Solutions
15-410, Spring 2006, Homework Assignment 1.

1 Memory Madness (10 pts.)

1.1 4 pts

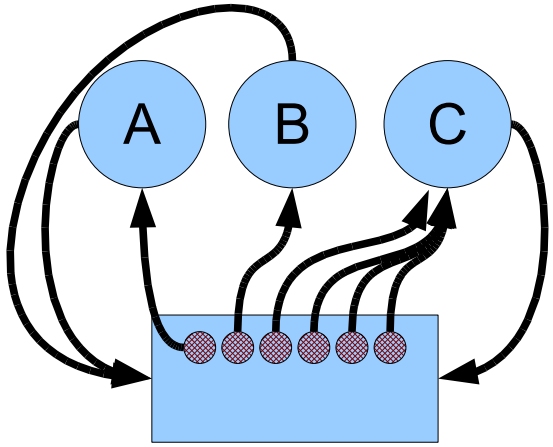
Illustrate how this system can deadlock.

Execution Trace

time	Process A	Process B	Process C
0			
1	t_a() /*1*/		
2		t_a() /*1*/	
3			t_a() /*1*/
4			t_a() /*2*/
5			t_a() /*3*/
6			t_a() /*4*/
7			t_a() ...
8		t_a() ...	
9	t_a() ...		

1.2 2 pts

Sketch a process/resource graph showing the deadlock.



Note: no rescuer
(everybody in a cycle)

1.3 4 pts

Is the following state safe? Why or why not?

Since Process B has no "head room" (according to its declaration, it cannot allocate any more resources), it will complete its work and exit in "reasonable" time. This will give the resource allocator 200M to work with. Since neither of the two remaining processes has more than 200M of head room, the resource allocator could choose to run either one to completion, at which point there would be plenty of free resources to run

the third to completion. Thus there are two safe sequences from this state (B, C, A; B, A, C) making this clearly a safe state.

2 Paradise Lost (5 pts.)

The problem is that we check `going_out_of_business` only once before committing to the loop, even though any time we aren't holding the mutex somebody could set it to true. That's essentially a *second* form of losing one's paradise.

```
mutex_lock(&m);

w = (workitem *) 0;

while(!w && !going_out_of_business) {
  if (!(w = (workitem *) dequeue(q)))
    cond_wait(&new_work, &m);
}
mutex_unlock(&m);
return(w);
```

3 IRET (10 pts.)

On many processors the question of whether we're restoring through a user-mode trap frame or a kernel-mode trap frame is encoded in the equivalent of `%EFLAGS`, which in those cases contains a single bit indicating "user mode" versus "supervisor mode" (interestingly enough, some IBM documentation on the PowerPC refers to "user mode" as "problem mode"). However, these processors typically restore a fixed number of registers regardless, so while they know the answer to this question they don't need to know it.

An IA32 processor, however, *does* need to know whether to restore three or five registers. It turns out not to be a coincidence that in each push case the final three registers are the same and in the same order. This means that `IRET` can pop, and inspect the values of, `%EIP`, `%CS`, and `%EFLAGS` in order to decide whether to continue on and restore `%ESP` and `%SS`.

Hopefully it is clear to you that `%EIP` wouldn't decide the matter. If you are familiar with other processor architectures you might expect `%EFLAGS` to contain the answer, but it doesn't. This leaves only `%CS`, which, thankfully, *does* contain the answer–as you should recall, the processor privilege level is encoded in `%CS`, so `IRET` can compare the current (kernel) privilege level to the privilege level which was running before the "surprise" to see if a stack switch had to occur on the way in. If so, then we need one on the way out, so we pop `%ESP` and `%SS`.