

CS 574
Midterm Exam
October 10, 2005

The following exam is open book and open notes. You may feel free to use whatever additional reference material you wish, but **no electronic aids** are allowed. Please note the following instructions. There will be a ten point deduction for failure to comply with them:

- start each problem on a new sheet of paper
- write your social security number, but not your name, on each sheet of paper you turn in
- show your work whenever appropriate. There can be no partial credit unless I see how answers were arrived
- be succinct. You may lose points for facts that, while true, are not relevant to the question at hand

You have until 10:20 to finish the exam. The questions are equally weighted.

1. Process memory regions are managed using two completely separate data structures (linked lists and red-black trees). What is the purpose of each of them?
2. When an interrupt or exception occurs, the hardware puts just enough of the CPU state on the system stack to be able to do a return. The Linux assembly-code interrupt handler puts all the rest of the CPU state on the stack before doing any other work. Why?
3. Suppose two kernel threads are obtaining input and placing it in a buffer. They each use the same code:

```
while (1) {
    awry[i++] = newin();
    if (i > 100) i = 0;
}
```

The `newin()` function correctly obtains one unit of input. `awry` and `i` are both shared between the threads. In this machine, the `i++` operation is not atomic.

- (a) Show that it is possible for inputs to be lost using this code (assume some other code someplace is correctly removing the inputs from the array – the inputs aren't lost due to `i` wrapping around to 0).
 - (b) Modify the code so that it will correctly put each new input in a new location in the array. You may add new local or global variables, you may restructure the code, and you may add any of the synchronization primitives from Chapter 5. A correct solution will only lock the operations that actually need it (so don't just insert a spin-lock or something at the beginning and the end of the loop).
4. When the kernel decides to switch the current process, the first step is to change the Page Global Directory, which installs a new address space. How does the kernel keep executing when its address space has just been changed out from under it?
 5. In writing an ordinary program, what ideas from the kernel's slab allocator could be useful in doing memory management? This is assuming you're doing your own memory management, and not just calling `malloc()` whenever you want a new object.