ECE-C353: Systems Programming
Homework Assignment 5: Message Queues

You have been given a simple program (`control.c` that simulates receiving commands over the network. It its current state, this program simply prints commands received over the simulated network to the screen along with a time stamp (in milliseconds) and the command's priority. Compile this program with:

```
$ gcc -o control control.c -lm
```

Your job is simple. Modify the provided program to send these messages to a message queue with their corresponding priority. If `Ctrl+C` is pressed, then `control.c` should put the command `SHUTDOWN` in the message queue with the highest priority, print "`Sending SHUTDOWN message.`", close the message queue, unlink the message queue, print "`Shutting down.`", and then terminate successfully.

Now, write another program named `worker.c` that will be controlled by the messages sent to the message queue by `control.c`. The behavior of `worker.c` is simple:

- `worker.c` can only be in one of two states: **running** or **paused**. It should start in the paused state.
- If `worker.c` is **running** it will attempt to calculate:

$$Result = \sum_{i=0}^{N} i \tag{1}$$

  where N is effectively infinity, but `UINT_MAX` will suffice for this assignment. Store the value of *Result* in an `unsigned long int`.
- If `worker.c` is **paused** it will stop accumulating values of i into *Result*, but it will maintain its current values of i and *Result* so that it may resume the computation where it left off if it is later set to the `running` state again. It should try to not use the CPU when paused. See the man pages for `sleep()` and `pause()`. Either or a combination of these is fine. Avoid 100% CPU usage when paused. You can check CPU usage with the `top` command.
- If the `RESET` command is received from the message queue, the values of i and *Result* will both be set to zero and it will print "`Reset.`" to the screen. The state of `worker.c` will not change. If it is already **paused**, it will stay paused. If it is already `running`, it will continue running.
- If the `PRINT` command is received from the message queue, `worker.c` will print the current value of *Result* to the screen in the format "Current Result: 234", for example.
- If the `RUN` command is received from the message queue and `worker.c` is currently **paused**, it will print "`>>RUNNING<<`" to the screen and change its state to **running**. If `worker.c` is already in the `running` state, this command has no effect and nothing will be printed to the screen.
- If the `PAUSE` command is received from the message queue and `worker.c` is currently **running**, it will print "`[PAUSED]`" to the screen and change its state to **paused**. If `worker.c` is already in the `paused` state, this command has no effect and nothing will be printed to the screen.
- If the `SHUTDOWN` command is received from the message queue, `worker.c` closes the message queue, prints "`Shutting down.`" to the screen, and terminates successfully.
- If `worker.c` is unable to open the message queue, it should print an informative message to `stderr` and exit with failure.

Note, messages generated by `control.c` all have a priority of `0` by default. This is to make debugging your program easier. Once you have everything working, try changing `#define USE_PRIORITY` to `1` to test your program with messages that have different priorities.

Deliverables:

You will submit 3 file via BBLearn:

- `control.c` – Modified to send commands to a message queue.
- `worker.c` – Your program that is properly controlled by commands received from the message queue.
- `README.txt` – A description of your code including how to compile and run it.