Lab 5
Web Server

Tasks

- Basic Web Server
- Concurrent Web Server
    - Defaults to the basic web server in special cases
    - Handles three different methods for implementing concurrency

Basic Server

Components

- Opening the socket
- Listening for incoming requests
- Accepting a request and processing it
    - Reading the HTTP header
    - Getting the document path
    - Expanding the path*
    - Figuring out the Content type
    - Writing the HTTP reply header
    - Writing data
    - Close connection

Opening/Listening on a socket

Opening (from daytime-server.cc)

```
int  masterSocket  =  socket(PF_INET, SOCK_STREAM, 0)
int  err  =  setsockopt(masterSocket,  SOL_SOCKET,  SO_REUSEADDR,
             (char  *)  &optval,  sizeof(  int  )  );
int  error  =  bind(  masterSocket,
         (struct  sockaddr  *)&serverIPAddress,
          sizeof(serverIPAddress)   );
```

Listening (again from daytime-server.cc)

```
int  error  =  listen(  masterSocket,  QueueLength);
```

What is QueueLength ?

Incoming Requests

- Waiting for an incoming request

```
while(1){
        int slaveSocket = accept( masterSocket,
              (struct sockaddr *)&clientIPAddress,
              (socklen_t*)&alen)
}
```

- What is missing ?

```
while(1){
        int slaveSocket = accept( masterSocket,
              (struct sockaddr *)&clientIPAddress,
              (socklen_t*)&alen);
}
```

Incoming Requests

- Waiting for an incoming request

```
while(1){
        int slaveSocket = accept( masterSocket,
              (struct sockaddr *)&clientIPAddress,
              (socklen_t*)&alen)
}
```

- What is missing ?

```
while(1){
        int slaveSocket = accept( masterSocket,
              (struct sockaddr *)&clientIPAddress,
              (socklen_t*)&alen)
       processRequest(slaveSocket);
       close(slaveSocket);
}
```

processRequest(socket)

1. Read the HTTP header(adapted from daytime-server.cc)

```
while((n = read(socket, &newChar, sizeof(newChar)){
            length++;
            if(newChar == ' '){
                  if I have not seen GET
                        int gotGet = 1;
                  else if I have not seen docPath
                        curr_string[length-1]=0;
                        strcpy(docpath, curr_string);
            }
            else if(newChar == '\n' && oldChar == '\r'){
                     break;
            }
            else{
                  oldChar = newChar;
                  currString[length-1] = newChar;
            }
}
Read the remaining header and ignore it. We don't need it
```

2. get the document path

3. map the document path to the real file

```
char cwd[256] = {0};
cwd = getcwd(cwd);
if docpath begins with "/icons" make filepath
        cwd+"http-root-dir/"+docpath
if docpath begins with "/htdocs" make filepath
        cwd+"http-root-dir/"+docpath
else make filepath
        cwd+"http-root-dir/htdocs"+docpath
```

When is the docpath "/" ? what do we do ?

cwd+"http-root-dir/htdocs/index.html"

4. Expand filepath

Expand ".." , return error if it results in a path which maps to

cwd+"/http-root-dir"

or higher

Valid : lore:1234/dir1/subdir1/../subdir2/a

Invalid : lore:1234/dir1/subdir1/../../../

(will take you to the parent of htdocs, which is INVALID)

How do we check this ?

Check if the length of the expanded path is less than the length of

cwd+"/http-root-dir"

5. Determine Content type

```
if(endsWith(filepath, “.html”) ||
        endsWith(filepath, “.html/”)){
                strcpy(contentType, “text/html”);
}
if(endsWith(filepath, “.gif”) ||
        endsWith(filepath, “.gif/”)){
                strcpy(contentType, “image/gif”);
}
else
        strcpy(contentType, “text/plain”);
```

what is endsWith(const char *, char *) ?

