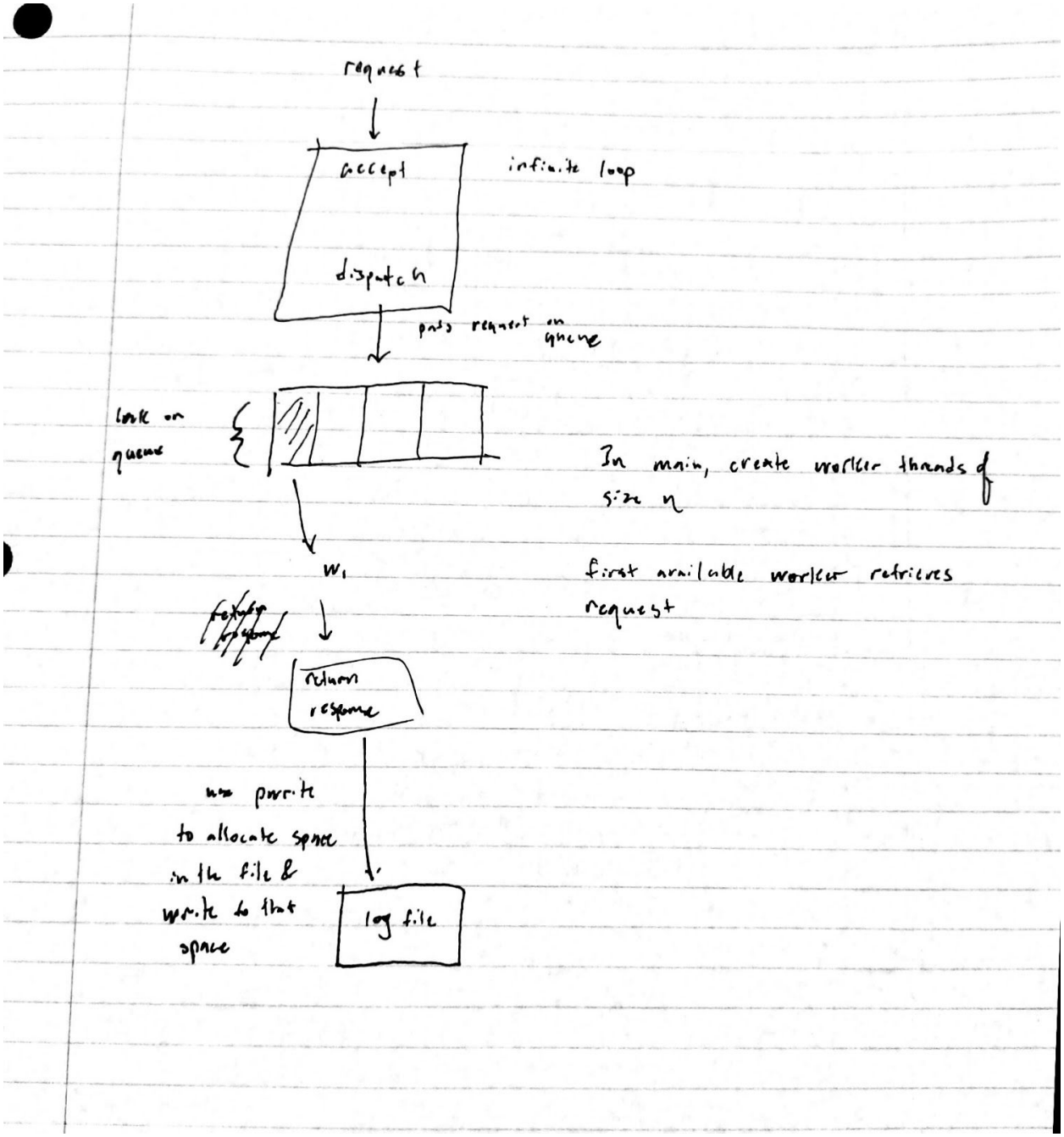
The main objective in this assignment is to implement multithreading for our server from HW1. I plan on using a pool of worker threads to handle multiple requests concurrently.
I will have a dispatcher thread that only enqueues requests. Then, when a request is queued, this will send a signal to the worker threads to dequeue the request. I got inspiration from this YouTube video: https://youtu.be/P6Z5K8zmEmc

High Level (drawing is depicting a dispatch thread with a queue of worker threads)



Major parts of the assignment:
Using GETOP to parse command line requests

1. Simple enough, read documentation and learn how to use switch case statements

Creating a queue and worker threads to manage concurrent requests

1. Initialize a threads of size (specified by user) (these threads will never be killed)
    a. A struct of our global variables to be shared between the threads should be passed as an argument.
2. In an infinite loop
    a. Accept requests from the socket and enqueue these requests
    b. This causes a condition variable to trigger and dequeue will be triggered forcing a worker thread to remove a request
    c. Worker thread will do a lot of HW1 in which it parses the request and returns a response. In addition, each worker thread will need to check if logging is enabled (bool). More of this in logging

Logging: Using pwrite to calculate a global offset (this should be its own function)

1. Before we do anything, we must create a variable assigned to the current global offset. This current offset will be of local scope and incremented until the request has finished. Next, before we do any writes, we must calculate the global offset for the next request. This will be the current global offset plus the exact space the next request will occupy. There will be locks around these operations as these will be shared among threads. In essence, these are critical regions that must not be accessed by other threads.
2. Then we iterate through the file descriptor, convert to hex, and then pwrite to the file.

I should create a second function to log the errors since they will not be converted to hex.

Healthcheck:

1. When we enqueue a request, we should increment a shared global variable which keeps track of all the requests. There should be a lock around this operation.
2. In the case of an error, we should increment a global error count var. There should be a lock around this operation.
3. When healthcheck is specified by GET, it should return a custom hex message. Will be similar to logging so some of that code can be reused. If healthcheck is requested as a PUT or HEAD, my program should return a 403.