

ICS EXE 12

June 2, 2021

1 Locking

What are problems in the following simple implementation of lock?

```
1 typedef struct __lock_t {
2     int flag;
3 } lock_t;
4
5 void init (lock_t *mutex) {
6     /* 0 -> lock is available 1 -> held */
7     mutex->flag = 0;
8 }
9
10 void lock (lock_t *mutex) {
11     /* spin-wait (do nothing) */
12     while (mutex->flag == 1) ;
13     /* now SET it! */
14     mutex->flag = 1;
15 }
16
17 void unlock(lock_t *mutex) { mutex->flag = 0; }
```

2 Address Translation

This problem concerns the way virtual addresses are translated into physical addresses. Below are the specifications of the system on which the translation occurs:

- The main memory is byte addressable.
- The memory accesses are to **1-byte** words (not 4-byte words).
- The system uses **one-level** page table.
- Virtual addresses are **25 bits** wide.
- PPN is **9 bits** wide.
- The amount of PTE is 2^{14} .
- The TLB is **4-way** set associative with **16** total entries.

2.1

The VPO bits	_____
The physical address bits	_____
The page size	_____
The number of bits for TLBT	_____

The contents of the TLB and first 16 entries of the page table are given below. All numbers are in hexadecimal. According to the illustration and answer the questions. Please fill in the blanks.

Set	Tag	PPN	Valid	Tag	PPN	Valid
0	0CD	09	1	AA1	00	1
	3E0	62	0	C4C	48	1
1	312	45	0	010	75	1
	987	3A	1	D39	3F	0
2	038	E3	0	0A7	13	1
	18B	52	1	49B	11	0
3	6C0	42	0	075	50	0
	013	39	1	0F2	0D	0

TLB: 4 sets, 16 entries, 4-way set associative

VPN	PPN	Valid	VPN	PPN	Valid
00	39	1	08	0D	0
01	52	1	09	45	0
02	E3	0	0A	13	1
03	00	1	0B	3A	1
04	11	0	0C	09	1
05	50	0	0D	42	0
06	62	0	0E	3F	0
07	75	1	0F	48	1

Page Table: Only the first 16 PTEs are shown

2.2

Please translate virtual address to physical address and get the data from cache if hit (otherwise entering '-')

Parameter	Value
Virtual Address	0x014F2D3
VPN	0x_____
TLB Index	0x_____
TLB Tag	0x_____
TLB Hit? (Y/N)	____
Page Fault? (Y/N)	____
PPN	0x_____
Physical Address	0x_____

Parameter	Value
Virtual Address	0x07C01A5
VPN	0x_____
TLB Index	0x_____
TLB Tag	0x_____
TLB Hit? (Y/N)	____
Page Fault? (Y/N)	____
PPN	0x_____
Physical Address	0x_____

3 Virtual Address

Suppose the virtual addresses **32 bits** wide. The page size is **4kB**. The VPN is **10 bits** wide. Please answer the following questions.

```

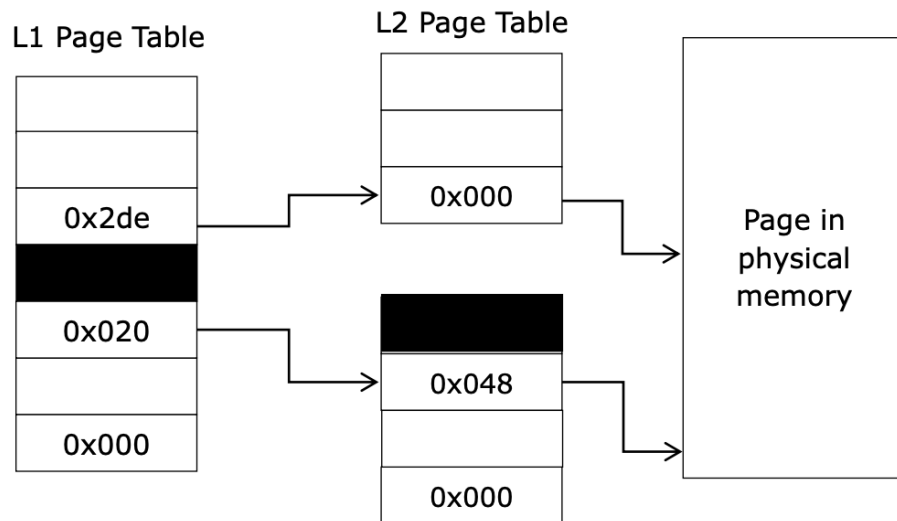
1  #include <unistd.h>
2  #include <sys/mman.h>
3  #include <stdio.h>
4
5  #define ARRAY_LEN (125*1024)
6
7  int main(void) {
8      int idx;
9      int fd = open("ics.txt", O_RDWR);
10 A:
11     char* sbuf = mmap(0, ARRAY_LEN,
12                       PROT_READ | PROT_WRITE, MAP_SHARED, fd, 0);
13     char* pbuf = mmap(0, ARRAY_LEN,
14                       PROT_READ | PROT_WRITE, MAP_PRIVATE, fd, 0);
15
16 B:
17     for (idx = 0; idx < ARRAY_LEN; idx++) {
18         sbuf[idx] = sbuf[idx] + 1;

```

```

19     pbuf[idx] = pbuf[idx] + 1;
20 }
21
22 C:
23     fork();
24
25 D:
26     for (idx = 0; idx < ARRAY_LEN; idx++) {
27         pbuf[idx] = pbuf[idx] + 1;
28     }
29
30 E:
31     munmap(sbuf, ARRAY_LEN);
32     munmap(pbuf, ARRAY_LEN);
33     close(fd);
34     return 0;
35 }

```



Assumption:

- 1) After setting **MAP_SHARED** flag, the updates to the shared mapping area will be immediately synchronized to the file.
- 2) After setting **MAP_PRIVATE** flag, the changes made to the file after **mmap()** call and before copy-on-write are visible to the mapped area. The modification on the mapped area memory will cause a copy-on-write (COW) mapping.
- 3) The ics.txt is a file filled with a lot of character '0' and its size is more than **ARRAY_LEN**.

The address of **sbuf** is **0xb7be2000** and that of **pbuf** is **0xb7cd4000** before label **B**. The following figure shows part of the page table when the program arrives at label **A**. The number within block is the index of page table. The white block without number means one or more empty page table entries. Please answer the following questions. (**NOTE** please ignore the page fault on the stack for the following questions)

3.1

Please fill in the following blanks.

The index of L1 PTE for pbuf is _____

3.2

How many page fault exception are raised by code between label B and C? How many of them will handle COW? Please also write down your explanation.

3.3

How many page fault exception **at most** are raised by code between label D and E of the parent process? How many of them will handle COW **at most**? Please also write down your explanation.

3.4

Please write down the value of below variables of process when the program reach label C.

sbuf[0] = _____ **sbuf**[1] = _____

pbuf[0] = _____ **pbuf**[1] = _____

3.5

Please draw a graph like above to show the page table of the process when the program reach label **E**. **Note** that you can use a **black** block to represent the consecutive filled PDE/PTEs. And use an empty block to represent the consecutive unfilled PDE/PTEs. (For example, you can draw (0x1) (black block) (0xA) to represent the consecutive filled entries from 0x1 to 0xA)