Solutions
15-410, Spring 2008, Homework Assignment 1.

# 1 PUSHA (10 pts.)

## 1.1 5pts

Add code to the template below so that running it and inspecting memory appropriately would make it possible to see what PUSHA really does.

```
.globl pushy
.globl examine

pushy:
    PUSHA    # protect our caller

    # now color registers
    MOVL $1, %eax
    MOVL $2, %ebx
    MOVL $3, %ecx
    MOVL $4, %edx
    MOVL $5, %esi
    MOVL $6, %edi
    MOVL $7, %ebp
    PUSHA
examine:
    POPA    # undo effects of test
    POPA    # restore for caller
    RET
```

## 1.2 5pts

Now specify a small number of Simics commands you would use to make your decision.

```
Floppy inserted in drive 'A:'. (File /tmp/de0u/hw1s/p1/bootfd.img).
simics> break (sym examine)
Breakpoint 1 set on address 0x1007dd with access mode 'x'

simics> r
Reached Breakpoint 1
Code breakpoint 1 reached.
[cpu0] cs:0x001007dd p:0x001007dd  popad
pushy () in game

simics> x %esp 64
ds:0x00117f60                         0600 0000 0500 0000          ........
ds:0x00117f70  0700 0000 887f 1100 0200 0000 0400 0000  ................
ds:0x00117f80  0300 0000 0100 0000 d489 1100 54a0 0200  ............T...
ds:0x00117f90  c47f 1100 a87f 1100 20a0 0200 947f 1100  ........ .......
ds:0x00117fa0  2080 1100 708a 1100                       ...p...
```

Note that as we read "down" along the memory dump we are reading "up" in memory (but "down" the stack). We see %edi, %esi, and %ebp, then something which sure looks like a saved %esp, and then %ebx (not %ebp). So it looks as if in the quoted text the first mention of %ebp is spurious.

# 2 Safety in Numbers (10 pts.)

## 2.1 4pts

| time | Process A | Process B | Process C |
|---|---|---|---|
| 0 | request(1) | | |
| 1 | | request(1) | |
| *2 | | | request(1) |
| 3 | release(1) | | |
| 4 | | release(1) | |
| 5 | | | release(1) |

## 2.2 6pts

The state after the request at $T = 2$ is granted is unsafe because no safe sequence exists. Note that *every* process can allocate one more tape drive without violating its maximum-resource declaration, and that the number of available tape drives is zero. Since the maximal resource needs (2) of *no* process can be satisifed with what it holds (1) plus what is free (0), there is no process which can certainly run to completion. If there is no first process in the sequence then there is no sequence of such processes, so there is no safe sequence. By the way, the system is safe again after $T = 3$.

# 3 Latches and Locks (20 pts.)

The key observation is that these mutexes involve spin-waiting, given the assumptions that we are on a multi-processor and that the instruction sequences protected by them are short (we discussed exactly this thinking in class). We totally cannot afford to violate the "soon" assumption and hold one of these mutexes for milliseconds, so we don't.

```
typedef struct lock {
  mutex_t m;
  int avail;
  cond_t released;
} lock_t;

int lock_init(lock_t *lp)
{
  mutex_init(&lp->m);
  lp->avail = 1;
  cond_init(&lp->released);
  return (0); // ok to skip most error code on exams
}

int lock_acquire(lock_t *lp)
{
  mutex_lock(&lp->m);
  while (!lp->avail)
    cond_wait(&lp->released, &lp->m);
  lp->avail = 0;
  mutex_unlock(&lp->m);
  return (0);
}

int lock_release(lock_t *lp)
{
  mutex_lock(&lp->m);
  lp->avail = 1;
  cond_signal(&lp->released);
  mutex_unlock(&lp->m);
  return (0);
}
```