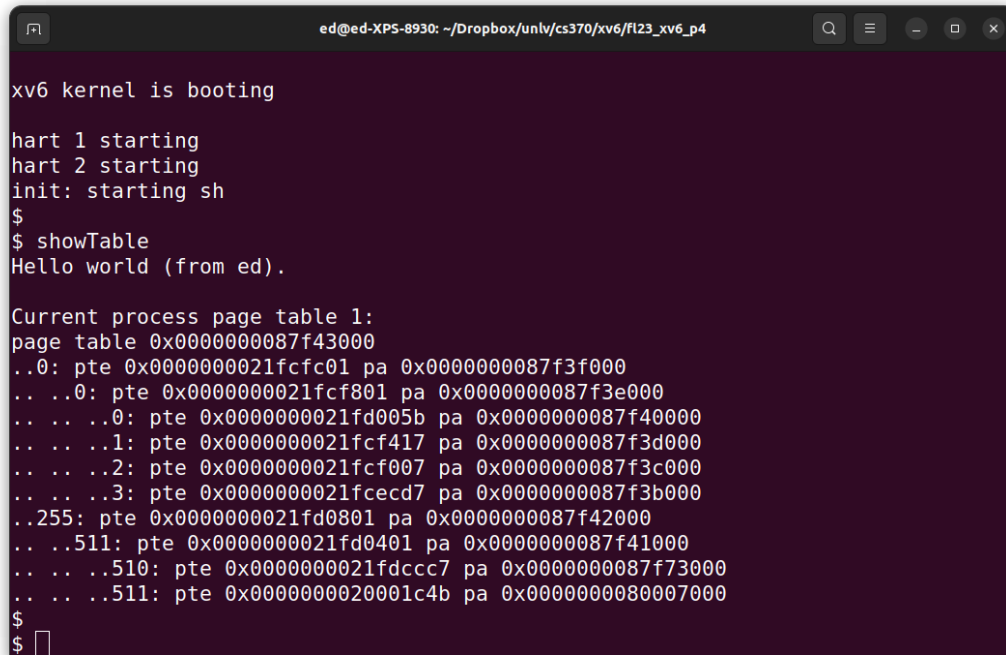
void vmprint(pagetable_t pagetable)
{
    static int level = 2;
    static int rootdirectory = 1;
    if(rootdirectory)
        printf("page table %p\n", pagetable);

    for(int i = 0; i < 512; i++){
        pte_t pte = pagetable[i];
        //int flg = pte - ((pte >> 10) << 10);
        if((pte & PTE_V) && (pte & (PTE_R|PTE_W|PTE_X)) == 0){
            uint64 child = PTE2PA(pte);
            for(int j = PXROOTLEVEL; j > level; j--){
                printf(".. ");
                printf("..%d: pte %p pa %p\n", i, pte, child);
                level--;
                rootdirectory = 0;
                vmprint((pagetable_t) child);
            } else if(pte & PTE_V){
                level = 0;
                for(int j = PXROOTLEVEL; j > level; j--){
                    printf(".. ");
                    printf("..%d: pte %p pa %p\n", i, pte, PTE2PA(pte));
                    level = PXROOTLEVEL;
                }
            }
        }
    }
}
```

Since this is a new system call, you will need to add the necessary stubs to fully implement the system call (similar previous projects). This will include the **user.h**, **usys.pl**, **syscall.h**, and **syscall.c** files.

### User-Level Test Program:

The user level program to test the new system call, **showTable**, will use the new **vmrpting** system call and display the page table for the current processes in the **xv6** system. You should display a “Hello World (from <yourName>)” header with your name as shown below.



```
ed@ed-XPS-8930: ~/Dropbox/unlv/cs370/xv6/fl23_xv6_p4
xv6 kernel is booting
hart 1 starting
hart 2 starting
init: starting sh
$
$ showTable
Hello world (from ed).

Current process page table 1:
page table 0x0000000087f43000
..0: pte 0x0000000021fcfc01 pa 0x0000000087f3f000
.. ..0: pte 0x0000000021fcf801 pa 0x0000000087f3e000
.. ..0: pte 0x0000000021fd005b pa 0x0000000087f40000
.. ..1: pte 0x0000000021fcf417 pa 0x0000000087f3d000
.. ..2: pte 0x0000000021fcf007 pa 0x0000000087f3c000
.. ..3: pte 0x0000000021fced7 pa 0x0000000087f3b000
..255: pte 0x0000000021fd0801 pa 0x0000000087f42000
.. ..511: pte 0x0000000021fd0401 pa 0x0000000087f41000
.. ..510: pte 0x0000000021fdccc7 pa 0x0000000087f73000
.. ..511: pte 0x0000000020001c4b pa 0x0000000080007000
$
$
```

Modify the **Makefile** as necessary (also similar to Project #1).

### Project Setup:

This project will be done entirely in **xv6**. To get started with this project first follow the instructions below:

- **sudo apt-get update**
- **sudo apt-get upgrade**
- **git clone https://github.com/mit-pdos/xv6-riscv.git xv6\_p4**
- **chmod 700 -R xv6\_p4**

*Note*, the **xv6\_p4** directory may be changed as desired.

## Summary of Updates Files

As described, you will be either updating or creating the following files:

```
Makefile
user/user.h
user/usys.pl
kernel/syscall.h
kernel/syscall.c
kernel/sysproc.c
kernel/defs.h
```

Pay close attention to user space vs kernel space.

## Final Report

Prepare a final report including the below sections. The report should be 1-2 pages, PDF format, 12pt font, and single spaced.

- Results
  - Example output of the **showTable** system program (showing your name).
- Summary
  - Explanation of changes made.
  - Explanation of the provided code.
    - Include an overview of the **xv6** process memory map.
    - *Note*, must address what the PTE\_V is, the PTE2PA() function, why there are multiple (3) “levels”, and why recursion was used.

The report should include appropriate titles and section headers. Additionally, spelling and grammar will be part of the final report score.