1. Goals

The goal of this program is to implement a multi-threaded HTTP server that performs GET and PUT requests and write out the record of each request, including both header information and data (dumped as hex).

2. Assumptions

I am assuming we will need only one mutex to achieve synchronization. I might need more than that.

3. Design

The general approach I am taking is to first create worker threads that sleeps when there are no requests, then in dispatch thread where it listens to the incoming requests, it pushes the request into a global queue, and broadcast to workers so that they wake up and perform requests pulled from the global queue.

I will create worker threads using pthread_create that sleeps if there are no requests. I can achieve this by creating a function that takes an array of pthread_t and size as parameters and uses for loop to create a separate thread for each pthread_t in the array. In each thread, I will use condition mutex and condition cond to sleep and wake up depending on if there are requests. After waking up, a thread can take a request and process the request i.e GET and PUT.

The functions that perform GET and PUT requests will be bounded by a mutex so that at any time only one request can be processed. We need this synchronization so that only one function can access a file and perform requests. And also to reserve a space in the file it's going to use to log the request.

For reserving a space in the file, I will use a shared variable OFFSET that saves the starting point where the thread will log the request. I will use a function to calculate the space required to log the requests. We can calculate this by getting header length and content length and other extra spaces and endlines. All this should be done using synchronization so that each request has a unique reserved space in the file. Note we don't need synchronization for actually logging the request in the file because every request has their unique space, and can write in the file at the same time.

The dispatch thread listens to the incoming requests and pushes each request in the global queue, and wakes up worker threads using synchronization. We need synchronization here so that requests don't get lost if dispatcher tries to push more than one request into the queue at the same time.

4. Pseudocode

```
1.    procedure parseHeader:
2.        sscanf(header)
```

```
3.              return filename, httpMethod
4.      procedure GET:
5.              lock_mutex
6.              writeFileInput(file, socket)
7.              localOffset = globalOffset
8.              reserveSpace(header, content)
9.              unlock_mutex
10.             writeLog(localOffset, header, content)
11.
12.     procedure PUT:
13.             lock_mutex
14.             fd = open(filename, O_WRONLY|O_CREAT)
15.             write(fd, input, length)
16.             localOffset = globalOffset
17.             reserveSpace(header, content)
18.             unlock_mutex
19.             writeLog(localOffset, header, content)
20
21.     procedure reserveSpace:
22.             globalOffset += headerLength + ContentLength + extraStuffLength
23.
24.    procedure writeLog:
25.             pwrite(logfd, buffer, length, localOffset)
26.
27.     procedure initWorkers:
28.             for(1...size)
29.                     pthread_create(workers[i], NULL, processRequests, NULL)
30.     processRequest:
31.             lock_mutex
32.             if request is empty:
33.                     pthread_cond_wait()
34.             else
35.                     clientSocket = request.pop()
36.             unlock_mutex
37.             read(clientSocket, buffer, BUFFSIZE)
38.             parseHeader(buffer)
39.             if httpMethod == Get:
41.                     GET(filename, socket)
42.             else if httpMethod == PUT:
```

```
43.                 PUT(filename, socket)

44.
45.     Main:
46.         //create socket server
47.         //dispatcher
48.         while(true)
49.             //accept request
50.             lock_mutex
51.             queue.push(request)
52.             //broadcast
53.             unlock_mutex
```