CSE 421/521 - Operating Systems
Fall 2011

LECTURE - VII

PROJECT - I DISCUSSION

Tevfik Koşar

University at Buffalo
September 20th, 2011

An HTTP Request

- <command> <argument> <HTTP version>
- <optional arguments>
- <blank line>

- GET /index.html HTTP/1.0

Server Response

- <HTTP version> <status code> <status message>
- <additional information>
- <a blank line>
- <content>

- HTTP/1.1 200 OK
  Date: Thu, 06 Nov 2008 18:27:13 GMT
  Server: Apache
  Content-length: ……

  <HTML><HEAD><BODY> ….

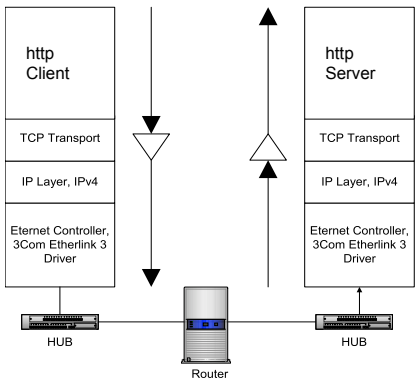
Example

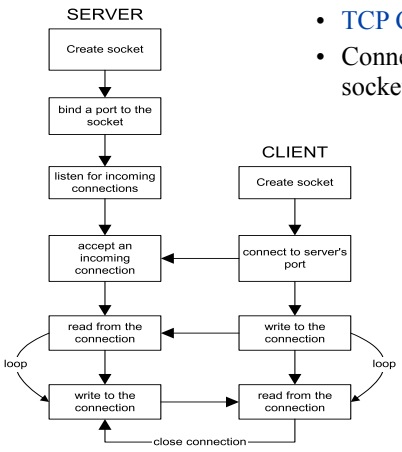
```
$ telnet www.cnn.com 80
Trying 64.236.90.21...
Connected to www.cnn.com.
Escape character is '^]'.
GET /index.html HTTP/1.0

HTTP/1.1 200 OK
Date: Thu, 06 Nov 2008 18:27:13 GMT
Server: Apache
Accept-Ranges: bytes
Cache-Control: max-age=60, private
Expires: Thu, 06 Nov 2008 18:28:14 GMT
Content-Type: text/html
Vary: Accept-Encoding,User-Agent
Connection: close

<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN""http://
www.w3.org/TR/html4/loose.dtd"><html lang="en"><head><title>CNN.com -4
```

Basics of a Server (Web, FTP ..etc)

1. Listen to a Network port
2. Interpret incoming messages (requests)
3. Serve requests
   a. Read requested files
   b. Send them over network
4. Run consistently in the background *(daemon process)*

Network Communication





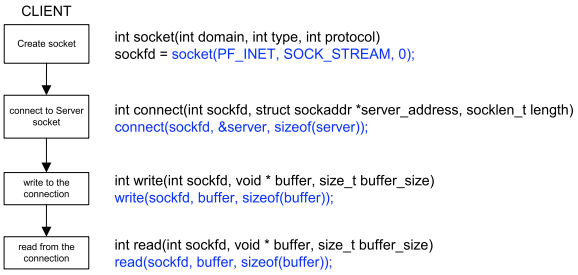
- TCP Client-Server view
- Connection-oriented socket connections

Sockets

- A **Socket** is comprised of:
  - a 32-bit node address (IP address)
  - a 16-bit port number (like 7, 21, 13242)
- Example: 192.168.31.52:1051
  - The 192.168.31.52 host address is in "IPv4 dotted-quad" format, and is a decimal representation of the hex network address 0xc0a81f34
- First developed at UC-Berkeley in 1983, Berkeley Socket API part of BSD 4.2

Server Side Socket Details

Client Side Socket Details



Simple Web Server

Logic of a Web Server

- 1. Setup the server
  - *socket, bind, listen*
- 2. Accept a connection
  - *accept, fdopen*
- 3. Read a request
  - *fread*
- 4. Handle the request
  - a. directory --> **list it**
  - b. regular file --> **cat the file**
  - c. not exist --> **error message**
- 5. Send a reply
  - *fwrite*

1. Setup the Server

```
int init_socket(int portnum)
{   ...
    gethostname( hostname , 256 );          /* where am I ?         */
    hp = gethostbyname( hostname );         /* get info about host  */
    ...
    bzero( (void *)&saddr, sizeof(saddr) ); /* zero struct & fill host addr*/
    bcopy( (void *)hp->h_addr, (void *)&saddr.sin_addr, hp->h_length);
    saddr.sin_family = AF_INET ;            /* fill in socket type  */
    saddr.sin_port = htons(portnum);        /* fill in socket port  */

    sock_id = socket( AF_INET, SOCK_STREAM, 0 );    /* get a socket */
    ...
    rv = setsockopt(sock_id, SOL_SOCKET, SO_REUSEADDR, &on, sizeof(on));
    ...
    bind(sock_id, (struct sockaddr *) &saddr, sizeof(saddr));
    ...

    listen(sock_id, 1) != 0 );
    ...
    return sock_id;
 }
```


2. Accept Connections

3. Read Requests

```
int main(int ac, char *av[])
{
   ...
   sock = init_socket(portnum);
   ...
   /* main loop here */
   while(1){
        /* take a call and buffer it */
        fd    = accept( sock, NULL, NULL );
        ...
        fpin  = fdopen(fd, "r" );
        fpout = fdopen(fd, "w" );

        /* read request */
        fgets(request,BUFSIZ,fpin);
        ...
        while( fgets(buf,BUFSIZ,fp) != NULL && strcmp(buf,"\r\n") != 0 ) ;

       /* do what client asks */
        process_rq(request, fpout);
        ...
        fclose(fpin);
        fclose(fpout);
   }
   return 0;
   /* never end */
  }
```



```
   void process_rq( char *rq, FILE *fp)
{
    ...

    /* create a new process and return if not the child */
    if ( fork() != 0 ) return;

    if ( sscanf(rq, "%s%s", cmd, arg) != 2 ) return;

    ...

    if ( strcmp(cmd,"GET") == 0 )
    {
     if ( not_exist( item ) )
         do_404(item, fp );
     else if ( isadir( item ) )
         do_ls( item, fp );
     else
         do_cat( item, fp );
    }
    ...
    exit(0);
}
```


4.b Cat File

```
   void do_cat(char *f, FILE *fpsock)
{
    char *extension = file_type(f);
    char *content = "text/plain";
    FILE *fpfile;
    int c;

    if ( strcmp(extension,"html") == 0 )
        content = "text/html";
    else if ( strcmp(extension, "gif") == 0 )
        content = "image/gif";
    else if ( strcmp(extension, "jpeg") == 0 )
        content = "image/jpeg";

    fpfile = fopen( f , "r");
    if ( fpfile != NULL )
    {

        fprintf(fpsock, "HTTP/1.0 200 OK\r\n");
        fprintf(fpsock, "Content-type: %s\r\n", content );
        fprintf(fpsock, "\r\n");
        while( (c = getc(fpfile) ) != EOF )
            putc(c, fpsock);
        fclose(fpfile);
    }
}
```


Acknowledgments

- Advanced Programming in the Unix Environment by R. Stevens
- The C Programming Language by B. Kernighan and D. Ritchie
- Understanding Unix/Linux Programming by B. Molay
- Lecture notes from B. Molay (Harvard), T. Kuo (UT-Austin), G. Pierre (Vrije), M. Matthews (SC), B. Knicki (WPI), M. Shacklette (UChicago), J. Kim (KAIST), A. Dix (Hiraeth), and J. Schaumann (SIT).