OPERATING SYSTEMS (COMP 310/ECSE 427) – Alternate Midterm – March 12, 2013

Name: **Student ID:**

Provide **brief answers to all** questions. This exam is worth **100 points** in total.

1. This question has multiple parts **[20 points]**
   a. What are the major issues faced when developing a scheduler for symmetric multiprocessors? Briefly explain how the O(1) scheduler addresses these issues.

There are two major issues: contention for the task queue and cache footprint management. The O(1) scheduler addresses these issues by having a local task queue per processor. This allows concurrent access and also the tasks remain local to the processors as much as possible.

   b. Give a memory configuration and sequence of memory requests for which the best-fit would outperform first-fit.

Consider a memory configuration with holes in decreasing sizes. Suppose both algorithms are scanning from address 0 and we have the following holes (starting address, size): (1000, 500), (2000, 300), (3000, 100). We get the following requests for memory allocation: 300 and 400 (in that order). Best fit would put the first request in the hole starting at 2000 and the second one at the address starting at 1000. First fit would put the first request at the hole starting at 1000. At this point the largest hole would be of size 300 – the one starting at 2000. The second request cannot be allocated.

   c. Buddy algorithm is used to manage dynamic memory allocation. The memory region allocated by the Buddy algorithm has a maximum capacity of 64 Kbytes. Suppose the memory region starts at 0x800000. Find all possible starting addresses for the block allocated by Buddy for a request of size 5 Kbytes. What is the internal fragmentation of an allocation performed by Buddy?

Starting addresses are 0x800000, 0x802000, 0x804000, 0x806000, 0x808000, 0x80A000, 0x80C000, 0x80E000. The internal fragmentation is 3K for an allocation performed by the buddy.

   d. What is the maximum internal fragmentation you can get with the Buddy in the above situation?

The largest block that could be allocated by the buddy is 64Kbytes. So, the maximum internal fragmentation is 32767 bytes.

2. Deadlock problems [**25 points**]
   a. State the necessary and sufficient conditions for a deadlock to happen.

Circular wait.

Hold and wait.

No preemption.

Mutual exclusion.

   b. A system has four processes and five allocatable resources. The current allocation (or the "Hold" matrix) is given by [1 0 2 1 1; 2 0 1 1 0; 1 1 0 1 0; 1 1 1 1 0]. The claim matrix (or "Max" matrix) is given by [1 1 2 1 3; 2 2 2 1 0; 2 1 3 1 0; 1 1 2 2 1]. The available vector is given by [0 0 x 1 1]. What is the smallest value of x for which this is a safe state?

The available vector was corrected to [0 0 x 1 2] in the exam hall. The smallest x is 1.

   c. A computer has six tape drives, with *n* processes competing for them. Each process may need two drives. For which values of n is the system deadlock free?

The only way to have a deadlock is to have 6 processes and each one hold a tape. Suppose we have just 5 processes. At least one process should get 2 tapes. So it can complete and then two more processes can get 2 tapes each. So they can complete too. Therefore, the system will be deadlock free for n < 6.

   d. Briefly discuss at least two ways of implementing deadlock prevention.

First way is by ordering the resources in an arbitrary order. When multiple resources need to be acquired, we should acquire them in the predefined order. Because all processes are respecting the same order, circular wait is prevented. So we have prevented deadlocks.
Second way is by allocating all resources in one single allocation. If a resource in the group is not available, then none of the resources are allocated. This invalidates the hold-and-wait condition. So the deadlock is prevented.

3. Synchronization problems [**30 points**]
   a. Implement a barrier synchronization mechanism using semaphores. Your barrier should be reusable (can be called multiple times). You need to use minimum number of semaphores.

```
Semaphore bwait = 0
Semaphore mutex = 1
int pcount = N

wait(mutex)
        N = N -1
        if (N > 0) {
                signal(mutex)
                wait(bwait)
        } else
                signal(bwait)
        N = N + 1
signal(mutex)
```

b. Implement a monitor using semaphores. In this monitor, signal() wakes up all the threads that are waiting on the condition variable on which the signal was invoked. Also, the signaling thread does not wait. Instead, the thread(s) woken up by the signal wait until the signaling thread leaves the monitor. Although signal() wakes up all threads, they are prevented from being active within the monitor at the same time – which would violate the condition of the monitor.

```
Semaphore  mutex = 1
Semaphore next = 0
Semaphore xsem = 0
int xcount
int nextcount

wait(mutex)
<body of a function>
if (nextcount > 0)
        signal(next)
else
        signal(mutex)

cwait(x)
        xcount++
        if (nextcount > 0)
                signal(next)
        else
                signal(mutex)
        wait(xsem)
        xcount—
        nextcount++
        wait(next)
        nextcount—
```

```
csignal(x)
    for I from 0 to xcount -1
        signal(xsem)
```

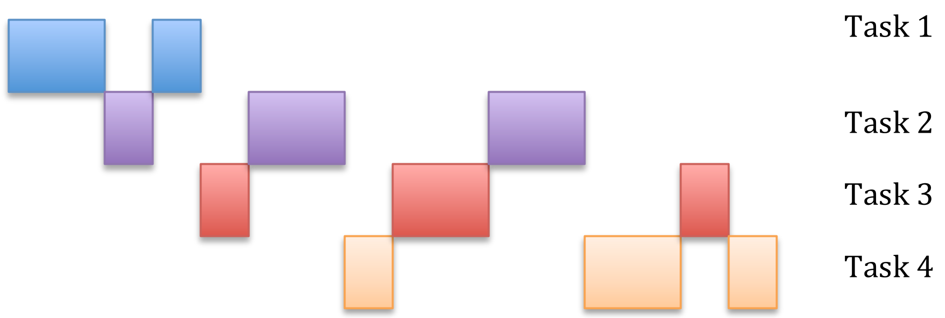
4. Scheduling problems [**25 points**]
   a. Provide a sequence of 10 tasks (specify their service times and arrival times) that would give the same average completion times for shortest job first and first-come first served. If you cannot find such a task sequence explain why it is not possible.

Any task sequence where tasks come in **non decreasing** sizes would give the same average completion times for both SJF and FCFS. For example, (0, 10), (5, 10), (10, 15), (15, 20), (15, 20), (20, 30), (20, 30), (25, 35), (30, 40), (50, 60) would give the same average completion times for both.

   b. Rate monotonic scheduler is asked to schedule a task with deadline 50 and service time 30. Give a second task it cannot schedule but an earliest deadline scheduler can schedule.

Consider another task with a deadline of 75 and service time of 25. This task has lower priority than the first task. The RMA will schedule the first task every 50 time units. In the first 75 units, there will be only 20 free time units. Therefore, RMA cannot schedule the second task. The earliest deadline first scheduler can schedule both tasks. In the first 150 time units, we have 30 + 25 + 30 + 25 + 30 = 140 time units. Therefore, EDF can find a feasible schedule.

   c. Schedule the following tasks using a multi-level feedback queue scheduler. There are four levels in the scheduler and the maximum quantum increases from 1 (at the top) to 8 at the bottom. Task 1 (arrival 0, service 3), Task 2 (arrival 2, service 5), Task 3 (arrival 4, service 4), Task 4 (arrival 6, service 4)


Task 1
Task 2
Task 3
Task 4