Matt Billone & Venkata Gutta

Functions and Implementation:

In main.cpp:

getInputTime- takes in the input line and turns it into a string of characters. Truncates the string down until it finds the time that the input is to be put into the system and returns it. Used everytime an input gets put into the system

displayStatus- checks if the terminating input is the current input being read, if it is not then it prints all the information from each of the queues that are holding jobs for the entire system using the traverseAndPrint function from this file and the printStatus function from Queues.cpp. If the D 9999 command is executed, we have our own final status printing implementation within the main for this project.

readInputCommand- takes in the current input line and all of the queues for the system. We need to take in all the queues of the system since in this function, other functions related to each job are called which has the queues in their function calls. When reading the command, this function changes the string for the current line being read into a array of characters, and then truncates the array to the exclusively look at which input command being called. After finding which input command is being called, it will call the corresponding function associated with that input command.

configSystem- takes in the input character array that is given from readInputCommand function. Reads the first index of the character array which should be either, M(memory), S(devices), or Q(quantum). It will read all of these starting with M and ending with Q. To make sure that the correct letter is always the first index of the character array, we truncate the input to the next letter after each one is read. For example, (C 9 M=45 S=12 Q=1) will be (M=45 S=12 Q=1) when it gets to the config system command. Then it will truncate after reading M and the number for it to (S=12 Q=1) and will truncate and read the correct information until the character array is empty.

createJob- takes in the array of characters from readInputCommand, the system queue, and the submit queue. We start by creating linked list nodes for the job. We make the job node that will be put through the system and a copy node that will be put into the system queue to be used later when we print out all of the system history at the end of the simulation. We set the statuses for the job to be the submit queue since that is the first queue that the job gets put into before anything else. The arrival time gets set to the current time of the system since when the job is created into the system is the same thing as the arrival time. Then, using the same truncating method that we did in config system, we get the memory required for the job, the devices required for the job, the time needed to run for the job, and the priority. We added an else to catch any input commands that are not recognized for this input. Finally, we put the job node into the submit queue, and the copy into the system queue using the addToEnd function from Queues.cpp.

requestDevices- takes in array of characters from the input, cpu, ready, wait, and system queues. Starts off by reading the job number and the devices being requested by the job from the input. After those are read, we use banker's algorithm to see if we can grant the request by looking at the requested devices from the job and comparing it to the devices that the system has available. If the devices that the job requires is less than the available, then the devices will be granted to the job. If not, then the function will return if the devices is more than the max amount of devices that the system holds. After granting the devices to the job, the devices granted boolean becomes true, and the job is removed from the cpu. After that, the available devices is updated and the job goes back to the ready queue. If the devices are greater than the available devices but less than the maximum devices that the system has, then the job is put on the wait queue and the next job available is taken from the ready queue onto the cpu. If none of this happens, then an error message comes up.

realeaseDevices- takes in the character array from the input, the cpu, wait, ready, and system queues. First, the job number and the amount of devices to be released is read from the input. Please note that all of the reading from the input uses the the truncation that we have been using from the start to get certain data from the inputs. Next, we check if the job is running on the cpu, if it is not we display a message saying that the job is not on the cpu. Then we check if the devices granted boolean is true for the job. If they have not been granted, then the job cannot release devices until the devices required for the job is allocated. Then we check if the devices that are supposed to be released is equal than the devices that were requested from the job previously. If they are not equal then the devices are not released. If the devices to be released are the right number, then the available devices gets updated, effectively saying that the devices have been released and are now available for other jobs to use. After the devices have been released, then the function checks the waiting queue to see if there are any jobs that can now be fulfilled with the new number of devices available.

readyQHandling- takes in the system, ready, and cpu queues. For this to run, there needs to be something in the ready queue. If there is, we check if the cpu queue is empty, and then put the job on the cpu if it is free. After that we update the status of the job to "CPU" since it is now running on the cpu.

submitQHandling- takes in the system, hold1, hold2, and submit queues. Executes if the submit queue contains jobs. Checks if the system can take on the job by comparing the memory and devices of the job are less than or equal to the amount of memory and devices that the system has to offer. If the system cannot handle the job, then the job is moved from the submit queue to the rejected queue(the status of the job is also changed to REJECTED). If the job can be handled by the system, then the job gets moved from the submit queue to one of the holding queue based on its priority. If it has a priority of 1, then the job will be put into hold queue 1 to then be sorted by shortest job first. If it is has priority of 2, it will go into hold queue 2 to then be sorted by first in first out. This will happen until the submit queue is empty and all jobs are in the right holding queue.

cpuHandling- takes in the system, cpu, ready, hold1, hold2, wait, and completed queues. Only runs if there is a job running on the cpu. While on the cpu, the time left for the job to complete ticks done as the cpu handling runs. The job runs on the cpu until the job is finished executing(the timleft variable on the job node == 0). Once the job is finished running, we calculate its turn around time and its weighted turn around time. After that, we release the memory and devices that the job was using back to the system. After that, we remove the job from the cpu, add it to the complete queue, and update its status to the "COMPLETED" status. We then move all the remaining jobs to their new respective queues since there is room for them. To do this we call waitQHandling, hold1QHandling, hold2Qhandling, and readyQHandling.

hold1QHandling- takes in the system, hold1, and ready queues. Only runs if there are jobs in the hold1 queue. First, we set up a temporary queue to do work on, and set a shortestJob variable to a really high number, this case being 9999. We take the first job in the queue and set it's runtime to the shortest run time of the queue to compare it to the rest of the runtimes of the jobs in the queue. As long as there is a job inside of the temp queue, and it is not the head node, we check to make sure that there is enough memory and available devices to send it to the next queue. After traversing through the queue and making sure that the shortest time job is getting removed, we remove that job from the queue and send it to the ready queue, changing its status to ready queue in the process. We also update the available memory in the system.

hod2QHandling- takes in the system, hold2, and ready queues. Like the rest of the handling functions, will only run if there is a job in the queue. This queue is our FIFO queue and does just that. We check that there is enough memory and devices in the system to move on to the next queue, and take the job off the front of the queue, moving it to the ready queue, and updating the available memory of the system and status of the jobs being moved around.

waitQHandling- takes in the system, wait, and ready queues. Makes a temp node to be used to change stuff around. If the devices required of the first job on the queue is less than the devices available, then the job gets removed from the wait queue and put back onto the ready queue.

updateStatus- goes through the queue and searches for the job that is being changes. Once it finds it, it changes the status of the job to the correct status.

charToInt- takes in the part of the character array that contains a number, extracts the number and returns it.

In Queues.cpp:
printStatus- takes in a queue and prints out the information regarding whether or not the job was completed, its turnaround time, its weighted turnaround time if it is completed. If it is not completed it will print how much time there is left until it is completed.

addToFront- takes in the node and the queue to be added to, adds it to the front of the queue

addToEnd- takes in node and the queue to be added to, adds it to the end of the queue

Remove- takes in the queue and the job number to be removed, removes it and returns it to be placed into a different queue.

printQ- takes in a queue and prints out all the job numbers and position of each job in the queue

The input doesn't work well the the implementation should work right and what all the functions do and perform is explained above.

Running the code:
Makefile is included
So run "make all" which will compile main.cpp and Queues.cpp
Then run the command "make compile" then do "make run"