Midterm Exam
CSCI 3753
Operating Systems
Spring 2012

Name: _____

Suggested Solutions

Credit also given for solutions not necessarily shown here.

Instructions:

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 (44 points): Synchronization and Deadlock

(a) [10 points] Using locks instead of semaphores, devise a solution that enforces ordering, i.e. code C1 in process P1 is guaranteed to execute before code C2 in process P2.

```
Boolean flag=FALSE;  // global flag
Lock lock=unlocked;  // global lock
```

P1:	P2:
C1;	Acquire(lock);
// signal P2	// wait to be signaled by P1
Acquire(lock);	While(flag==FALSE){
flag = TRUE;	Release(lock);
Release(lock);	Acquire(lock);
	}
	C2;

Also acceptable solution: Lock lock=locked; // assuming lock can be initialized like this

P1:	P2:
C1;	Acquire(lock);
// signal P2	// wait to be signaled by P1
Release(lock);	C2;

(b) [15 points] Suppose there is a one-lane east-west tunnel. Cars moving west have priority, so as long as there is at least one westward-moving car, then eastward-moving cars have to wait. There can be only one eastward-moving car in the tunnel at a time, but more than one westward-moving car at a time in the tunnel. Even though westward cars have priority, if there is already an eastward-moving car in the tunnel, then a newly arrived westward-moving car must wait until the eastward-moving car has exited the tunnel. Design a synchronization solution for this tunnel using only semaphores and locks (integers and Booleans are OK too). You should design separate code for west-moving and east-moving cars, and declare any shared global variables. Hint which of the classic synchronization problems does this tunnel most resemble?

+8 for mostly right (e.g. +8 for single lock solution)
This is basically the first readers/writers problem,
with west-moving cars = Readers, and east-moving cars = Writers.

```
// West-moving cars share data structures:
semaphore mutex, east   // initialized to 1
int westcount;          // initialized to 0, controlled by mutex
// East-moving cars also share semaphore wrt
```

Eastward cars:

```
wait(east);
// drive into tunnel
signal(east);
```

Westward cars:

```
wait(mutex);
westcount++;
if (westcount==1) wait(east);
signal(mutex);

// drive into tunnel

wait(mutex);
westcount--;
if (westcount==0) signal(east);
signal(mutex);
```

could also add an eastQ semaphore initialized to 1, called by wait(eastQ) just before wait(east) for eastward cars (and signal(eastQ) after signal(east) for eastward cars). This would be a refinement of 1st R/W that allows a new west car to wait at most one east car before grabbing the tunnel for all west cars. Otherwise, without eastQ, new west cars would have to wait on the east semaphore until it is emptied of all east cars that had been pending. It's OK to leave out eastQ for this problem.

(c) [10 points] What are the four necessary and sufficient conditions for deadlock? Which condition in the monitor-based solution to Dining Philosophers was broken to prevent deadlock?

(+2) each
Mutual exclusion – atomic locks/semaphores
Hold and wait – hold one resource while waiting for another
No preemption – once a resource is held, only that process can release it. No other entity (process or OS) can force the original process to release the resource
Circular Wait – the sequence of holds-and-waits forms a circle. P(i) waits on P(i+1%N)
(+2) DP solution broke either hold and wait or circular wait.

(d) [9 points] If Dijkstra's Banker's Algorithm fails to find a safe sequence, is the system deadlocked? Justify your answer. Can there be more than one safe sequence? Why or why not? In step 3 of both the Banker's Algorithm and the Deadlock Detection Algorithm, Work = Work + Alloc(i). Explain the intuition behind this step.

(+3) Such a system would be unsafe, but not necessarily deadlocked. The definition of unsafe depends on worst-case or maximum resource analysis, which is a conservative indicator warning you that trouble (deadlock) may be ahead, but is not necessarily deadlocked right now. Can draw a diagram showing set of deadlocked states is a subset of set of unsafe states
(+3) Yes, there can be more than one safe sequence. See example from lecture notes.
(+3) The algorithm is emulating that each selected process gives up its allocated resources at each step to see if the remaining resources are enough to satisfy one more process' needs, iterating to find a sequence of releases that keeps the system out of trouble.

Problem 2 (44 points): Processes, Threads, and Scheduling

(a) [12 points] Assume that you have a preemptively multitasked system, in which the scheduler is invoked on each time slice to choose the next process to run. Which of the following scheduling policies are preemptive and which are not FCFS, SJF, EDF, RR, multi-level feedback queue? Justify your answers. Which one minimizes the average wait time and which one minimizes the response time?
(+2 each)
Preemptive: SJF (a new process shorter than all other processes can jump ahead or preempt all longer processes). Gave +1 if non-preemptive and said SJF ran to completion.
Preemptive: EDF (a new process with earlier deadline than all existing processes can jump ahead or preempt all existing processes). Gave +1 if non-preemptive and said EDF ran to completion.
Preemptive: MLFQ (a new process with a higher priority than all existing processes can preempt all existing processes)
Non-preemptive: FCFS (new processes can't jump ahead of prior processes because scheduling is based on the order of arrival time stamp)
Non-preemptive: RR (new process just waits its turn in the rotation). Also OK to list RR as preemptive because of time slice interrupts.
(+1) SJF minimizes average wait time
(+1) RR minimizes average response time, but since the question didn't ask for average, then also accepted EDF as minimizing interactive response time with real time response. Also accepted MLFQ, since it biases towards interactive processes in some cases.

