DESIGN DOCUMENT FOR ASGN2

My program has 5 modules:
Main()
readfromfile()
writetofile()
dispatch()
entry_point()

DATA STRUCTURES USED:
struct args{
	bufferdata = used to save the header of the request
	filename = name of the file
	length = length of the file or 0(initially) when it is a get request
	fd = file descriptor which is initialized to 0
	rqst_socket = the file descriptor of the socket where the request is coming from
	rqst_flag = to check if it was a PUT(1) or GET(0)
}

vector<args> v :used to store the requests as structs

Main():
/* I will be skipping over the logistics of the server
	because they are the same as asgn1. I will focus on the multi-threading.
*/
Creates one thread that is send to the dispatch() function because it will be our dispatch thread
The main function creates the specified number of threads and sends them to the entry_point(), each thread has a struct pointer which will be used once the requests come in

The main continues to where it waits for requests
Once request is recieved it:
First, parses the request(either PUT or GET)
	if(PUT){
			//parse filename and content length
			//push a struct to the vector

	v.push_back({bufferdata,filename,length,0,rqst_socket,1});
	}
	else if(GET){
			//parse the filename
			//push struct to the vector
			v.push_back({bufferdta,filename,0,0,rqst_socket,0});
	}
	else{
			send back an error
	}
The main functions only job is to recieve requests and push them onto the vector and
ofcourse it is always running inside of a while(1) loop so that the server only

terminates when Ctrl+D is hit

```
void readfromfile(void *i)
-------------------------------------------
This function performs the same details as the asgn1
The only difference is that instead of passing each varible to it such as:
filename
length etc....
the thread uses the struct pointer that was assigned to it to access those
variables
This allows for their to be minimal race conditions since each thread has
a different instances of its pointer
 Some psuedocode:


 for(int i = 0; i<strlen(filename);i++){
      check if each char in filename comlies with requirements
      if it doesnt:
      send err msg to socket
      close socket
      return;
 }
 if(length of filename != 27){
      send err msg to socket
      close socket
      return;
 }

mutex.down(nreader);
num_readers +=1;
if(first reader){
      mutex.down(&writing);
}
mutex.up(nreader);

get file descriptor(fd) for file by using open()

if(fd >0)
{
      mutex.down(read);
      //this is because i was having issues with sockets so i decided that
only one thread can read at a time
      if(fstat doesnt return error)
      {
                  read data of file to a buffer
                  close file
                  send ok response
                  send content length
                  send buffer of the file content
                  close socket
      }
```

```
      mutex.up(read);
      mutex.down(nreader);
      nreaders--;
      if(youre the last one){
      writing.up();
      mutex.up(nreader);
}


}
else{
      send appropriate err msg
      close socket
      return;
}
return;
}

void writetofile(void *i)
-----------------------------------------
writetofile is similar to readfrom file
I will skip the filename syntax check because it is also done in here as
well


psuedocode:

/*syntax check for file name and length of filename*/

//open the file and retrieve the file descriptor(fd)
//the algorithm used here is the same as profs r/w problem in the lecture
mutex.down(&writing)
if(open was succesful){
      //valreadfile will hold number of bytes read
      valreadile = read(in from the socket pointed by the i ptr)
      write(write the valreadfile number of bytes to the file);
      close file
      send the creat resp
      close socket
      mutex.up(&writing);
      return;
}
else{
      send appropriate err msg
      close socket
      return;
}

void dispatch(void *i)
-----------------------------------------
This fucntion is responsible for alerting the sleeping thread when there
is a new request
```

available in the vector. count is a global variable which is incremented everytime a new request comes in from main() so if all threads are working it continues to signal it until one of them goes to sleep. This is because the signal function has no effect if all the threads are awake.
psuedocode:

```
while(1){
      while(count == 0){
      //wait
      }
      signal thread
}

void *entry_point(void *i)
```

This is where the working threads are sent in order to wait for the signal of the dispatch thread

psuedocode:

```
while(1){
      mutex.down(&threads)
      wait();
      mutex.down(count);
      count--;//since one request is being popped off the vector
      //ptr is the pointer pointing to the element in the vector
      set i = ptr;
      //move ptr of the vector to the next element in the vector
      ptr++;
      mutex.up(&thread);
      if(PUT){
                  go to writetofile
      }
      else{
                  go to readfromfile;
      }
}
```