Kevin Andrade
Homework 2

1) Suppose the calls to pthread create() and pthread join() in the follow- ing code always succeed:

(1) 7 ---- Not possible. Since only two threads are created the code will execute the threads twice and even if they overwrite each other the most it would decrease by is 2.
(2) 8 ---- It is possible. Let's say thread 1 is created and finished the the code, Counter is now 9. Now lets say thread 2 is created, takes counter (which is 9) and decreases it to 8.
(3) 9 ---- It is possible. Since threads always have a race condition it could have been that thread 1 and thread 2 loaded their values to their respective register at a similar time. For example: thread 1 retrieves the value of counter into register but now thread 2 starts to execute and also retrieves value into the register, thread 1 is restored and continues its process so now counter is 9 it waits for thread 1 which returns immediately. It now waits for thread 2 which is still executing and the counter of thread 2 is now 9 also since the value that was loaded into the register was also 10. Thread 2 successfully returns and now counter is 9. Both thread 1 & 2 returned 9.
(4) Not possible. The value must be decreased at least once.

2) Three jobs, A, B, and C arrives in a single-CPU system at time 0. They need 20, 30, 20 seconds of CPU time respectively. Answer the following questions ((1) and (2)).

(1) B would finish in 70 seconds. A = 20, C = 20, B = 30. If SJF is implemented then it would be A would take 20, C would take 40 (20 + 20) and finally B would take 70 (20 + 20 + 30).
(2) B would be 30 since B is the largest job.
(3) 58 seconds
(4) 70 seconds
(5) yes. If the time slice is 30 seconds, then b would finish before c has the chance to run. But this scenario is not ideal for round robin. This scenario applies for sequences A,B,C or B,A,C. Another scenario is if c has a lot of IO then b might have a chance to finish before c does since c is constantly waiting (assuming other jobs are not affected by this).

3) A MLFQ scheduler has three queues with priorities 3 to 1 respectively (3 is the highest, 1 is the lowest).

(1)

Time Unit	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22
Priority 3	A	B																				
Priority 2			B	B	A	A																
Priority 1							A	A	B	B	A	A	B	B	A	A	B	B	A	B		

Kevin Andrade
Homework 2

(2)

Time Unit	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22
Priority 3	A					B																
Priority 2		A	A				B	B														
Priority 1				A	A				B	B	A	A	B	B	A	A	B	B	A	B		

4) The lottery scheduler relies on a good random number generator to pick the winner of a lottery when making scheduling decisions. Suppose a lottery scheduler uses a random number generator that generate random numbers in the range of 1 to 1000. Answer the following two questions.

(1) If the random number generator is mostly generating numbers between 1 and 500 then it would mean that any jobs that hold "tickets" higher than 500 will not execute as often as the jobs that hold tickets between 1 – 500 since most jobs will hold tickets between 1 - 500.

(2) (a little unclear on the question so I provide 2 possible scenarios about the question) ….(SCENARIO 1) If 5 even numbers are generated than any jobs that contain tickets outside of those 5 are worthless since the scheduler will only call jobs with the corresponding tickets. (SCENARIO 2) But in the scenario that the random number generator keeps rotating with 5 different even numbers, once it exhausts them all (different sequences composed of 5 distinct even numbers) then it will call jobs that have mostly even tickets in other words if tickets are assigned from 1 – 1000 then roughly 50 percent of those tickets are meaningless since odds will never be called.(APPLIES TO SCENARIO 1 & 2) Also, if the numbers that are being rotated are close to each other (2, 4, 6, 8) then it might be choosing the same process over and over. When numbers are spread out further (200, 400, 600, 800, 1000) then there is a higher chance of different jobs being picked.

5) This question relates to the homework questions 1 to 8 in OSTEP-26

(1) The frequency does not change anything since threads have their own register set and stack.

Kevin Andrade
Homework 2

(2) Since shared variables are located in the code section (which is shared between threads) there is some overwriting happening when the frequency is changed, hence affecting the result.

(3) It does. The critical condition is located when a value is loaded from some address and incremented and stored back. If this is interrupted and overwritten it can cause unwanted results.

```
{ //critical condition example.

mov 2000, %ax       # get the value at the address

add $1, %ax        # increment it

mov %ax, 2000       # store it back

}
```

(4) Value is 200 & I = 3, 6, 9, 12 ……. anything divisible by 3. The reason is due to these not interfering with the 3 initial critical lines of code above (or anything that deals with the shared variables) if we interrupt at the "wrong moment" then we end up overwriting which gives us the wrong result. This is why we interrupt at numbers divisible by three.