(b) [8 points] Describe the evolution of Linux scheduling algorithms, except multi-core (see next problem). What was the motivation at each step of development?
(+2) each, any four in sequence with motivation
Round Robin – simple to implement, fair, fast response time
MLFQ – support priorities to give real-time processes priority over non-RT processes, also process movement/aging possible
O(N) – give interactive processes more priority
O(1) – give interactive processes more priority, but in constant time, O(N) too slow/scales poorly
CFQ – red-black tree, O(log N), because too many heuristics in O(1)

(c) [8 points] Describe two types of SMP (symmetric multi-processing) multi-core scheduling. How do caching and load balancing affect these types of SMP multi-core schedulers?
(+4) The two types of SMP multi-core scheduling are:

- there is a global ready queue shared between all processes, and each process has its own scheduler, i.e. is self-scheduled.
- Each processor has its own ready queue and own scheduler, and again is self-scheduled. This is what most modern OS's implement. There may be process migration between processors.

(+2) Caching affects process execution performance. Each CPU has its own cache. If processes move too much between processors, then the CPU has to keep rebuilding its cache. Therefore, for both forms of scheduling, the system tries to keep processes executing on the same core. This is called process affinity. Hard affinity specifies that processes are immobile, fixed on a given core. Soft affinity allows limited movement. Maybe more process movement with a single ready Q, causing cache rebuilding.
(+2) Load balancing is implicit in the architecture with the global ready queue , since as soon as a processor is done executing a process, it seeks another one from the global queue, spreading the load. In the 2nd architecture, a given processor may be overloaded with too many processes that are CPU-bound. In this case, some processes may be forced to migrate to other processors to balance the load. This would then cause some caching issues, but the hope is that overall performance would eventually improve.
(d) [6 points] Is the following code thread-safe? Is it reentrant? Justify your answers.
Int gv; // global variable
Lock mutex; // global lock

```
f() {
…
Acquire(mutex);
gv = gv*2;
Release(mutex);
…
}
```

(+3) This is thread-safe. If many threads call f(), the mutex ensures only one thread at a time may access the global variable gv.
(+3) This is not reentrant. If a process grabs the mutex, is interrupted before releasing the mutex, and calls f() again, it will self-block on the mutex. Also, in general you want to avoid accessing global variables in reentrant code.

(e) [10 points] Describe five techniques that processes can employ to achieve IPC communication.

(+2 each)

shared memory – use shmat() to attach a piece of memory to multiple process' address spaces

message-passing via Unix-domain sockets, essentially a file, bind(), connect(), etc.

message-passing via Internet-domain sockets – set dest=localhost, more portable than Unix-domain sockets

message-passing via pipes – use pipe() to create a pipe, then fork(), child and parent read() and write() from different pipe descriptors

Signals – subset can be used for IPC, e.g. SIGUSR1, register signal handler via signal()

Also OK: message queues, RPC, indirect vs. direct message passing, named vs. anonymous pipes

Problem 3 (12 points): I/O, General OS

(a) [12 points] Explain the sequence of major steps that occur between when a process calls a library function to fscanf() some data from disk, and when the OS returns control to the calling process after the read to disk is complete. Assume the process blocks on the fscanf(). Your answer should consider what happens if the disk is busy. Describe one I/O optimization that a modern OS would employ in this sequence.

(+10) Some combination of the following steps in sequence:

- A process would call a library function, that is a wrapper on the actual system call or trap. In this case, a call to a read(), usually blocking.
- The library function would call `trap` SYS_READ
- This would invoke the OS's trap handler, which would first (i) change the mode bit and (ii) Save the process state
- index into the trap table, and find the system call handler for SYS_READ
- this handler would call the device manager and driver for the disk
- the driver would schedule a read for the disk, either polling the disk or waiting for an interrupt if busy
- when the disk is free, the disk would send an interrupt to the OS
- this would be caught by the interrupt handler, which would initiate the actual read of data from the device driver, which would load the device controller's DATA and CMD registers
- When the drive is finished, it will send another interrupt to the OS, which will invoke the interrupt handler to read the results
- These results would be passed back in the appropriate buffers to the process, which would then be woken up.

Solution ought to mention "interrupt" to receive full credit.

(+2) DMA (Direct Memory Access) is a technique for bypassing the CPU, so that large transfers of data can be made between memory and an I/O device, e.g. disk. In DMA, there is only one hardware interrupt when the entire data chunk is transferred, rather than for each small chunk, thus speeding up data transfer between disk and memory. Also OK optimizations: memory mapped I/O, interrupts > polling I/O, overlapped I/O allows CPU to do something else while waiting on disk I/O to complete, etc.

Problem 1: _____ / 44
Problem 2: _____ / 44
Problem 3: _____ / 12
Overall Score: _____ / 100