This Homework

is very interesting!

Introduction

- In this assignment...
- You have to modify a simple web server to a multiplexing web server
- You don't have to handle the network communication. We will handle that part.

Http Connection

- HTTP is a request/response standard typical of client-server computing
- web browsers act as clients
- an application running on the computer hosting the web site acts as a server

http://tw.yahoo.com

```
GET /index.html HTTP/1.1
Host: tw.yahoo.com
```

```
HTTP/1.1 200 OK
Date: Mon, 23 May 2005 22:38:34 GMT
Server: Apache/1.3.3.7 (Unix)  (Red-Hat/Linux)
Last-Modified: Wed, 08 Jan 2003 23:11:55 GMT
Etag: "3f80f-1b6-3e1cb03b"
Accept-Ranges: bytes
Content-Length: 438
Connection: close
Content-Type: text/html; charset=UTF-8
```

We will provide...

- A Simple Web Server
- handle only one request
- get stuck when file size > socket buffer

Your Job is...

- A Multiplexing Web Server
- handle multiple request
- Return whole file no matter how large it is

```c
// Initialize http server
init_http_server( &server, (unsigned short) atoi( argv[1] ) );

maxfd = getdtablesize();
requestP = ( http_request* ) malloc( sizeof( http_request ) * maxfd );
if ( requestP == (http_request*) 0 )
{
    fprintf( stderr, "out of memory allocating all http requests\n" );
    exit( 1 );
}
for ( i = 0; i < maxfd; i ++ )
    init_request( &requestP[i] );
requestP[ server.listen_fd ].conn_fd = server.listen_fd;
requestP[ server.listen_fd ].status = READING;

fprintf( stderr, "\nstarting on %.80s, port %d, fd %d, maxconn %d, logfile %s...\n",
        server.hostname, server.port, server.listen_fd, maxfd, logfilenameP );
```

requestP:

file des	0	1	2	3	4	...	...	1022	1023
flag	1	1	1	0	0	0	0	0	0

↑ Usually the server.listen_fd

```c
typedef struct {
    char hostname[512];     // hostname
    unsigned short port;    // port to listen
    int listen_fd;      // fd to wait for a new connection
} http_server;

typedef struct {
    int conn_fd;        // fd to talk with client
    int status;         // not used, error, reading (from client)
    // writing (to client)
    char file[MAXBUFSIZE];  // requested file
    char query[MAXBUFSIZE]; // requested query
    char host[MAXBUFSIZE];  // client host
    char* buf;          // data sent by/to client
    size_t buf_len;     // bytes used by buf
    size_t buf_size;        // bytes allocated for buf
    size_t buf_idx;         // offset for reading and writing
} http_request;
```

```
// Main loop.
while (1)
{
    // Wait for a connection.
    clilen = sizeof(cliaddr);
    conn_fd = accept( server.listen_fd, (struct sockaddr *) &cliaddr, (socklen_t *) &clilen );
    if ( conn_fd < 0 )
    {
        if ( errno == EINTR || errno == EAGAIN ) continue; // try again
        if ( errno == ENFILE )
        {
            (void) fprintf( stderr, "out of file descriptor table ... (maxconn %d)\n", maxfd
            continue;
        }
        ERR_EXIT( "accept" )
    }
    requestP[conn_fd].conn_fd = conn_fd;
    requestP[conn_fd].status = READING;
    strcpy( requestP[conn_fd].host, inet_ntoa( cliaddr.sin_addr ) );
    set_ndelay( conn_fd );

    fprintf( stderr, "getting a new request... fd %d from %s\n",
            conn_fd, requestP[conn_fd].host );
```

file des	0	1	2	3	4	...	...	1022	1023
flag	1	1	1	1	0	0	0	0	0

server.listen_fd ↑ ↑ **accept** return fd = conn_fd

```
while (1)
{
    ret = read_header_and_file( &requestP[conn_fd], &err );
    if ( ret > 0 ) continue;
    else if ( ret < 0 )
    {
        // error for reading http header or requested file
        fprintf( stderr, "error on fd %d, code %d\n",
                requestP[conn_fd].conn_fd, err );
        close( requestP[conn_fd].conn_fd );
        free_request( &requestP[conn_fd] );
        break;
    }
    else if ( ret == 0 )
    {
        // ready for writing
        fprintf( stderr, "writing (buf %p, idx %d) %d bytes to request fd %d\n",
                requestP[conn_fd].buf, (int) requestP[conn_fd].buf_idx,
                (int) requestP[conn_fd].buf_len, requestP[conn_fd].conn_fd );

        nwritten = write( requestP[conn_fd].conn_fd,
                          requestP[conn_fd].buf,
                          requestP[conn_fd].buf_len );
        fprintf( stderr, "complete writing %d bytes on fd %d\n",
                nwritten, requestP[conn_fd].conn_fd );
        close( requestP[conn_fd].conn_fd );
        free_request( &requestP[conn_fd] );
        break;
    }
}
```

```c
static int read_header_and_file( http_request* reqP, int *errP );
// return 0: success, file is buffered in retP->buf with retP->buf_len bytes
// return -1: error, check error code (*errP)
// return 1: continue to it until return -1 or 0
// error code:
// 1: client connection error
// 2: bad request, cannot parse request
// 3: method not implemented
// 4: illegal filename
// 5: illegal query
// 6: file not found
// 7: file is protected
```

