Jacob Reiser

For this project, I implemented a listener function to create a listener that waited for requests from the clients before sending those requests to a shared buffer. This buffer was handled by a function that created a number consumer threads based on the number of threads specified at the running of the webserver. These consumers exist in a while loop that is constantly checking the buffer for tasks to run. In the multi_threaded_server function, I created the pthread for the listener and n pthreads for the workers as specified at the running of the webserver. Finally, There is a while loop that attempts to merge the threads and retries for those consumers that crashed until the jobs are completed.

The shared data structure is a global array variable that is managed using a mutex lock and semaphores. When a new request is made, the listener thread locks the request array, adds the new request from a client using the global in variable, and then releases the array. The workers are constantly checking the buffer for jobs, and when one is available they will lock the buffer, take a specified task, decrement the buffer counter using the global out variable. This ensures each job is only being accessed by one thread at a time to avoid conflicts. The other handler for this is the tryjoin, so that any crashed threads wait for all successful threads to run before generating new threads to handle the crashed requests. This was tested by testing a client that ran more requests than threads present in the webserver. Test such as 20 request for 5 threads with a 10% crash rate and other permutations where more requests than threads are run.