**Operating System Part 4**

**Austin Blythe, Matthew Francis**

**Account: cs441104**

**8 April 2013**

**Program problem/statement:**  
For this iteration of the Operating System, we have implemented a queuing system. Whenever a program is being told to run, instead of beginning its run immediately, it is added to the queue (with the proper priority), and waits for the rest of the commands to come through before beginning to process it. During execution, only one process block is handled, and it has a maximum of 4 ticks to be used (including the original command to run it). Once it has run, it is moved to a shadow queue, to allow more processes to continue on without causing starvation. They are only moved from the shadow queue back to the main queue if and only if the main queue is empty. After processing a command, if there are any process control blocks that have had no execution time at all, they are promoted to a higher priority queue.

**O/S** **Architecture:**

* 16-bit words
* 256-word memory (16 bits) word addressable
* 3 general purpose registers (1-3), 1 special accumulator (A; 0)
* 8-bit program counter (PC)
* condition code flags: = or zero(010), > or positive(001), < or negative (100)
* machine instruction cycle - fetch, decode, opfetch, execute, writeback

**Difficulties encountered:**

* We were unsure on how many process control blocks to execute after the user(s) type in their commands. Currently, we have it set to only run one, meaning if a run, run, nop is entered, only user 1 will have their program run, it moves to the shadow queue, and user 2, because It has had no run time, is promoted in priority.
* The amount of variables in the process control block is increasing with each program, and it is becoming difficult to keep track of all the information that needs to be shifted around when moving the pcb in arrays.

**Procedure:**  
 We spent more than two hours debating on the best way to approach the queue problem, using a whiteboard to display it, we worked through different ways to handle the promotion and moving of shadow queue’s.

Once we had decided on a strategy, it was fairly straightforward coding from there on, only having to deal with bugs that occurred.  
**Additional Observations:**  
 We should have started on this project earlier, to allow time to ask about the proper order of actions handled, meaning move shadow queue over before or after priority promotion, things of that nature.

**Assumptions:**

We are assuming that only one execution cycle is being handled every time it cycles through the users. Meaning that it takes 2 rounds to handle each user, user 1 will execute its command, and be moved to the shadow queue. Being that user 2 has had no run time at this point, it is promoted to priority 1.  
  
**Detailed table of work:**  
A majority of the work occurred on the 6th and 7th, with 2.5 hours of the 6th debating on how to handle the queuing system.

|  |  |  |
| --- | --- | --- |
|  | **Austin** | **Matt** |
| **4/5/2013** | 3 hrs | 2 hrs |
| **4/6/2013** | 9 hrs | 8 hrs |
| **4/7/2013** | 4 hrs | 3 hrs |
| **4/8/2013** | 3 hrs | 2 hrs |