6. Open the file

If open() fails, you need to send a 404

●

7. Send HTTP Reply Header

Format :

```
HTTP/1.0  <sp>  200 <sp> Document  <sp> follows  <crlf>
Server: <sp> <Server-Type>  <crlf>
Content-type:  <sp> <Document-Type> <crlf>
{<Other Header Information> <crlf>}*
<crlf>
<Document  Data>

HTTP/1.0  <sp>  404 File Not  Found <crlf>
Server: <sp> <Server-Type>  <crlf>
Content-type:  <sp> <Document-Type> <crlf>
<crlf>
<Error  Message>
```

Example, sending 404, File Not Found

```
const char  *notFound = “File  not  Found”;
write(socket,  protocol, strlen(protocol));
write(socket,  space,  1);
write(socket,  "404",  3);
write(socket,  "File",  4);
write(socket,  "Not",  3);
write(socket,  "Found",  5);
write(socket,  clrf, 2);
write(socket,  "Server:", 7);
write(socket,  space,  1);
write(socket,  serverType,  strlen(serverType));
write(socket,  clrf, 2);
write(socket,  "Content-type:", 13);
write(socket,  space,  1);
write(socket,  contentType,  strlen(contentType));
write(socket,  clrf, 2);
write(socket,  clrf, 2);
write(socket,  notFound, strlen(notFound));
```

- Sending the file , start writing after two <clrf> in the reply header

```
while(count  = read(from file)){
        if(write(to  socket)  != count){
                perror("write");
        }
}
```

- Various error conditions to send a error reply http header (like 404)
    - Illegal access, request path is in or above http-root-dir/ and is not http-root-dir/htdocs or http-root-dir/icons
    - File does not exist or the path is invalid, example : contains "...", open() will take care of this.

In the above cases send a error reply to the client

Concurrent Web Server

Process based (-f)

Modify the logic for calling processRequest() as follows :

```
while(1){
    int slaveSocket = accept(blah blah blah);
    pid_t slave = fork();
    if(fork == 0){
            processRequest(slaveSocket);
            close(slaveSocket);
            exit(EXIT_SUCCESS);
    }
    //Why do we need this ??
    close(slaveSocket);
}
```

Clean Up Zombie Child Processes , use code from the SHELL project (lab 3).

Thread based (-t)

Modify the logic for calling processRequest() as follows :

```
while(1){
    int slaveSocket = accept(blah blah blah);
    //initialize pthread attributes
    .
    .
    .
    pthread_create(&tid, &attr, (void * (*)(void
    *))processRequest, (void *)slaveSocket);
    close(slaveSocket);
}
```

Will this work ?? why or why not ?

Thread based (-t)

Modify the logic for calling processRequest() as follows :

```
while(1){
        int slaveSocket = accept(blah blah blah);
        //initialize pthread attributes
        .
        .
        .
        pthread_create(&tid, &attr, (void * (*)(void
        *))processRequestThread, (void *)slaveSocket);
}
```

```
processRequestThread(int socket){
        processRequest(socket);
        close(socket);
}
```

Thread Pool based (-p)

Before blocking on accept, do the following, block on accept in the threads instead.

```
pthread_t tid[5];
for(int i=0; i<  0;i++){
        pthread_create(&tid, &attr,
                    (void *(*)(void *))poolSlave,
                    (void *)masterSocket);
}
pthread_join(tid[0], NULL);


void poolSlave(int socket){
    while(1){
            int slaveSocket = accept(blah blah blah);
            //check if accept worked
            processRequest(slaveSocket);
            close(slaveSocket);
    }
}
```

Some Issues :

1. You can add a mutex around the accept call in poolSlave as mentioned on the handout.

2. In the process based concurrent version, the accept can return -1 if SIGINT is caused with SIGCHILD, so you should modify the check for the value of slaveSocket as follows :

```
if(slaveSocket == -1 && errno == EINTR){
        continue;
}
```

This just continues in the while loop.

3. In the thread pool example, you should call accept() only in the threads, you should start the threads after the call to listen().

Arguments

- If no argument is specified , assume default port and basic server model (single threaded)
- If only port is given, assume basic server mode (single threaded)
- Both concurrency type and port can be specified as follows :

```
myhttpd [-f|-t|-p] [<port>]
```

- It is a good idea to verify that the arguments given for port and type of concurrency are valid.

Other Instructions

- Edit the given Makefile, if you are using pthreads (which you will for threaded concurrency), you should add

  -lpthread

  to the flags. Also add a target for myhttpd.

- Turnin instructions are on the handout.

- Please read the handout and the FAQ.

All the Best :)