CS1550 Project 1 Report

Colby King (cmk97@pitt.edu)

September 20, 2020

1. FIFO vs. Priority Queues in Semaphores

In this report, I am going to compare the benefits and drawbacks of using FIFO and priority queues in implementations of semaphores. When processes call wait() on a semaphore, and the its value is less than 0, it is put into a waiting queue. These are the queues which I will be discussing.

For the purposes of this report, I will assume a FIFO queue is implemented as doubly linked list with pointers to both the head and tail nodes. Therefore, insertion and removal from the beginning and end of the list takes $O(1)$ time. For the priority queue, I will assume that it is implemented as a tree structure using a max heap. Therefore, insertion and removal from the priority queue takes $O(\log n)$ time, where n is the number of processes in the queue.

The first and most obvious benefit of using a FIFO queue in a semaphore implementation is the runtime achieved by the insert and remove operations. Enqueuing a new process is as simple as appending it to the end of the queue using the reference to the tail node. The operation will always run in constant time. The same runtime can be achieved for dequeueing a process by removing it from the front of the list using the reference to the head node. By contrast, the priority queue's operations will necessarily take $O(\log n)$ time. Since insertion and removal operations are frequently used in implementations of a semaphores, having a constant time is very desirable.

Another benefit of using a FIFO queue is its simplicity with regard to implementation. Because it is simple to implement, it's easier to manage and less likely to contain bugs, which is very important when programming in kernel space. However, while FIFO is easier to implement, its simplicity poses limitations. A FIFO queue, by definition, will queue all processes in the order in which they are received. This may be what we want in some instances; however, there are many cases where we might want to prioritize some processes over others, regardless of age. In these situations, a priority queue is required.

While a priority queue is a slightly slower and a little more complicated to implement, it still operates very quickly relative to other data structures, and can also provide additional flexibility for our semaphore. If we wanted the ordering of our process queue to be anything other than the age of the process, we can use a priority queue to order it on any metric we choose about the process. For example, if we wanted our semaphore to wake up the next process with the shortest burst time, we could use a priority queue to sort the heap according to this metric. This sort of feature is not possible with a FIFO queue.

In summary, for more simple implementations of semaphores where we want the waiting processes to run in the order in which they are received, we should use a FIFO queue. It is both faster and simpler to implement than other alternatives. However, if we want our semaphore implementation to queue waiting processes by any other metric than age, we should use a priority queue since a FIFO queue is not capable of this functionality.