8/10

COSC 450 Operating System Mini-Test #2

10/05/2022

Name: Jung An.

1. (1 pt.) A solution for the race condition should have four necessary conditions. Discuss four necessary conditions.

    1) No two processes enter critical region ✓
    2) No process can interrupt other process outside of critical region ✓
    3) No process can wait forever to get into critical region ✓
    4) No assumption can be made about memory or speed of cpu ✓

2. (0.5 pt.) Why many-to-one multi-threading model is not having benefit on multi-processor system? Because many user space have one kernal space, it can still run one process at a time because it only has one kernal space for processing.

    -.3 X

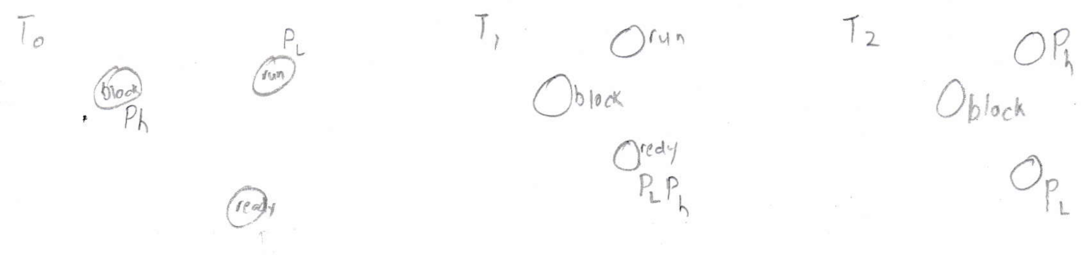
3. (0.5) There are two fundamental models of interprocess communication: shared memory and message queue. To use shared memory, programmer has responsible for synchronization and mutual exclusion of the shared memory. What is main advantage to use the shared memory?

    -.1 Shared memory is faster since it lives in user space compare to message queue which lives in kernal space

    why?

4. (1 pt.) Peterson's solution solves the race condition, but has the defect of requiring busy waiting. Not only does this approach waste CPU time, but also it can also have unexpected effect called the **priority inversion problem**. What is the **priority inversion problem** with busy waiting?

    The problem with priority inversion ✓ is that lower priority gets starved in ready ~~critical region~~ while higher priority cannot get into critical region due to lower priority being in critical region



PL is still in critical region except it is in ready state
Ph is busy waiting to get into critical region

5. (2 pt.)Let's assume there are two processes P_0 and P_1 sharing a resource in the critical section. The following shows a solution for the race condition with busy waiting. Variable **Permit** can be 0 or 1. If **Permit** = 0, only process P_0 can go to the critical section. Once P_0 finish its job in the critical section, P_0 set **Permit** = 1, let process P_1 enter critical section.
If **Permit** = 1, only process P_1 can go to the critical section. Once P_1 finish its job in the critical section, P_1 set **Permit** = 0, let process P_0 enter critical section. (we assume that when a process enter critical section, a process never terminated inside critical section)

```
repeat
        while Permit ≠ i do
            ; (no-operation)
         | Critical Section |

        Permit = j;

          | Remainder Section |
until false
```

-1.5

Show this solution cannot solve race condition.

This cannot solve race condition in case either P_0 or P_1 times out after reading permit

ex.
T_0 P_0 reads permit = 0 and goes into critical section

T_1
P_1 reads permit = 0 and times out.

T_2 P_0 finishes and set permit = 1.

Not true!!

P_1 is scheduled and still has permit $\cancel{=}$0 and waits.

??

P_0 is scheduled and permit = 1 and waits.

So both process is not waiting forever to get into critical region

6. (2 pt.) Mr. Computer tries to solve the race condition with semaphores in the Producer-Consumer problem. He comes up with following solutions.

   Does his solution solve the race condition? Discuss Mr. Computer's solution, if his method works for avoiding race condition. If his method does not work, write a scenario lead to the situation that violates the race condition.

```
#define N 100
semaphore mutex = 1;
semaphore empty = N;
semaphore full = 0;

void producer ()
{
    int item;
    while (ture)
    {
        item = produce_item();
        down (&empty);
        down (&mutex);
        insert_item(item);
        up(&mutex);
        up(&full);
    }
}
```

```
void consumer()
{
    int item;

    while (true)
    {
        down(&mutex);
        down(&full);
        item = remove_item();
        up(&mutex);
        up(&empty);
        consume_item(item);
    }
}
```

No it does. not

On consumer side mutex is down first so unless full != 0 it will wait forever in consumer since mutex is locked ✓

So it violates that process is waiting forever to enter critical region.

example

- Consumer is called first by Scheduler
- mutex becomes down ✓
- full is 0 so it waits until full != 0

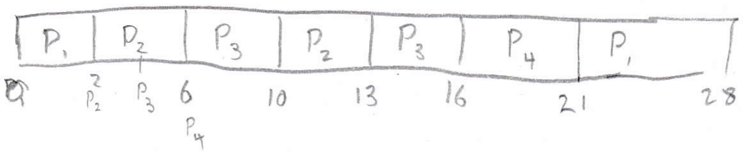
- Producer is called.
- Cannot do down mutex since it is down so it waits until mutex becomes up.

7. (2 pt.) Consider the following set of processes (each process is 100 % CPU-bounded).

Process	CPU-Time	Arrival Time	Priority
P_1	8	0	1
P_2	7	2	3
P_3	7	4	3
P_4	5	6	2

What is the average process waiting times and average turnaround time for the preemptive priority scheduling algorithms? There are rules for some cases

- Between processes with same priorities, use round robin with time unit 4.
- If preempted process has highest priority and does not use its time unit, it will keep CPU time.



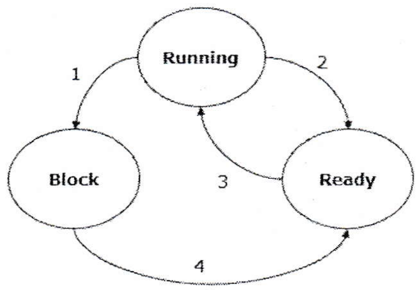
awaiting $(21-2)+(10-6)+((13-10)+(6-4))+(16-6)\ /4$

$19 + 4 + 5 + 10\ /4 = 9.25$

Turnarown $(28)+(13-2)+(16-4)+(21-6)$

$28 + 11 + 12 + 15\ /4 = 16.5$

8. (1 pt.) A process is stayed in one of three states: running, blocked or ready state. Briefly discuss each of states and transaction between states.



- Running state – Process is using cpu
- Ready state – Process is ready for short time scheduler
- Blocked state – Process is waiting of resource (input)
- Transaction 1 – Process require input and is put to block state
- Transaction 2 – Process timed out
- Transaction 3 – short time scheduler scheduled pIocess to CPU
- Transaction 4 – Process got it's required input (resource) is is ready to be called by scheduler