KEY TO EXAM ON PROCESSES AND SCHEDULING – SPRING 2007
CSC 262 – OPERATING SYSTEMS
NICHOLAS R. HOWE

1. **Vocabulary** (18 points). Give a definition or description of the following terms.

a). MIC chip

The Master Interrupt Controller receives signals from devices requesting interrupts, and relay those signals to the CPU.

b). O(1) Scheduler

Used in the Linux 2.6 kernel to schedule processes.

c). Atomic operation

An operation that cannot be interrupted by any interleaved operations.

d). Peterson Algorithm

A software mechanism for ensuring mutual exclusion between two cooperating processes.

e). TSL operation

Stands for "Test and Set Low". TSL(s,l) = <l:=s; s:=false;>

f). Two Generals Problem

A situation modeling distributed agreement, in which it is impossible to reach agreement using an unreliable channel.

g). Pipe

A mechanism used for interprocess communication, whereby one process sends output to the pipe, and another process reads input from it.

h). Multicast

A style of message passing communication where multiple sending processes send to multiple receiving processes.

i). Dutch Banker

A simple model for the allocation of a single resource.

2. **Process Scheduling** (24 points). Consider the process table below.

Process ID	0	1	2	3	4	5	6
Queue	0	0	1	2	1	2	2
Priority	0	1	3	1	4	3	5

Draw a Gantt chart showing which processes would execute when under the following scheduling policies over the next 1000 ms. You may ignore time spent on context switching overhead, and assume that each process runs for its full quantum without blocking or interrupts. However, a process that is allowed to run for 300 ms in total will terminate and be removed from the process table, perhaps allowing other processes to go.

a. Scheduling with multiple queues. Use round-robin scheduling between queues (all assumed to be equal priority; start in numerical order from 0) and strict priority scheduling within a queue (high numbers are high priority). Quantum is 100 ms.

0ms	100	200	300	400	500	600	700	800	900	1000
P1	P4	P6	P1	P4	P6	P1	P4	P6	P0	

b. Scheduling with multiple queues. Queue 0 is lowest priority, queue 2 is highest. Use strict priority scheduling between queues, and simple round-robin scheduling within a queue. (Assume that the process with the highest "priority" has been waiting longest for a turn.) Quantum is 150 ms.

0ms	150	300	450	600	750	900	1000
P6	P5	P3	P6	P5	P3	P4	

3. **Threads** (10 points). Which of the following will be shared by two related lightweight processes?

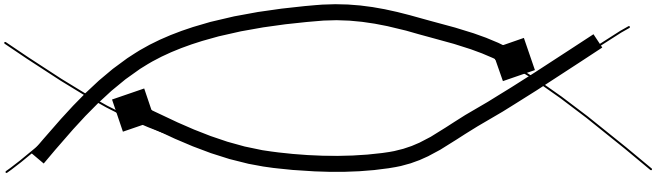
Code Segment -- Call Stack -- Data Segment (Heap) -- Status Registers – Resources

Code segment, data segment, and resources are shared.
Call stack and status registers are not.

4. **Deadlock** (24 points). The municipality of Deadlock City has a number of railroads passing through the town. Each railroad crosses the others at multiple points, and the trains passing though are long enough that they sometimes extend past several crossings at a time.

a. If multiple trains enter the city at once, it has been found that sometimes a situation arises where no train can continue to move forward. The only solution is to force one or

more trains to reverse slowly out of the city to let the others proceed. Draw a diagram showing one possible configuration of trains matching the description above.



b. It has been proposed to require all trains to "reserve" all the crossing points they wish to use before their entry into the city. Once a train has reserved a crossing point, no other train can reserve it until the original train has passed through. Trains unable to reserve all the crossings they need will have to wait outside the city until their crossings are free. Will this solve the problem? Comment on any disadvantages of this approach.

This will solve the problem: it prevents partial resource allocation. However, it is less efficient because trains will have to wait,and not all waiting trains would cause a problem if they were allowed to proceed. (Note that trains have to reserve all the crossings at once in order for this to work. If they can acquire them one at a time while they are waiting outside the city, then the problem is the same as before.)

c. An alternate solution is proposed. Each train would be given a unique priority number. Trains would declare the crossings they intend to use on entry to the city as described above. However, instead of being required to wait, trains can enter the city and proceed until they come to a crossing that has been declared on the itinerary of a train with higher priority, at which point they will have to wait until the higher-priority train passes through. Will this solve the problem in (a)? Comment on the advantages/disadvantages relative to (b).

This wouldsolve the problem: it prevents circular waiting. It is more efficient than the solution in (b), because some trains will be allowed to proceed that were not in (b). However, it still makes some trains wait unnecessarily. (Note also that we must be careful what to do when a new train comes in with higher priority than those already in the city. If we allow it to enter right away, it may become deadlocked with a lower-priority train.)

d. List Coffman's necessary conditions for deadlock, and identify any connections the the scenarios described above.

Coffman's conditions are mutually exclusive access to resources, non-preemptive granting of resources, partial resource allocation, and circular waiting. The scenario in (b) prevents partial allocation, while the scenario in (c) prevents circular waiting.

[Note: this problem was inspired by the city of Pine Bluff, Arkansas.]

5. **Classic Problems** (24 points). Consider the following proposed solution to the east-west bridge problem.

```
semaphore mutex(1), queue(0);
int crossing(0), waiting(0);
enum {east,west} dir = east;

void entryEast() {
  down(mutex);
  if (crossing > 0)
      && (dir == west) {
    waiting++;
    up(mutex);
    down(queue);
    waiting--;
  }
  dir = east;
  crossing++;
  up(mutex);
}

void exitEast() {
  down(mutex);
  crossing--;
  if (crossing == 0)
      && (waiting > 0) {
    up(queue);
  } else {
    up(mutex);
  }
}
```

```
void entryWest() {
  down(mutex);
  if (crossing > 0)
      && (dir == east) {
    waiting++;
    up(mutex);
    down(queue);
    waiting--;
  }
  dir = west;
  crossing++;
  up(mutex);
}

void exitWest() {
  down(mutex);
  crossing--;
  if (crossing == 0)
      && (waiting > 0) {
    up(queue);
  } else {
    up(mutex);
  }
}
```

a). Does this protocol ensure mutual exclusion between eastbound and westbound processes? Explain why/why not.

Yes, it does. If a process is in its critical section, then crossing will by greater than zero, and the direction variable will be set appropriately. Processes trying to enter in the opposite direction will be stopped.

b). Does the protocol prevent starvation in all cases? If yes, explain why. If not, explain how to fix it so that starvation cannot occur.

It does not. A steady stream of cars in one direction will prevent the other direction from crossing. To fix this, change `if (crossing > 0)` *to* `if ((crossing > 0)||(waiting > 0))`.

c). Contrast the protocol above with the protocol we developed on the homework for the north-south bridge, in terms of its efficiency.

It is less efficient. Because there are no separate queues for eastbound and westbound cars, we can only let one car at a time off the queue, even if all the waiting cars are going

in the same direction. In the homework protocol, multiple cars could be released when the direction changed.

d). Contrast the protocol above with the much simpler protocol below, again in terms of its efficiency.

```
semaphore bridge(1);

void entryEast() {
  down(bridge);
}

void exitEast() {
  up(bridge);
}

void entryWest() {
  down(bridge);
}

void exitWest() {
  up(bridge);
}
```

The simpler protocol always allows no more than one car at a time. The original protocol will allow multiple cars at some times (if they are new processes in the correct direction), so it is more efficient.