Prof. Rick Han, University of Colorado at Boulder

Midterm Exam
CSCI 3753, Operating Systems
Spring 2015

Name: ____SOLUTIONS____________

1. Enter your name above
2. You should have 5 pages with 3 problems
3. You should partially answer a question rather than leave it blank. Show the details of your work and state assumptions to receive full credit, but keep your answers to the point.
4. Feel free to use the back of pages. Just indicate when you've done so.
5. The Honor Code applies to this exam.

Problem 1 (50 points): Synchronization and Deadlock

(a) [10 points] Answer the following questions True or False. An incorrect answer will cancel a correct answer. You may leave a question blank. The lowest possible total score is zero. (+2 for each correct, but incorrect answers cancel correct answers)

__T____ Mutual exclusion is necessary to prevent race conditions.
__F____ A spinlock implementation of locks uses the test-and-set atomic instruction.
__F____ Signaling with a semaphore is equivalent to signaling with a condition variable.
__T____ Starvation is still possible in the monitor-based solution to the Dining Philosophers problem.
__T____ If the Banker's algorithm finds the system is in a safe state, then deadlock detection is unnecessary.

(b) [6 points] To emulate semaphore signaling, the following pseudocode implementation adds state to a condition variable. Does this approach enforce ordering, i.e. does C1 always execute first followed by C2 executing? Justify your answer.

```
/* variables shared between 2 threads */
Lock mutex;
Condition CV;
int state=0;
```

Thread T1:

```
C1;  // code C1
lock(mutex);
state=1;
unlock(mutex);

signal(CV);
```

Thread T2:

```
lock(mutex)
while(!state) {
  unlock(mutex);
  wait(CV);
  lock(mutex);
}
unlock(mutex);
C2;   // code C2
```

This does NOT enforce ordering. T2 can be interrupted in while() loop just after unlock(mutex) & before wait(CV). T1 then grabs lock, sets state, & signals CV. Then T2 calls wait(CV), waiting forever. C2 never gets executed.

(c) [8 points] Which of the following could result in a race condition? Check the ones that apply. An incorrect answer will cancel a correct answer. Lowest possible score is 0.
__X__ two threads modifying the same global variable
____ two readers reading from the same shared buffer
__X__ two processes modifying the same location in a block of shared memory
____ two philosophers grabbing the same semaphore for a chopstick
__X__ a signal handler interrupts a program's main control flow to alter a global variable
____ a child process modifies a global variable copied from its parent
(3 correct = +8, 2 correct = +6, 1 correct = +3)

(d) [8 points] List two approaches to deadlock prevention that break the hold-and-wait condition. List two approaches to deadlock prevention that break the no-preemption condition.
(+4) Hold-and-wait (any 2 of 3): 1) hold-all at process creation and release when process exits, 2) hold-all then release-all in steps during execution, 3) hold-one then release-one in steps during execution.

(+4) No-preemption: 1) requestor releases a held resource, 2) requestee releases a held resource but only if requestee is waiting on another resource.

(e) [8 points] Is the following system (processes P1, P2, P3 and resources R1, R2, R3) in deadlock? If no, prove it. If yes, which processes are in deadlock?

Alloc
$$\begin{array}{c|ccc} & R1 & R2 & R3 \\ \hline P1 & 0 & 1 & 0 \\ P2 & 2 & 1 & 0 \\ P3 & 1 & 0 & 0 \end{array}$$

Request
$$\begin{array}{c|ccc} & R1 & R2 & R3 \\ \hline P1 & 2 & 0 & 1 \\ P2 & 1 & 1 & 0 \\ P3 & 0 & 0 & 1 \end{array}$$

Max
$$\begin{array}{c|ccc} & R1 & R2 & R3 \\ \hline P1 & 3 & 5 & 1 \\ P2 & 3 & 2 & 2 \\ P3 & 1 & 0 & 1 \end{array}$$

Avail
$$\begin{array}{ccc} R1 & R2 & R3 \\ 2 & 0 & 1 \end{array}$$

Only need to run deadlock detection algorithm, not the Banker's deadlock avoidance algorithm. Deadlock detection only looks at Alloc, Request, and Avail, and ignores Max (and Need).

