Kavon Cacho
822794135

Assignment 4 non programming

SINGLE EXAMINEE AFFIDAVIT

"I, the undersigned, promise that this exam submission is my own work. I recognize that should this not be the case; I will be subject to plagiarism penalties as outlined in the course syllabus."

Student Name: Kavon Cacho
RED ID: 822794135
Date: 04/19/2022

Kavon Cacho
822794135

Question 1

1.

Clock hand at a page	Action
0xD	Threshold not met so just clear reference bit, move on
0xE	No action, move on
0xF	No action, move on
0x0	Difference greater than threshold, so update time, set reference bit, and because modified bit is 1, schedule write
0x1	No action, move on
0x2	No action, move on
0x3	Threshold not met so just clear reference bit, move on
0x4	Threshold not met, move on
0x5	Difference greater than threshold, so update time, set reference bit, and because modified bit is 1, schedule write
0x6	No action, move on
0x7	Threshold not met so just clear reference bit, move on
0x8	No action, move on
0x9	No action, move on
0xA	Difference greater than threshold, so update time, set reference bit, and because modified bit is 0, select as victim frame

2. The victim frame is 0x5 because if we follow the algorithm: the reference bit isn't set, so we check if the threshold is being exceeded. It is, and because the modified bit is not set, we select as victim frame instead of scheduling a write.

Kavon Cacho
822794135

Question 2

Time of history update	If a page fault happened during	Page	Referenced during interval before history update	History (4 bit string)	Victim
10ms	10ms – 20ms	0	1	1100	Page 1
		1	0	0100	
		2	1	1000	
		3	1	1100	
20ms	20ms – 30ms	0	0	0110	Page 0/3
		1	1	1010	
		2	1	1100	
		3	0	0110	
30ms	30ms – 40ms	0	0	0011	Page 0/3
		1	0	0101	
		2	1	1110	
		3	0	0011	
40ms	40ms-50ms	0	1	1001	Page 3
		1	0	0010	
		2	0	0111	
		3	0	0001	
50ms	50ms-60ms	0	0	0100	Page 3
		1	0	0001	
		2	0	0011	
		3	0	0000	

Kavon Cacho
822794135

Question 3

```
t1func(...) {

    sem_t* semaphore1;

    if (sem_init(&semaphore1, 0, 2) == -1)
        exit

    start thread t2 executing t2func(t2 args, semaphore1);
    start thread t3 executing t3func(t3 args, semaphore1);

    wait(semaphore1)
    wait(semaphore1)

    do work;

    //t1point
}

t2func(..., sem_t* semaphore1) {

    do_work;

    //t2point
    do_more_work;

    signal(semaphore1);
}

t3func(..., sem_t* semaphore2) {

    do_work;

    //t3point;

    signal(semaphore1);
}
```

Kavon Cacho
822794135

Question 4

No, this pseudocode does not provide a critical region to processes calling them on a shared variable. This is because it does not it does not meet all the conditions. It does not guarantee bounded waiting. It may cause a process to starve for the CPU because it could arrive to execute the critical section only to find it being busy. So it will keep waiting in the while loop with no upper bound on how much it is waiting for.

Question 5

```
main() {
    sem_t* semaphore;

    if (sem_init(&semaphore, 0, 0) == -1)
        exit

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

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