EXAM ON PROCESSES AND SCHEDULING – SPRING 2004
CSC 262 – OPERATING SYSTEMS
NICHOLAS R. HOWE

1. **Vocabulary** (24 points). Describe the meaning or role of the following terms in one or two sentences.

a). Semaphore Table

Data structure in the kernel that keeps track of all the semaphores on which processes are blocked.

b). Interrupt Controller

Hardware device that receives requests for interrupts from various sources and passes a signal to the CPU

c). Process Table

Data structure in the kernel that keeps track of vital data on all processes currently running.

d). Gantt Chart

A graphical device used to show order and lengths of process execution on a computer processor.

e). Peterson Algorithm

A software algorithm using shared variables to ensure mutual exclusion of critical sections between two processes. Uses no special hardware instructions.

f). Memory Interlock Instruction

A special hardware operation allowing multiple memory actions to take place as a single atomic instruction.

g). Monitor (the software construct, not the hardware device!)

A piece of modular software designed to handle process interaction and ensure mutual exclusion.

h). Two Generals Problem

A classic problem in process communication. Two processes communicating over an unreliable channel cannot synchornize actions without some chance of error.

2. **Process Management** (12 points). Write a few paragraphs explaining the differences between threads and heavyweight processes. Be sure to consider data structures, efficiency issues, and typical usage.

Heavyweight processes include all the data structures and resources to run on their own. This includes code, status, call stack, and data segments. Lightweight processes or threads, by contrast, include only those data structures necessary for independent execution in a shared data environment. The have their own status and call stack, but share code and data with other threads in the same group.

Because threads share a data environment, switching between two related threads takes less time than switching between two heavyweight processes. They also take less memory than an equal number of corresponding heavyweight processes would, due to the sharing.

Threads are typically used for handling different tasks that work towards a single ultimate goal – handling different aspects of a program that interacts with the user, for example. Heavyweight processes are more appropriate when the tasks to be carried out are unrelated (which would make sharing of data pointless).

3. **Interrupts** (16 points) Below are a number of events that take place during the handling of an interrupt. Number them in chronological order.

a. __3__ CPU signals on INTA line
b. __5__ Key state registers stored in process table stack and replaced
c. __7__ Interrupt code read off system bus
d. __1__ Signal reaches MIC or SIC on IRQ
e. __8__ Execution jumps to handler for specific interrupt type
f. __4__ MIC or SIC puts interrupt code on system bus
g. __6__ User programmable registers stored in process table stack
h. __2__ MIC signals on INT line

4. **Race Conditions** (16 points). For each of the following programs, state the possible values for *x* and *y* if the program terminates. Also, state whether the program will always terminate, sometimes terminate, or never terminate.

You may assume that the variables are initialized as follows:

x : integer init 2
y : integer init 3
s0 : general semaphore init 0
s1 : general semaphore init 1
s2 : general semaphore init 1

a). **cobegin** x := x+y; // y := x+y; **coend**

Any interleaving is possible. (x,y) ∈ {(5,5),(5,8),(7,5)}
Always terminates.

b).
```
cobegin
   DOWN(s1); x := x+y; UP(s1);
//
   DOWN(s1); y := x+y; UP(s1);
coend
```

Either order is possible, but no interleaving.
(x,y) ∈ {(5,8),(7,5)}
Always terminates.

c).
```
cobegin
   DOWN(s0); x := x+y; UP(s1);
//
   DOWN(s1); y := x+y; UP(s0);
coend
```

Second branch goes first. (x,y) = (7,5)
Always terminates.

d).
```
cobegin
   DOWN(s1); DOWN(s2); x := x+y; UP(s2); UP(s1);
//
   DOWN(s2); DOWN(s1); y := x+y; UP(s1); UP(s2);
coend
```

Either order is possible, but no interleaving.
(x,y) ∈ {(5,8),(7,5)}
Sometimes terminates.

5. **Scheduling** (12 points). Consider the hypothetical process table shown below. For each of the scheduling policies listed, state which process would run next. Also, assuming no process blocks, terminates, or becomes unblocked, which processes would *never* run? If it matters, you may assume that process 3 has run most recently.

Process ID: 0 Priority: 15 Quanta: 2 Status: Ready Next: 1	Process ID: 1 Priority: 0 Quanta: 5 Status: Ready Next: 2	Process ID: 2 Priority: -10 Quanta: 0 Status: Ready Next: 3	Process ID: 3 Priority: 8 Quanta: 8 Status: Blocked Next: 4	Process ID: 4 Priority: 8 Quanta: 8 Status: Ready Next: 0

a. Round robin

Process 4 next. Process 3 would not run.

b. Strict priority (higher numbers = higher priority)

Process 0 next. No other processes would run.

c. Linux `SCHED_OTHER`

Process 1 would run next. All processes would run eventually except 3.

6. **Semaphores** (12 points). Consider the following protocol for the *sleeping barber* problem:

```
const int chairs(5);
int waiting (0);
semaphore customers(0);
semaphore barbers(0);
semaphore mutex(1);

1     barber: process
2        while true do
3           DOWN(customers);
4           DOWN(mutex);
5           waiting := waiting-1;
6           UP(barbers);
7           UP(mutex);
8           {cut hair};
9        endwhile;
10    endprocess;

11    customer: process
12       DOWN(mutex);
13       if (waiting < chairs) then
14          waiting := waiting+1;
15          UP(customers);
16          UP(mutex);
17          DOWN(barbers);
18          {get haircut}
19       else
```

```
20          UP(mutex);
21        endif;
22    endprocess;
```

a). How would the protocol's behavior change if lines 4, 7, 12, 16, and 20 were eliminated? If there could be a change in behavior, describe a specific scenario where it would be evident.

The value of waiting could become corrupted if two processes executed lines 5 and 14 at the same time. This could make processes wait when they shouldn't, or vice versa.

b). How would the protocol's behavior change if lines 16 and 17 were exchanged? If there could be a change in behavior, describe a specific scenario where it would be evident.

This would cause deadlock, since the customer would block before releasing the mutual exclusion. No other processes could make progress, since they would all stop at lines 4 or 12.

c). How would the protocol's behavior change if line 6 were eliminated? If there could be a change in behavior, describe a specific scenario where it would be evident.

Customer processes would never wake up for their haircut.

7. **Deadlock Avoidance** (8 points). A particular system has 2 Scanners, 3 Plotters, 1 Surveyor, and 2 Printers. Consider the following set of resources, current allocations, and potential needs:

	Scanners		Plotters		Surveyors		Printers	
	Current	**Max**	Current	**Max**	Current	**Max**	Current	**Max**
Process A	1	**1**	0	**1**	0	**1**	0	**1**
Process B	0	**1**	1	**1**	1	**1**	0	**0**
Process C	0	**1**	0	**0**	0	**1**	1	**1**
Process D	0	**0**	1	**3**	0	**1**	0	**1**

a. Is it safe to grant Process B access to a Scanner? Why or why not? (Give either a plan for satisfying all processes completely, or set of requests that would be impossible to satisfy.)

Yes. Process B would be completely satisfied, and would eventually release all resources. Then we could (for example) satisfy processes D, C, and A.

b. Is it safe to grant Process C access to a Scanner? Why or why not?

No. If process B requests a Scanner and the remaining processes requests a Surveyor, no processes can be satisfied.

c. Is it safe to grant Process A access to a Printer? Why or why not?

Yes. We can still satisfy process A, B, or C's maximum requests. Following those, we could satisfy D.

d. Is it safe to grant Process D access to a second Plotter? Why or why not?

Yes. We could still satisfy process B, then any of the others.