Documentation

Process

In P1, we created two queues that are always sorted according to priority (highest -> lowest). We have four states for the processes – NEW, RDY, BOR, RUN. The two queues represent Blocked and Ready processes. There are four priorities – HIGH (0), MEDIUM (1), LOW (2), LOWEST (3). Initially, all processes are put into the Ready queue. The process that is currently running is not in the Ready queue. We also added a priority element to the PCB, to include priority functionality.

There are pointers to the head and tails of the two queues – headReady, tailReady, headBlocked, tailBlocked. headReady and tailRady point to the head and tail in the ready queue, while the headBlocked and tailBlocked point to the head and tail of the Blocked queue. The processes will run infinitely according to the priority. The release processor obtains the next highest priority to be ran. The process switch will switch the processes whenever there is higher priority process that is just set(using set_process_priority()) or processes with the same priority. In the case of the higher priority just set, it will pre-empt the currently running process with lower priority compare to the process just set.

We also implemented the set_process_priority() and get_process_priority() as per the requirements. Set_process_priority() will set the priority of the process by passing in the ID of the process and the priority we want the process to be. Get_process_priority() is a getter method that is used to get the priority of the process by passing in the process ID and it will return the priority.

Memory

In P1, we created a data structure for holding memory blocks. The memory block contains an address to the free block, a pointer to the next one and a released flag. In request memory, if the headBlocked is equal to NULL then we push it onto the blocked queue. We have an initial 4 byte padding for the pointer. In memory initialization we go down to up, then we bring the head down. In request memory we go upwards. In release memory, we set the new node's next pointer to the head. And then we move the head to the new node.