CSC 246 Spring 2019 Homework 4

Due: March 22 2019, 11:55PM

ChangeLog

03/20: clarify that both prob_3(.c) and proc_3(.c) are acceptable file names.

03/03: homework 4 online, and it is due on the Friday after the Spring break.

Problem 1

Assuming a system with the following setup: its virtual address is 32 bits long, its page size is 4KB, and it has a linear page table with page table entries (PTE) of size 4 bytes.

1. How many bits are for the offset?
2. How many bits are for the virtual page number (VPN)?
3. How many entries does a linear page table have?
4. What is the total size of the page table of one process?
5. In a live system, how much memory will be occupied by page tables?

Put your answers in problems.txt (ASCII file).

Problem 2

Assuming another system with the following setup: its virtual address is 32 bits long, its page size is 4KB, it has a linear page table, its TLB has 32 entries, its physical memory has 1024 pages, and the LRU replacement policy is used whenever such a policy might be needed. In this problem, we will try to understand what happens while running some code.

Before running the test code, we first run the following initialization code once. The initialization code allocates some amount of memory and sets the first interger of each page to 0. Assuming malloc() returns page-aligned memory in this example.

```
// allocate NUM_PAGES*PAGE_SIZE bytes, where PAGE_SIZE is 4KB
void *orig = malloc(NUM_PAGES * PAGE_SIZE);

int *ptr = (int *) orig;
for (i = 0; i < NUM_PAGES; i++) {
  *ptr = 0; // init first value on each page
  ptr += PAGE_SIZE;
}
```

After running the initialization code above once, we run the following test code:

```
ptr = (int *) orig;
for (i = 0; i < NUM_PAGES; i++) {
  int x = *ptr; // load value pointed to by ptr
```

```
  ptr += PAGE_SIZE;
}
```

We are only interested in memory accesses to the malloc'd region through ptr and ignore all other memory accesses, i.e., we assume other memory accesses do not affect the 32-entry TLB and 1024-page physical memory. We further assume that memory and TLB start from a clean state before we run the initialization code.

1. How many TLB hits, TLB misses, and page faults occur during the test code when NUM_PAGES is 16, 32, and 2048, respectively? Your answers include 9 numbers.
2. If a memory access takes roughly time M and a disk access takes time D, how long does the test code take to run, in terms of M and D and ignoring other costs, when NUM_PAGES is 16, 32, and 2048, respectively? Your answers include 3 formulas.
3. If we change the various replacement policies in the system to MRU. How long does the test code take to run, in terms of M and D and ignoring other costs, when NUM_PAGES is 16, 32, and 2048, respectively? Your answers include 3 formulas.

Put your answers in problems.txt (ASCII file).

Problem 3

In this problem, you will write a C program (C++ is fine as well) that can show the memory accesses with a multi-level page table, just as the -c or --solve mode in paging-multilevel-translate.py. We will have an in-class exercise on a memory image generated by this python program before the Spring break.

Since paging-multilevel-translate.py solves the problem while it is generating the program, we define the input of the C program differently, and our C program takes three parameters: a file path, the value of PDBR, and a virtual address to translate, where the file path pointing to a file that contains the 128 lines starting with "page". The output, including format, should be the same as the one generated with the -c or --solve mode given the same memory image, PDBR value, and virtual address. For example:

```
./proc_3 /path/to/the/128/lines 118 0x05e1
Virtual Address 0x05e1:
  --> pde index:0x1 [decimal 1] pde contents:0xc1 (valid 1, pfn 0x41 [decimal 65]
    --> pte index:0xf [decimal 15] pte contents:0xab (valid 1, pfn 0x2b [decimal 4
      --> Translates to Physical Address 0x561 --> Value: 1e

./proc_3 /path/to/the/128/lines 118 0x210b
Virtual Address 0x210b:
  --> pde index:0x8 [decimal 8] pde contents:0xd2 (valid 1, pfn 0x52 [decimal 82]
    --> pte index:0x8 [decimal 8] pte contents:0x7f (valid 0, pfn 0x7f [decimal 12
      --> Fault (page table entry not valid)
```

where the file /path/to/the/128/lines contains the content from "./paging-multilevel-translate.py -s 246 -n 5". Note 118 is a decimal number, and the output format above is slightly different from the python program that we are adding "0x" for all hexadecimal numbers. As usual, please print meaningful error messages for common cases like wrong file name, wrong file format, and wrong parameters. You can also handle other error cases when you see necessary.

After you finish, turn in your proc_3.c, README, and Makefile. Since I usually name the files as prob_3.c and

prob_3 but not proc_3.c and proc_3, we will accept both names.