2017UCP1003

Malaviya National Institute of Technology Jaipur

Time: 1.5 hrs **MTE (2019-20s) Operating System** **Max. Marks: 40**

1 (a) Explain "priority inversion" in context of CPU scheduling.
(b) Contrast any two methods to resolve priority inversion.
(c) Discuss significance of `virtual runtime (vruntime)`, `sched_latency`, `min_granularity` and `weight` on performance of a CFS (Completely Fair Scheduler).
(d) Contrast deterministic stride versus probabilistic lottery scheduling. **[3, 4, 4, 3 = 14]**

2 (a) "A cycle in resource allocation graph (each resource has exactly one instance) can be used to detect a deadlock". How is this statement modified in case of multiple resource instances?
(b) A system has four processes and five allocatable resources A, B, C, D, E. The current allocation and maximum needs are shown below. Available means the resources available after all allocations. What is the smallest value of x for which this is a safe state?

	Allocated	Maximum	Available
	A B C D E	A B C D E	A B C D E
Process 1	1 0 2 1 1	1 1 2 1 2	0 0 x 1 1
Process 2	2 0 1 1 0	2 2 2 1 0	
Process 3	1 1 0 1 0	2 1 5 1 0	
Process 4	1 1 1 1 0	1 1 2 2 1	

[3+3 = 6]

3 (a) Three kinds of threads share access to a singly-linked list: `searchers`, `inserters` and `deleters`. It is known that `Searchers` merely examine the list; hence they can execute concurrently with each other. `Inserters` add new items to the end of the list; insertions must be mutually exclusive to preclude two `inserters` from inserting new items at about the same time. However, one insert can proceed in parallel with any number of searches. Finally, `deleters` remove items from anywhere in the list. At most one `deleter` process can access the list at a time, and deletion must also be mutually exclusive with searches and insertions. Write pseudocode using semaphores to provide a solution to this problem.
(b) Explain how a counting semaphore can be implemented using only binary semaphore(s).
(c) Explain implementation of `acquireLock()` and `releaseLock()` using a `CompareAndSwap(M, old, new)` instruction, which compares the contents of a memory location `M` with value `old` and, only if they are the same, modifies the contents of that memory location to the value `new`. **[5,5,2 = 12]**

4. Explain working of Eisenberg and McGuires solution for N-process Critical Section Problem.

```
enum pstate = { IDLE, WAITING, ACTIVE };
pstate flags[N];
int turn = random integer from [0, 1, ..., N-1]; // N is number of processes
repeat {
    flags[i] := WAITING;
    index := turn;
    while (index != i) {
        if (flags[index] != IDLE) index := turn;
        else index := (index+1) mod N;
    }
    flags[i] := ACTIVE; index := 0;
    while ((index < N) && ((index == i) || (flags[index] != ACTIVE))) {
        index := index+1;
    }
} until ((index >= N) && ((turn == i) || (flags[turn] == IDLE)));
turn := i;
/* Critical Section Code of the Process */
index := (turn+1) mod N;
while (flags[index] = IDLE) {
    index := (index+1) mod N;
}
turn := index;
flags[i] := IDLE;
/* REMAINDER Section */
```

[8]