Department of Computer Science & Engineering
Indian Institute of Technology Kharagpur
Mid-semester Examination, Spring 2016
CS30002: Operating Systems

Full Marks: 40 **Time: 2 Hours**

Instructions: Answer all the questions. Answers must be brief and to the point. Take and state suitable assumptions, if necessary

1.(a) Consider the following solution for the critical section problem.
The two processes, P_0 and P_1, share the following variables

```
boolean blocked[2] /* initialized to false */
int turn:    /* initialized to 0 or 1*/
```

The structure of the Process i (i=0 or 1) is as follows

```
do{
      blocked[i]=true:
      while (turn!=i)
      {
          while(blocked[1-i]) :  /* busy wait */

          turn=i ;
      }

Critical section ():

      blocked[i]=false:

Non-Critical section():
} while(1):
```

"The above solution satisfies all the necessary requirements of the critical section problem". Prove or disprove with proper justification.

(b) Consider the following program

```
P1: {
Shared int x
x=10;
while(1)
{
        x=x-1;
        x=x+1;
        If(x!=10)
                printf("x is %d",x);
}
}
```

```
P2: {
Shared int x
x=10;
while(1)
{
        x=x-1;
        x=x+1;
        If(x!=10)
                printf("x is %d",x);
}
}
```

Note that the scheduler in a uniprocessor system would implement pseudo parallel execution of these two concurrent processes P1 and P2 by interleaving their instructions.
Show a sequence of execution such that statement "x=10" is printed. You should remember that the increment and decrement are not done atomically.
(c) Determining the time quantum q is a complex task for the system developer. Assume that the average context switch time between processes is s, and the average amount of time a I/O bound process executes before generating an I/O request is t (t >> s). Discuss the effect on the response time of each of the following quantum settings (i) s<q<t (ii) q=t (iii) q>>t (iv) q=s.

(d) Consider a system which implements preemptive priority-based scheduler and Peterson's solution for mutual exclusion. In such a system, is it possible that a high priority process gets delayed indefinitely because of the presence of lower priority processes? Assume that no process does any I/O and there is no deadlock in the system.

[3+2+3+2]

2.(a) Consider a Linux kernel implementing O(1) scheduler. The system supports classical 140 priority levels (0-139).
(i) Explain the role of the bitmap vector in the active list to schedule a new process.
(ii) The expired list doesn't play any role in the scheduling activity. Then why does the kernel keeps bitmap in the expired list too.
(iii) What is the role of (a) **lock** field in the run queue (b) **sleep_avg** in Task_Struct
(iv) Assume that the O(1) scheduler implements the following policy to calculate Bonus

Average **sleep avg.**	Bonus
0 to 100 ms	0
>100 ms & ≤ 500 ms	5
>500 ms	10

Show that, a lowest priority user process would not get any chance to become an **Interactive** process, irrespective of its I/O activities.
(b) Explain how kernel data structures are protected from being modified by the user process. Can user process access the kernel data structures?

[2+2+2+2+2]

3. (a) All the scheduling algorithms discussed in the class treat the ready queue as a single pool of processes, from which to schedule the next running process. Consider **Fair-Share scheduling (FSS)**, a variation of Unix scheduling strategy, which is designed to enforce fairness to a group of (user) processes. In FSS scheduling, the concern is not how a particular process performs, but rather, how a set of processes, coming from a group of users, perform. So the scheduling decision will be made based on the processes coming from a group of users. Scheduling is done based on the priority, which takes into account the process (category) priority, recent processor usage and the processor usage of the group , in which the process belongs to. The system divides the user processes into a set of groups and allocates a fraction of processor time to each group.
The priority of process j in group k gets calculated in the following way.

$$CPU_j(i) = \frac{CPU_j(i-1)}{2}$$

$$GCPU_k(i) = \frac{GCPU_k(i-1)}{2}$$

$$P_j(i) = Base_j + \frac{CPU_j(i)}{2} + \frac{GCPU_k(i)}{4 \times W_K} + nice_j$$

where $CPU_j(i)$ = measure of CPU utilization by process j through time interval i
$GCPU_k(i)$=measure of CPU utilization of group k through time interval i
$P_j(i)$=priority of process j at the beginning of time interval i; lower value indicates higher priority
W_k=Processor fraction allocated to group k, where $0 < W_k \leq 1$ and $\sum_k W_k = 1$

Now consider that the system has two groups; process A is in first group and the processes B and C are in the second group, with each group getting the processor share of 0.5 (W_K). Assume all the processes are CPU bound and they are always ready to run (starts with Process A). Once per second, the priorities are recalculated and scheduler selects a process for execution. The CPU utilization count of the running process gets incremented 60 times per second; as for the CPU utilization count of the *corresponding group*. Consider that all the processes have same base priority 60 and nice value 0. Illustrate the execution of these three processes, including the priority and scheduling decision after every second and show the process CPU utilization and group CPU utilization count. Show all the information at time t=0, 1, 2, 3, 4, 5 (in second).

(b) "Disabling Interrupt is not an elegant solution to avoid race condition", justify with two major reasons.

(c) In round robin scheduling, timer generates interrupt once the time quantum expirer. Outline the interrupt service routine of the timer interrupt.

[6+2+2]

4.(a) In the class, we introduced the busy waiting based implementation of the Producer Consumer problem. Now let us assume that your system provides two synchronization primitives namely block() and wakeup(P). A process X invoking block() system call gets switched from running to waiting state. Another process Y invoking wakeup(X) system call switches the process X from waiting to ready state. Propose a non-busy waiting based implementation of the bounded buffer Producer Consumer problem. Your implementation need not guarantee the race condition free solution.

(b) Demonstrate the race condition in your above implementation, if it exists. (if your solution is free from race condition, prove it).

(c) State Strict Alternation solution to solve the critical section problem. Justify, if the solution satisfies (i) Progress (ii) Bounded waiting.

(d) Consider a processor does not have a TSL (Test and Set lock) instruction, but it supports a system call "**COMP_N_SWAP(MEM, TESTVAL, NEWVAL)**". This system call checks the content of memory location MEM against a TESTVAL. If the memory location's current value is TESTVAL, it is replaced with NEWVAL; otherwise it (MEM) is left unchanged. The old memory value is always returned. This system call allows the executing processor to exclusively allocate the memory bus, so that other processors cannot access the memory ; all the aforementioned steps are performed as a single indivisible (atomic) action. Show how you can use COMP_N_SWAP to implement mutual exclusion. Propose the ***entry_section*** and ***exit_section*** code modules.

[2.5+1.5+3+3]