Operating Systems Quiz3 ECE344, Winter 2021

Duration: 1 hour

Examiner: D. Yuan

Instructions

Examination Aids: This is an open book exam.

All questions have been provided in this exam booklet. You need to provide your answers in Quercus.

If any of the questions appear unclear or ambiguous to you, then make any assumptions you need, state them and answer the question that way. If you believe there is an error, state what the error is, fix it, and respond as if fixed.

Please be brief and specific as possible. Clear, concise answers will be given higher marks than vague, wordy answers. Marks will be deducted for incorrect statements in an answer.

Work independently.

MARKING GUIDE

Q1: (1)
Q2: (1)
Q3: (7)
Q4: (3)
Q5: (3)
TOTAL: (15)

In this quiz, you will walk through memory management steps for a simple computer. Assume the following:

- The size of virtual address space is 256 bytes
- The page size is 16 bytes
- The size of physical memory is 96 bytes
- The allocation of a page fame (i.e., physical page) is as following:
    - Whenever there is an unallocated page frame, allocate it
    - If there are multiple unallocated page frames, allocate the one with the smallest address
    - When all page frames are allocated, the OS will evict the **least recently used** frame and allocate it
- When there is a context switch, the TLB is flushed
- Copy-on-write is used for fork()
- The TLB has 4 entries, numbered from 0 to 3. Upon a TLB miss, when there are multiple entries that are invalid, the MMU will always allocate the entry with the smallest entry number.

Question 1 (1 point): How many bits does a virtual address have? And how many bits are used for the virtual page number (VPN)?

Answer: 8 bits for virtual address. 4 bits are used for VPN.
Marking: 0.5 points each.

Question 2 (1 point): How many page frames are there in the physical memory?

Answer: 6 page frames.
Marking: binary.

Question 3 (7 points): Consider the following sequence of memory accesses, which are made from the same process without a context switch in between. You will need to answer:

- **PA**: What's the physical address after the translation
- **Hit/miss**: Whether the access results in a (1) TLB hit (TLBH), (2) TLB miss (TLBM), (3) Page Fault (PF), or a combination of these three
- The **TLB entries** that are valid **after** the memory access finishes. You should specify each valid entry using the following form: VPN:PFN. For example, a "3:4" indicates that virtual page 3 is mapped to page frame 4.
- The valid **page table mappings** after the memory access finishes. Use the same format as the TLB entries: for example: "3:4" indicates that virtual page 3 is mapped to frame 4.

At the beginning everything, including TLB, physical memory, page table, is empty.

Mem. access	PA	Hit/miss	TLB entries 0 1 2 3	Page table mappings
load 0x10	0x00	TLBM PF	1:0	1:0
load 0x20	0x10	TLBM PF	1:0 2:1	1:0 2:1
load 0x14	*0x04*	*TLBH*	*1:0 2:1*	*1:0 2:1*
load 0xF4	*0x24*	*TLBM PF*	*1:0 2:1 F:2*	*1:0 2:1 F:2*
load 0x1A	*0x0A*	*TLBH*	*1:0 2:1 F:2*	*1:0 2:1 F:2*
load 0x20	*0x10*	*TLBH*	*1:0 2:1 F:2*	*1:0 2:1 F:2*
load 0xA2	*0x32*	*TLBM PF*	*1:0 2:1 F:2 A:3*	*1:0 2:1 A:3 F:2*
load 0x38	*0x48*	*TLBM PF*	*1:0 2:1 3:4 A:3*	*1:0 2:1 3:4 A:3 F:2*
store 0xAA	*0x3A*	*TLBH*	*1:0 2:1 3:4 A:3*	*1:0 2:1 3:4 A:3 F:2*
load 0x64	*0x54*	*TLBM PF*	*6:5 2:1 3:4 A:3*	*1:0 2:1 3:4 6:5 A:3 F:2*
store 0x1A	*0x0A*	*TLBM*	*6:5 1:0 3:4 A:3*	*1:0 2:1 3:4 6:5 A:3 F:2*
store 0xFA	*0x2A*	*TLBM*	*6:5 1:0 F:2 A:3*	*1:0 2:1 3:4 6:5 A:3 F:2*
load 0x50	*0x10*	*TLBM PF*	*6:5 1:0 F:2 5:1*	*1:0 3:4 5:1 6:5 A:3 F:2 // Page replacement, 2:1 was the victim page*
load 0x14	*0x04*	*TLBH*	*Same as above*	*Same as above*
load 0xBF	*0x4F*	*TLBM PF*	*B:4 1:0 F:2 5:1*	*1:0 5:1 6:5 A:3 B:4 F:2 // Page replacement, 3:4 was the victim page*
store 0x24	*0x34*	*TLBM PF*	*B:4 1:0 2:3 5:1*	*1:0 2:3 5:1 6:5 B:4 F:2 // Page replacement, A:3 was the victim page*

