Replacement Policy (10 points)

Assume we have a primary device with 4 physical blocks, please complete the following questions.

1. Suppose we are using **LRU** replacement policy, please complete the following table. (**NOTE**: you do not need to consider the order of primary device contents). (1'*5=5')

Reference String	**1**	**4**	**5**	**3**	**1**	**5**	**1**	**8**
Device Content	1 - -	1 4 -	1 4 5	**[1]**	**[2]**	**[3]**	**[4]**	**[5]**

2. Suppose we are using clock algorithm with the following rules:
   - The clock pointer points to position 0 initially.
   - We do not reset the clock pointer after we have found an evict page, so next time we start from the position after previous victim.
   - Each page brought in were set as accessed initially.

   Please complete the following table. (**NOTE**: use "*" to represent current clock pointer). (0.5'*10=5')

Reference String	1	4	5	3	1	5	1	8
Device Content	1 * -	1 4 *	1* 4 5	**[6]**	**[8]**	**[10]**	**[12]**	**[14]**
Reference Bit	1 0 0	1 1 0	1 1 1	**[7]**	**[9]**	**[11]**	**[13]**	**[15]**

Address Translation

Assume we have a machine with the following specifications:

- The memory is byte-addressable
- The memory accesses are to **1-byte** words (not 4-byte words)
- The system uses a **two-level** page table
- Each page is **1024 bytes**
- The size of one page table equals to the size of page
- Length of each PTE is **8 bytes**
- PPN is **10 bits** wide
- The TLB is **4-way** set associative with **16** total entries
- The following figure shows the format of virtual address:

VPN-1	VPN-2	VPO

1. Warm-up questions. (2'*5=10')

The number of PTEs in one page table	**[1]**
The VPO bits	**[2]**
The virtual address bits	**[3]**
The physical address bits	**[4]**
The TLB tag bits	**[5]**

2. Contents of TLB are given below. All numbers are in hexadecimal. Refer to the given information and fill in the blanks. If the value is unknown or meaningless, enter "--" for them. **NOTE:** Accesses are independent, you **don't** need to consider TLB eviction after previous accesses. Besides, you **don't** need to consider cache here.

Set	Tag	PPN	Valid	Tag	PPN	Valid
0	7dd	ee	0	f2	2d0	1
	--	--	--	4ca	9f	1
1	83b	ac	1	189	44	0
	37e	43	1	33d	18a	1
2	3d	2f	0	5e	3b	1
	4c3	19e	0	--	--	--
3	f28	233	1	--	--	--
	d4c	3f	1	37e	1f8	0

Please translate virtual addresses given below to physical addresses.(2'*8=16')

Parameter	Value
Virtual Address	0x37ec48

TLB Tag	0x__**[6]**__
TLB Hit? (Y/N)	__**[7]**__
Page Fault? (Y/N)	__**[8]**__
Physical Address	0x__**[9]**__

Parameter	Value
Virtual Address	0xf23f3
TLB Tag	**0x__[10]__**
TLB Hit? (Y/N)	__**[11]**__
Page Fault? (Y/N)	__**[12]**__
Physical Address	**0x__[13]__**

Memory Mapping

```
1. #define BUF_SIZE (16*4096)
2. int main(void) {
3.    int fd1 = open("ics.txt", O_RDWR);
4.    int fd2 = open("ics2.txt", O_RDWR);
5.
6.    char *sbuf = mmap(0, BUF_SIZE, PROT_READ | PROT_WRITE,
7.                      MAP_SHARED, fd1, 0);
8.    char *pbuf = mmap(0, BUF_SIZE, PROT_READ | PROT_WRITE,
9.                      MAP_PRIVATE, fd2, 0);
10.
11.   for (int i = 0; i < BUF_SIZE; i++) {
12.      pbuf[i] = pbuf[i] + sbuf[i];
13.   }
14.
15.   if (fork() == 0) {
16.      sbuf[0]++;
17.      pbuf[0]*=2;
18.   }
19.
20.   printf("pbuf[0] = %d\n", (int)pubf[0]);
21.   printf("sbuf[0] = %d\n", (int)sbuf[0]);
22.   return 0;
23.}
```

We have the following assumptions:

- The program runs on an Intel **x86_64** Linux system.
- Sizes of file ics.txt and ics2.txt are both larger than **BUF_SIZE**.

- Before the execution, ics.txt is filled with **0x1** while **ics2.txt** is filled with **0x4** for each byte.
- After fork, the child process is always executed first.
- Only after the child process exits, the parent process will continue to execute.
- After setting **MAP_SHARED** flag, updates to the shared mapping area will be immediately synchronized to the file.
- 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 will cause a copy-on-write (COW) mapping.
- The address of **sbuf** is **0x1ffc0000000**. The address of **pbuf** is **0x1ffc0040000**.
- You **don't** need to consider page faults on the stack or code.

1. How many page fault exceptions are raised between **line 11 and 13**? How many page faults are caused by COW? (2'*2=4')

   Total page faults: __[1]__ **COW**: __[2]__

2. After the execution, what's the output of parent and child process? (2'*4=8')

   PARENT: **pbuf[0]** = __[3]__ **sbuf[0]** = __[4]__

   CHILD: **pbuf[0]** = __[5]__ **sbuf[0]** = __[6]__

3. After execution, what will we get if we read a byte in **ics2.txt** from beginning? (2')