Jay Offerdahl
EECS 678
Lab 8 Report

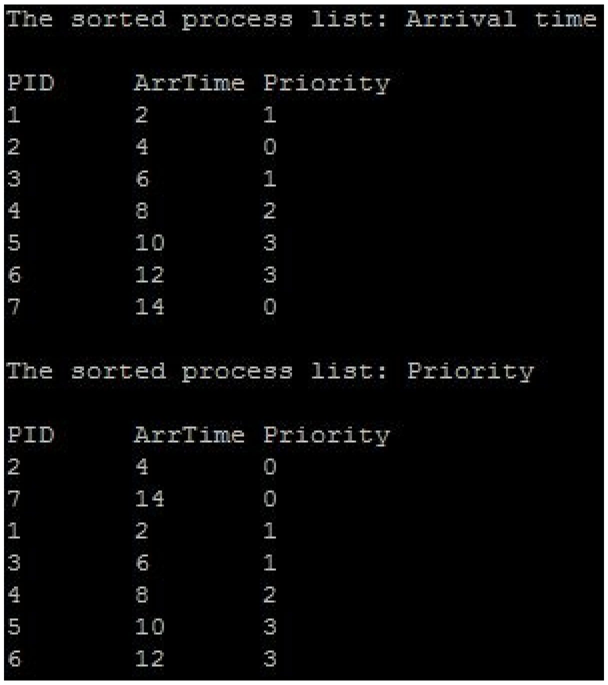
Lab 8 Report: Function Pointers

For this week's lab on function pointers, I based most of my program off of our project two "queuetest.c" test file, which we used to test the priority queue used in the scheduler.

To begin, I implemented a process data structure at the top of main.c which defined the type "process_t". This structure included the process's process ID, the time at which the process arrived, as well as the priority associated with the process. I initialized these processes from the datafile process.txt, which included each process on one line, with the three attributes separated by a space. I stored all of processes in a process_t pointer array.

Next, I first sorted the processes by arrival time using a function call to qsort(...). I sent in the parameters for the processes array, the number of processes, the size of a process, and the compare function which returns a negative value if the first parameter is less than the second. This compare function called "compareArr" takes in two void pointers, as specified by qsort, and casts the pointers to "process_t" objects before taking their member variables and computing the difference.This allows the sort function to sort in ascending based on arrival time. After sorting the data based on arrival time, I printed out the process list, which produced correct output.

Lastly I sorted the processes array based on their priorities using qsort again. This time, I passed in a function pointer to comparePri(...), a second compare function I wrote which first calculated the difference of priorities, returning the higher priority (lower number). If this return value was calculated to be 0, the compare function would then return the input with the earlier arrival time. This ensured that when comparing two processes, we could effectively find out which process should be placed before another process.



```
The sorted process list: Arrival time

PID     ArrTime Priority
1       2       1
2       4       0
3       6       1
4       8       2
5       10      3
6       12      3
7       14      0

The sorted process list: Priority

PID     ArrTime Priority
2       4       0
7       14      0
1       2       1
3       6       1
4       8       2
5       10      3
6       12      3
```

It's worth noting that when sorting priorities, they are still sorted in "descending" order, because a lower number priority (e.g. 0 as compared to 3) has higher priority. So, while the priorities seem like they're in ascending order on the output, they're actually in descending order.