The first 2 rows are already filled in for you: the first memory access, which reads the content at virtual address 0x10, results in a TLB miss (TLBM) and a page fault (PF). The address 0x10 is mapped to physical address 0x00. After this instruction, the TLB has 1 valid entry at entry 0 that has 1:0, which maps virtual page 1 to page frame 0, and this mapping is also in the page table. After the second instruction, the TLB has 2 valid entries: 1:0 at entry 0 and 2:1 at entry 1.

Your job is to complete the table for the remaining memory accesses. You can write your answer in plain text, with one row for each entry, separate columns with a comma ",". For example, the first two rows can be written as:

```
 load  0x10, 0x00, TLBM PF, 1:0, 1:0
 load  0x20, 0x10, TLBM PF, 1:0 2:1, 1:0 2:1
```

Marking: 0.5 marks for each row.
Accept any TLB replacement policy.
Also, note that TLB entries cannot be rearranged. For example, the second row's TLB entries are 1:0 2:1, but 2:1 1:0 is not correct (the new entry, 2:1, should occupy entry 1 instead of entry 0 as entry 0 was occupied by 1:0).

Question 4 (3 marks): Now assume we use a two-level page table. The first half of the bits in the VPN will be used to index the first-level page table, whereas the other half of the bits in VPN are used to index the second-level page table.

(A) How many entries are there in each level of page tables?

Answer: 4 entries in each level page tables.
Marking: binary, answer depending on Q3.

(B) How many entries are **valid** in the first-level page table after the above memory access sequence completes?

Answer: all 4 entries are valid.
Marking: depend on their answer in Q3

(C) For the **first** second-level page table, list all entries that are valid at the end of the above access sequence in the following format:

- 0:3
- 2:1
- 3:4

The above sample answer indicates that there are 3 entries that are valid: the first entry (index 0) maps to page fame 3, the 3rd entry (index 2) maps to page frame 1, and the 4th entry (index 3) maps to page frame 4.

1:0
2:3

Marking: 0.5 points off each mistake

Question 5 (3 marks): After the above memory access sequence, the process calls **fork**(), and the child process is immediately scheduled to execute on the same core. It issues the following sequence of memory accesses:

Mem. access	PA	Hit/miss	TLB entries 0 1 2 3	Page table mappings
load 0x22	*0x32*	*TLBM*	*2:3*	*1:0 2:3 5:1 6:5 B:4 F:2*
load 0xFF	*0x2F*	*TLBM*	*2:3 F:2*	*Same as above*
store 0xFF	*0x5F*	*TLBM PF*	*2:3 F:5*	*1:0 2:3 5:1 B:4* ***F:5*** *// COW, page replacement, 6:5 is the victim page*
load 0xF0	*0x50*	*TLBH*	*Same as above*	*Same as above*
load 0x20	*0x30*	*TLBH*	*Same as above*	*Same as above*
load 0x62	*0x12*	*TLBM PF*	*2:3 F:5 6:1*	*1:0 2:3 6:1 B:4* **F:5** *// Page replacement, 5:1 is the victim page*

Complete this table. Note that in the last column, Page table mapping, you only need to provide the mappings for this child process.

Marking:
-1, not flushing TLB
-1, if not knowing that the initial page mapping should be the same as end of Q3 due to copy on write
-1, if not knowing that the store at row #3 triggers a PF.
-3, if non-sense (e.g. the same values copy and pasted into each row, even though it might be correct for some cells)

-half of marks if there is explanation but no actual table values (probably due to time constraints)