1 Goal

The goal of this program is to add on to our previous assignment to have the server handle multiple concurrent requests by implementing multithreading. The other goal is to implement logging of requests as they come through on the server. For a put request we will log the header (ReqType FileName length #), the contents of the file in hex, and the tail (========\n) and for a get request we will just log the header and tail.

2 Assumptions

This program should be ran on Linux 18.04. Logging failed requests was not implemented. I ran on a dual core laptop (two threads).

3 Design

The general approach I'm taking is in main I will first parse arguments. I then set up the server and create the thread pool by running the dispatcher function on each thread. After that the code will loop in an infinite while loop, taking requests from Accept() locking and pushing them to the queue, then unlocking and signaling.

In the dispatcher function, there's an infinite while loop inside this loop I lock, then have another while loop checking for if the q is empty, if it is empty, we will put that thread sleep. It will be woken up by the signal call in main when thread is needed. Outside this inner while loop, I will process the request. I will pop the socket file desc from the queue then unlock. I will then parse the header and call the get or put function depending on which request it is. If it's neither of those, I will send a 400 bad request and close the connection.

4 Pseudocode

This is the pseudocode for the new functions.

```
Var globalConLen = 0; //length for log
```

```
getReq(int newsockfd, char *fileName, char *ver, char * logfile):
    Fd = Open(fileName)
```

```
        If (fd < 0):
                if(ERRNO == EACCES):
                        Send 403 error.
                        Close socket
                        Return;
                else:
                        Send 404 error
                        Close socket
                        Return
        End Initial IFs
        lock()
        Int64 miniFileCur = globalConLen;
                miniFileCur += xtraBufLen
                miniFileCur += xtraBufLenT
        unlock()

        Fs = getFileSize(fd)
        curPos = 0;
        if (logfile != NULL) {
                 pwrite(logfd, xtraLogBufH, xtraBufLen, miniFileCur);
                miniFileCur += xtraBufLen;
                pwrite(logfd, xtraLogBufT,xtraBufLenT,miniFileCur);
                  // add written len to curpos len thingy
                miniFileCur += xtraBufLenT;
        }


        While (CurPos < FS):
                read(fd, buf,..)
                send(newsockfd,buf,..)
        End While
        close(Fd)
        Send Okay and Content Length
        close(newsockfd)

End getReq()




putReq(int newsockfd, char *fileName, int64_t len, char *ver, char *logfile):
        pthread_mutex_lock()
        Int64_t miniFileCur = globalConLen;
```

```
        globalConLen += xtraBufLenH
        globnalCOnLen += len * 3;
        globalConLen += ceil(len / 20) * 9;
        globalConLen += xtraBufLenT;
        globalConLen += xtraBufLenT;

        unlock();
        if (logfile != NULL)
           pwritten = pwrite(logfd, xtraLogBufH, xtraBufLenH, miniFileCur);
        if (logfile != NULL)
           miniFileCur += xtraBufLenH;

        Bool createdFile = false
        If open(fileName) < 0:
                createdFile = true
        Fd = open(filename,....)
        curPosFile = 0
        contentLen = len
        Char buf[11]
        while(curPosFile < contentLen):
                recv(newsockfd,buf...)
                Int pwriten = pwrite(logfd,....miniFIleCur);
                miniFileCur += pwritten;
                write(fd,buff..)
                If recv or write fail -> close file and socket and return;
        If (CreatedFile):
                dprintf(newsockfd,"HTTP/1.1  201 Created")
        Else:
                dprintf(newsockfd,"HTTP/1.1 200 OK")

        close(fd)
        close(logfd)
        close(newsockfd)
End putReq()

Void *dispatcher(void * argp){
        while(1){
                lock()
                while(q.empty()){
                        pthread_wait()
                }
        }
```

.
.
.
.
Code below parses message, calls get if get req, put if put req.

}