Kurt Slagle
EECS[645]

Lab 08 – Quick Sort and Function Pointers

Data Structure

```
struct process
{
     int pid;       //The process ID
     int arrTime;   //The arrival time
     int priority;  //The priority of the process
};
```

A structure to hold the information about each process. I used this because it made organization easier.

Comparing and Sorting

I used 2 different comparison functions.

```
int sortArrival(const void * a, const void * b)
{
     struct process p = *(struct process *)a;
     struct process q = *(struct process *)b;

     return (p.arrTime – q.arrTime);
}

int sortPriority(const void * a, const void * b)
{
     struct process p *(struct process *)a;
     struct process q *(struct process *)b;

     if(p.priority == q.priority)
          return p.arrTime – q.arrTime;
     return p.priority – q.priority;
}
```

Having 2 functions allowed me to separate the two sorting requirements easily.
We already know the number of processes (it was shown in the slides) -7, so I declared the structure as:
`struct process processArray[7];`

There are 3 data values per process, so I declared a 21-element array to hold the data
`int processes[7 * 3];`

I declared a buffer to hold the data being read from the file
`char line_buffer[BUFSIZ];`

I declared a file pointer
```
FILE *in;
```

I then tried to open the file, and alerted the user if it could not be opened.
```
in = fopen(“process.txt”, “r”);
if(!in)
{
     printf(“Could not open file %s for reading”, “process.txt”);
     exit(-1);
}
```

I then set up arrays to temporarily hold the data that is held in the text file
```
int pids[7] = {0};
int arrTimes[7] = {0};
int priorities[7] = {0};
int sizeOfProcess = sizeof(struct process);
```

I then filled up the arrays that hold the info
```
for (int i = 0; i < 7; i++)
{
     pids[i] = processes[3 * i];
     arrTimes[i] = processes[3 * i + 1];
     priorities[i] = processes[3 * i + 2];
}
```

Then I use those arrays to fill up the data in the processes array
```
for (int i = 0; i < 7; i++)
{
     processArray[i].pid = pids[i];
     processArray[i].arrTimes = arrTimes[i];
     processArray[i].priority = priorities[i];
}
```

Now it's time to sort
```
qsort(processArray, 7, sizeOfProcess, sortPriority);
```

…

```
qsort(processArray, 7, sizeOfProcess, sortArrival);
```

Between the two sorts, output was provided to show the result of the firs sort, and then output was provided to show the results of the second sort.
Sorted by priority

--Output on next page

```
PID:   2     Arrival Time:   4    Priority:   0
PID:   7     Arrival Time:  14    Priority:   0
PID:   1     Arrival Time:   2    Priority:   1
PID:   3     Arrival Time:   6    Priority:   1
PID:   4     Arrival Time:   8    Priority:   2
PID:   5     Arrival Time:  10    Priority:   3
PID:   6     Arrival Time:  12    Priority:   3


Sorted by arrival time
PID:   1     Arrival Time:   2    Priority:   1
PID:   2     Arrival Time:   4    Priority:   0
PID:   3     Arrival Time:   6    Priority:   1
PID:   4     Arrival Time:   8    Priority:   2
PID:   5     Arrival Time:  10    Priority:   3
PID:   6     Arrival Time:  12    Priority:   3
PID:   7     Arrival Time:  14    Priority:   0
```