CPE 434/534 Homework Assignment 3

Due Tuesday, Oct. 6th

Student Name \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Attach this sheet to your submission as the cover page.

1. \_\_\_\_\_(3 points) Assume a 32 bit virtual address with a three level page table indexed by fields of 7, 7, 6 bits with a 4k page size (12 bits).

*A mmu with 4 fields of size 7, 7, 6 has first level page tables of size 2\*\*7x4 (512), second level tables of size 2\*\*7x4 9512) and third level tables of size s\*\*6\*4 (256).*

1. \_\_\_\_(1 point)What is the size of the page table structure for the smallest program

*The smallest program has one page starting at address 0 for its text and heap and one page starting at ffffffff for its stack. So the smallest program needs 1x512 +3x512 +2+256 bytes of page table structure.*

1. \_\_\_\_(1 point)What is the size of the page table structure for the largest program

*The largest program has all of the address space mapped. The size of the page tables are dominated by the final tables, all of which are populated, which (2\*\*7)\*(2\*\*7)\*(2\*\*6) entries each of 4 bytes, about 4Million. The other tables contribute some to the size but are smaller by at least a factor of 64.*

1. \_\_\_\_(1 point)How would this change if you used fields of 6, 7, 7 bits instead of 7, 7, 6 bits

*The size of the tables would not change much. For the smallest there would be one table of size 2\*\*6, and four tables of size 2\*\*7. It would not change for the largest table.*

1. \_\_\_\_\_ (4 points) Assume you have two processes, a, b, with a shared memory segment of several pages.

` *to make the problem tractable, I will assume there is only one core and only one process runs at a time. If this were not the case then we would have to do all the updating whenever a fault occurred in any program. even harder would be the need to update the modified and referenced bits which are maintained by the tlb and would have to be flushed on every update. This information is needed to decide whether to write a page to disk if the space is being recovered and needs to be accurate across all processes.*

*One generic solution that was not reasonable was to make all shared pages non-writable. It would not be reasonable for the operating system to prevent a program from writing to shared pages since it is used particularly for sharing data between processes.*

*Another generic solution might have been to make all shared pages non-pagable but I do not recall seeing that solution from any student.*

a-\_\_\_\_\_(1 point) if a page fault occurs in process’s a shared memory segment, should you update process’s b’s shared memory segment page tables accordingly. Remember, the page

tables are distinct for each process, even if they point to the same memory locations.

*With only one process running it probably makes sense to update the page tables of the non-running process at the time of the fault. It could be delayed if you wish until the next process is restarted since that is when the second process needs up to date page tables. But since it is likely to be done it can be done at fault time.*

B\_\_\_\_\_(1 point) if so, what kind of structure would you use to keep track of the shared pages for this purpose

*Because the page table information (r,m) is common across all users of the shared data a separate table needs to be kept with this information (duplicated) since it cannot be kept just with the page tables for one process.*

C- \_\_\_\_(1 point) how would your answer change if the page were shared by hundreds of processes

*If there are hundreds of processes sharing a set of pages then these probably should be updated only when a process is restarted. With that many processes it is possible that a page might come and go from memory multiple times before any one process needs to access it.*

d-\_\_\_\_(1 point) how would this change for shared libraries

*shared libraries do not have this problem since they are non-pagable.*

3- \_\_\_\_\_\_ (2 points) text 3-11

For the following program

Int x[M];

Int step = M;

For (int I = 0;I,N,i+=step) x[i] = x[i]+1

1. If this program is run on a machine with 4KB pages and a 64 entry TLB what size of M and N will cause a fault for every iteration of the inner loop.

*If the step size was 4KB then each access would touch another page and cause a fault.*

1. How would your answer change if the loop was executed many times

*The size of the array would have to be of size 4KBx64 to make sure every TLB entry was touched before reuse.*

4- \_\_\_\_\_\_(2 points) text 3-16

Given that a TLB has 1024 entries and can be accessed in 1ns, a page table lookup in main memory takes 100ns, and the average page replacement time is 6ms.

If the TLB handles 99% of all accesses, and only .01% of those accesses lead to a page fault, what is the average access time to memory.

*Time = 0.99\*1 + 0.01\*100 + 0.0001\*6,000,000 which is about 602 ns.*

5- \_\_\_\_\_(2 points) while you cannot predict the future use of pages, some computer systems allowed the programmer to predict the future address space footprint so the system could prepage the requested pages. Programmers, however, are sometimes confused and sometimes lie. How would you reduce the effect of these two possible errors in the predictions of the programmer

*For this question I was looking for a technical answer so firing the programmer was not an acceptable solution. Using the working set algorithm to confirm the reasonableness of the programmers estimates would be a good solution. Note, we are not using the working set algorithm since it would inhibit valid pre-paging estimates, we are only using it to verify that the programmer estimates are reasonable.*

6- \_\_\_\_\_(two points) text 3-28

If you have a memory system with 4 memory slots and 8 valid addresses.

1. What would be the page fault for an access string of 0172327103 for a FIFO algorithm

0 fault, 01 fault, 017 fault, 0172 fault, 1723 fault, 2 (in store, no fault), 7 (in store, no fault), 1 (in store, no fault), 7230 fault, 3 (in store, no fault) total of 6 faults

1. How would that change for LRU

0 fault, 01 fault, 017 fault, 0172 fault, 1723 fault, 1732 (no fault), 1327 (no fault), 3271 (no fault), 2710 (fault), 7103 (fault) total of 7 faults.