Step 0: Work = Avail = [2 0 1]
Step 1: Find Request(i) <= Work. Request(3) = [0 0 1] <= Work (Alternatively can pick P1 at this step)
Work += Alloc(3) = [1 0 0]. Work = [3 0 1] (if picked P1, then Work += Alloc(1) = [0 1 0], so Work = [2 1 1]. At this point, every process' request can be clearly satisfied)
Step 2: Request(1) = [2 0 1] <= Work.
Work += Alloc(1) = [0 1 0]. Work = [3 1 1]
Step 3: Request(2) = [1 1 0] <= Work.

All processes satisfied. There is no deadlock.
Partial credit if only ran the Banker's deadlock avoidance algorithm. Should find the system to be in an unsafe state (P1 & P2's max needs can't be met).

(f) [10 points] The following code is a modified version of the solution to synchronizing the Hawaii one-lane bridge problem from our problem set #2. Recall only one direction at a time of cars can cross the bridge. Is this solution starvation-free to westbound and/or eastbound cars? Justify your answer. If there are already other cars on the bridge going in the same direction, can a newly arrived car always join them? Justify your answer.

```
int westcount=0, eastcount=0;
Semaphore bridgemutex=1, mutex1=1, mutex2=1, block=1;
```

Westbound cars:

```
P(block)
P(mutex1)
westcount++;
if (westcount==1) P(bridgemutex);
V(mutex1)
V(block)

// Cross bridge westbound

P(mutex1)
westcount--;
if (westcount==0) V(bridgemutex);
V(mutex1)
```

Eastbound cars:

```
P(block)
P(mutex2)
eastcount++;
if (eastcount==1) P(bridgemutex)
V(mutex2)
V(block)

// Cross bridge eastbound

P(mutex2)
eastcount--;
if (eastcount==0) V(bridgemutex)
V(mutex2)
```

(+7) The solution is starvation-free for both westbound and eastbound cars. The block semaphore ensures it is starvation-free. If there are multiple cars in one direction on the bridge, and a new car C arrives in the opposite direction, it grabs block. Any cars that arrive after that in any direction are queued up on block. Eventually, all the current cars leave the bridge, waking car C, who travels next on the bridge. If there are a string of cars after C in the same direction as C, they all get woken up from block. Multiple such cars can now cross. The first car in the opposite direction is also eventually woken from block. Hence, there is no starvation in any direction.

(+3) If there are already cars on the bridge going in the same direction, a newly arrived car CANNOT always join them if there is an already waiting car in the opposite direction. The newly arrived car will block on the block semaphore, but the already waiting car is blocked even earlier on the block semaphore, and will be served first.

Since the latter question should have stated "If there are already other cars on the bridge going in the same direction *as a newly arrived car*, …", then credit was also given if it was assumed that "in the same direction" only applied to the cars already on the bridge, but not necessarily to the newly arrived car (it may be going in the opposite direction). In this case, the newly arrived car would be blocked.

Problem 2 (32 points): Processes, Threads, and Scheduling

(a) [10 points] Answer the following questions True or False. An incorrect answer will cancel a correct answer. You may leave a question blank. The lowest possible total score is zero. (+2 for each correct, but incorrect answers cancel correct answers)

___F___ Threads share a call stack with other threads in the same process.
___F___ On a fork(), copy-on-write creates a duplicate copy of all code & data pages in the child process.
___T___ Shared memory IPC is faster than message passing for large data transfers.
___T___ In multi-core scheduling, hard affinity exploits caching but can imbalance load.
___T___ In rate-monotonic scheduling, the task's priority is inversely related to its period.

