Assignment 04

Part I Non programming (**60** points)

You must work on this part I on you own

Due: **11:59pm**, **Apr 21st**

Part I 5 questions – **12** points each

Make sure to include the **single examinee affidavit** at the beginning of your answer sheet. You must work on your own for this part, it would be a red flag if we find submissions from two students are with exact same incorrect answers in multiple parts.

1. Below is a 1-level page table for a 32-bit virtual address space, with each page using 28 bit.

Page #	Frame #	Valid	Referenced	Modified	Executable	Last access time in virtual time
0x0	0x3	1	0	1	1	1234
0x1		0	0	0	1	0
0x2		0	0	0	1	0
0x3	0x4	1	1	0	0	1351
0x4	0x1	1	0	0	0	1303
0x5	0x8	1	0	1	1	1294
0x6		0	0	0	1	0
0x7	0x6	1	1	0	0	1362
0x8		0	0	0	0	0
0x9		0	0	0	0	0
0xA	0x5	1	0	0	0	1287
0XB	0x9	1	1	1	0	1410
0xC	0x7	1	1	0	0	1335
➔0xD	0xB	1	1	1	0	1427
0xE		0	0	0	0	0
0xF		0	0	0	0	0

Use WSClock page replacement algorithm to select a victim frame and report the results.

Imagine page entries (rows) are arranged in clock positions, with the clock hand starting from **page D**, walking the clock by moving down in the table one page at a time. After reaching the end of the table, the clock hand moves to page 0x0 (the beginning of the

table) and moves down from there.
The last virtual time each page is referenced is put in the last column (note this column is not part of the page table). A daemon periodically copies referenced bit, modified bit, and virtual time to a Clock data structure for the WSClock algorithm to use.

Assuming the current virtual time is **1500**, and the age threshold in virtual time unit for the WSClock algorithm is **200**.

Simulate the algorithm:

1) Show details of the algorithm walking through entries until selecting the victim frame, you must indicate the action taken for each page entry (use the following table) as a result from the execution of the algorithm:
    a. update to the page entry
    b. any page write that was scheduled
    c. reason why the action was taken
    d. put no action if no action was taken
2) Report the victim frame selected by the algorithm and indicate why.

clock hand at a page	Action
0xD	
0xE	
0xF	
0x0	
…	
…	
…	

2. 4 pages are loaded in memory and are accessed at the following times:

```
page 0: 2, 14, 46
page 1: 6, 25
page 2: 15, 21, 33
page 3: 8, 12, 18
```

Use a not frequently used (NFU) with aging page replacement algorithm with:

- 10 ms periodic page access history updates: 10ms, 20, 30, 40, 50, …., etc.
- 4 history bits for representing the page access history

Find the victim page if a page fault occurred during a particular interval between two history updates: i.e., between 10m and 20ms, **or** between 20ms and 30ms, etc. (Assuming there was ONLY ONE page fault occurred through the access history)

Use the following table to work out your solution. Note the history bit string is updated only once with every history update even the page could be referenced multiple times during the interval of the history updates.

Time of history update	If a page fault happened during	Page	Referenced during interval before history update?	History (4 bit string)	Victim
10ms	10 ms – 20 ms	0			
		1			
		2			
		3			
20ms	20 ms – 30 ms	0			
		1			
		2			
		3			
30ms	30 ms – 40 ms	0			
		1			
		2			
		3			
40ms	40 ms – 50 ms	0			
		1			
		2			
		3			
50ms	50 ms – 60 ms	0			
		1			
		2			
		3			

3. Following pseudocode has thread t1 launch threads t2 and t3:

```
t1func(…) {
    start thread t2 executing t2func(t2 args);
    start thread t3 executing t3func(t3 args);

    do_work;

    // t1point
}

t2func(…) {
    do_work;

    // t2point
    do_more_work;
}

t3func(…) {
    do_work;

    // t3point;
}
```

Use semaphores to add synchronization code so that t1 **cannot** reach t1point before t2 reaches t2point and t3 reaches t3point.

4. Do the following entry and exit (pseudocode) subroutines provide a critical region to processes calling them on a shared variable? Justify your answer.

```
entry(pointer to lock) {
   locked = test-and-set-lock(lock);
   while (locked)
     locked = test-and-set-lock(lock)
}

exit(pointer to lock) {
   *lock = 0
}

entry (lock)
access to shared data //critical region
exit(lock)
```

5. Use semaphore to modify the following pseudocode so that operations on the shared variable **var** are atomic. Keep your critical sections as small as possible. And do not change the existing logic sequence.

```
main() {
        int var;
        for (j=0; j <  NumThreads; j++) {
                start new thread executing tfunc(&var);
        }
}

tfunc (int *var) {
        z = 10;
        y = z * (*var);
        print y;
        z = 2 * y;
        *var = *var / 2;
}
```