Demo

This program works perfectly when only one request

What if ...

file des	0	1	2	3	4	5	6	7	...
flag	1	1	1	1	1	0	0	0	0

What should you do?

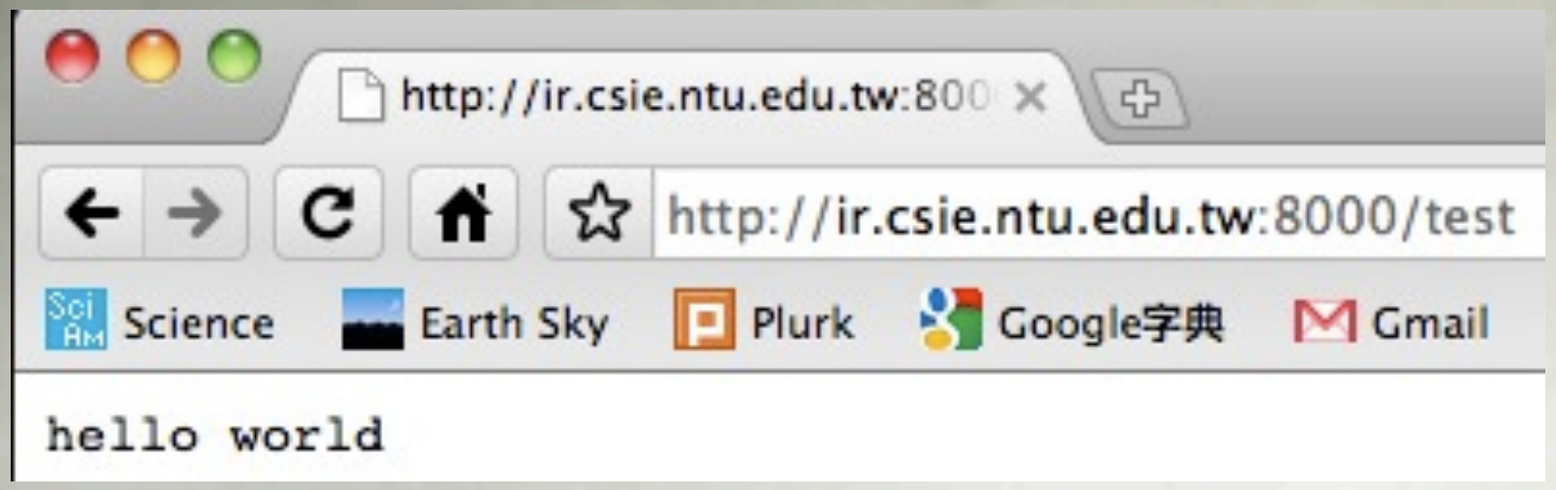
- Use **`select`**
  - **select**() allows a program to monitor multiple file descriptors, waiting until one or more of the file descriptors become "**ready**" for some class of I/O operation
  - If we have a nonblocking descriptor that we want to read from and we call select with a **timeout** value of 5 seconds, select will block for up to 5 seconds

Detail Spec

- You can download the files from:
  - 課程網頁 -> Assignment
- You can compile the program using:
  - `$> gcc -Wall sp_hw1_httpd.c`
- You can start up server using:
  - `$>./a.out 8000 log_file`
  - 3000 < port < 30000

How to test

- Use Explore (Not IE, please)



- Use telnet (2 \n\n after the GET)

```
austintodo@ir:~/public_html/sp_hw1$ telnet localhost 8000
Trying 127.0.0.1...
Connected to localhost.
Escape character is '^]'.
GET /sp_hw1_httpd.c HTTP/1.1
```

How can I get full score (15 pts)

- Support multiple requests for small files (4 pts)
- Support multiple requests for large files (6 pts)

- Output runtime information to the log_file by printf and dup/dup2. (1 pt)
- Report (4 pts): Problem description, solution, discussions, ...
- Hand in makefile, source code & report