(b) [6 points] Is the function f() thread-safe? Is f() reentrant? Justify your answers.

```
Lock mutex;
int g=0;
int f(int i) {
    int x = g;
    x = x - 2;
    x = i;
    x = x*x;
    return x;
}
```

f () is thread-safe and reentrant. The statement x=i essentially makes int x=g and x=x-2 irrelevant. Hence, the returned value of x is essentially i^2, where i is a parameter and x is a local variable. A 2nd thread would have its own parameter and local variable. A thread reentering f() would pass in another local parameter and use another own local variable, not conflicting with the main control flow's parameter and local variable.

(c) [4 points] Consider three processes P1,P2 and P3 as follows:

Process	Priority	Entering Time	Running Time
P1	1	T	4
P2	3	T	2
P3	2	T+3	1

What is the average turnaround time with an SJF (Shortest Job First) scheduler? What is the average response time with a Priority scheduler? (priority 1 > priority 2 > …) Assume preemption in both cases.

SJF schedules the tasks: Let T=0. P2 gets time [0,2], P1 gets [2,3], P3 get [3,4], P1 gets [4,7]

Average turnaround time = (7+2+1)/3 = 10/3

Priority schedules the tasks: P1 gets time [0,4], P3 gets [4,5], P2 gets [5,7]

Average response time = (0 + 1 + 5)/3 = 2

(d) [12 points] The Linux CFS scheduler selects the task with the maximum wait time.

- Why is this equivalent to selecting the task with the minimum virtual run time? The task with the longest wait time is intuitively the task that has had the least time executing on the CPU
- Why does CFS add min_vruntime to newly activated tasks? This re-normalizes or catches up the newly activated task to the current level of run times, but places it at the start of the queue. Otherwise, if the task had been asleep a long time and was placed with its actual run time, it would take a long time for the system to increment the run time until it was commensurate with the other tasks, effectively blocking all other tasks from running for a long time.
- How are interactive tasks given preference by CFS? Interactive tasks are newly active, hence are given min_vruntime and are put at the head of the queue.
- Name one reason why CFS uses a Red-Black tree. By organizing all runtimes of tasks in a RB tree, the system can quickly find the minimum task to schedule next while reinsertion to the tree is relatively fast too (O(logN))
- In what two ways does CFS integrate priorities? CFS increases the time slice given to higher priority tasks and decreases the amount of virtual run time added to each high priority task.

Problem 3 (18 points): I/O, General OS

(a) [8 points] Answer the following questions True or False. An incorrect answer will cancel a correct answer. You may leave a question blank. The lowest possible total score is zero. (+2 for each correct, but incorrect answers cancel correct answers)

__T____ When a context switch is triggered by a system call, and the system traps to the OS, the mode bit is set to kernel mode.

__F____ Early cooperative multitasking systems were favored over multiprogrammed batch systems because they provided good fault isolation.

__T____ One advantage of memory-mapped I/O over port-mapped I/O is that no special hardware instructions are needed for I/O.

__T____ In PC bootstrapping, the BIOS searches for a hard drive to boot from to load the primitive bootloader.

(b) [4 points] Describe two ways that DMA with interrupts is more efficient than polling I/O.

Polling wastes cycles periodically checking a device if it's ready. (+2) Interrupts efficiently inform the CPU once the device is ready, allowing the CPU to proceed with computation while I/O is overlapped with the computation. (+2) DMA with interrupts is even better by issuing only one interrupt at the end of a long data transfer, rather than interrupting the CPU for every small amount of data transferred.

(c) [6 points] What is a microkernel and how does it operate? Name one advantage and one disadvantage of microkernels compared to monolithic kernels.

(+3) A microkernel is an approach to OS design in which the kernel contains only the minimal components. These are usually the scheduler, virtual memory, and message passing components. All other components such as the file system are viewed as external to the OS.

(+3) One advantage is the fault of a component doesn't necessarily bring down the whole OS, unlike a monolithic kernel. One disadvantage is message passing overhead makes microkernels slow.

Problem 1: ________	/ 50
Problem 2: ________	/ 32
Problem 3: ________	/ 18
Overall Score: ________	/ 100