CSCI 3753 Spring 2014 **Vince Coghlan**

Homework #2

Problem 1. [20 Points] What is a process context switch? When does it occur? Describe the steps involved in implementing a process context switch

A process context switch is when the OS takes the state of a process and stores in somewhere, then restores another process state. This allows for multitasking. This occurs when the OS wants to begin to execute another process while a current process is still running. The entire process will generally follow the following steps: first, a syscall or other interrupt that will enter the OS. This syscall will then run and begin to store various things onto the OS's stack frame. It will then put another process's variables and such back on the stack, and return from the interrupt to begin executing that code.

Problem 2. [10 Points] What are the advantages and disadvantages of threads vs processes?

Since processes require a full serperate address space, threads are much easier to create. Since threads share memory, however, one must be careful to modify the data one thread at a time, and this can become costly for the program, and the programmer. An advantage of a process is that a process can run multiple threads within it, whereas a thread is just a thread. Processes are much more simple and far easier to understand for a typical programmer. Threads require locking, mutexi, semaphores, and are, overall, much more involved.

Problem 3. [40 Points] Unisex bathroom problem: CU wants to show off how politically correct it is by applying the U.S. Supreme Court's "Separate but equal is inherently unequal" doctrine to gender as well as race, ending its long-standing practice of gender-segregated bathrooms on campus. However, as a concession to tradition, it decrees that when a woman is in the bathroom, other women may enter, but no men, and vice versa. A child may enter the bathroom only if there is at least one adult present in the bathroom. Finally, at most N (N > 1) individuals may use the bathroom at any time.

You task is to write three functions: man_use_bathroom( ), woman_use_bathroom( ), and child_use_bathroom( ). Provide a monitor-based solution that manages access to the bathroom. Your solution should be fair, starvation free and deadlock free.

Since much of the question isnt very specific, there will be a women, b men, and c children trying to use the bathroom. Assume that all of these people were declared in either the init function, or somewhere else. The details are in the comments, since much of this code will not run. Also, the problem contains inherent starvation, since if the bathroom is full of women, and more women keep coming, the men will have to wait until they are done. There is not a way of fixing this without only letting a very small number of a gender use the bathroom at a time and trade off, which is innefficient. To solve this problem I tried to make it as fair as possible, and in only very extreme cases will the person waiting starve, or find another bathroom.

```
monitor bathroommonitor {
  // four states of the bathroom
  enum {mens, womens, empty, full} state;
  // each allows each person to be signaled.
  condition w[a];
  condition m[b];
  condition c[c];
  // number of men, women, children, and people currently
  // using the bathroom
  int men;
  int women;
  int children;
  int num;

  man_use_bathroom(int i) {
    // add man to list of people waiting to use the bathroom
    men++;
    // if there are women in there wait
    if (state == womens) {
      m[i].wait();
    }
    // if its full wait
    else if (state == full) {
      m[i].wait();
    }
    else if (state == (empty || mens)) {
      // go into the bathroom
```

```
      num++;
      // he is no longer waiting in line
      men--;
      // set the state to full if he makes it full
      if (num == N) {
        state = full;
      }
      else {
        state = mens;
      }
      // pee or whatever
      use_bathroom();
      // leave
      num--;
      // if he was the last person, signal the next women
      if (num == 0) {
        state = empty;
        if (women) {
          w[women].signal();
        }
      }
      else {
        state = mens;
        // if there are more children waiting than other guys,
        // let them go first
        if (children > men){
          c[children].signal();
        }
        else {
          m[men].signal();
        }
      }
    }
}
woman_use_bathroom(int i) {
  // this all works the same as with the men.
```

```
    women++;
    if (state == mens) {
      w[i].wait();
    }
    else if (state == full) {
      w[i].wait();
    }
    else if (state == (empty || womens)) {
      num++;
      women--;
      if (num == N) {
        state = full;
      }
      else {
        state = womens;
      }
      use_bathroom();
      num--;
      if (num == 0) {
        state = empty;
        if (men) {
          m[men].signal();
        }
      }
      else {
        state = womens;
        if (children > women) {
          c[children].signal();
        }
        else {
          w[women].signal();
        }
      }
    }

}
```

```
  child_use_bathroom(int i) {
    children++
    if (state == full) {
      c[i].wait();
    }
    else {
      num++;
      children--;
      if (num == N) {
        state = full;
      }
      use_bathroom();
      num--;
      // signal the group that has the most people waiting for
      // fairness.
      if (men > women) {
        m[men].signal();
      }
    }
  }

  init() {
    state = empty;
    // in here somewhere we will set up all of the people who
    // want to use the bathroom, for now I will leave it general.
  }
}
```