CS 283 : Systems Programming
Lab3: HTTP Client-Server

yl3385
Yena Lee

Description

Part A

Using network sockets, write a C program called client that receives three command-line arguments in the form: client host port file

and sends a request to a web server. The command-line arguments are

host : Represents the web server to connect to

port : Represents the port number where a request is sent. Normally an HTTP request is sent over port 80, but this format allows for custom ports

file : Represents the file requested from the web server.

Your program should create a client socket that connects to the server indicated by host,

and send the following: GET /index.html HTTP/1.1\r\n Host: www.google.com\r\n \r\n

Then, your program should read the entire result and display it on the screen.

<Client program>

server1.c | client1.c | test.c

```c
/*
* fileclient.c - An echo client
*/
#include "csapp.h"

int main(int argc, char **argv)
{
  int clientfd, port, i, m, n, rc;
  char *host, *file, *buf;
  char *hostname, *filename;
  rio_t rio;
  struct hostent *hp;
  struct sockaddr_in serveraddr;

  if (argc != 4)
  {
    fprintf(stderr, "usage: %s <host> <port> <file>\n", argv[0]);
    exit(0);
  }
  host = argv[1];
  port = atoi(argv[2]);
  file = argv[3];

  clientfd = Open_clientfd(host, port);

  while(1){
    //rc = connect(clientfd, (SA *) &serveraddr, sizeof(serveraddr));
    if (rc<0) {
        perror("Connection failed");
        exit(1);
    }

    Rio_readinitb(&rio, clientfd);
    buf = strcat(file," ");
    buf = strcat(buf, host);

    printf("Send the message\n");

    n = write(clientfd,buf,strlen(buf));
    return -1;
    }
  Close(clientfd);
  exit(0);

}
```

1) receives three command-line arguments which is host, port, file.

```c
  if (argc != 4)
  {
    fprintf(stderr, "usage: %s <host> <port> <file>\n", argv[0]);
    exit(0);
  }
  host = argv[1];
  port = atoi(argv[2]);
  file = argv[3];
```

2) sends a request to a web server

```c
clientfd = Open_clientfd(host, port);
```

- create a client socket that connects to the server

```c
int open_clientfd(char *hostname, int port)
{
    int clientfd;
    struct hostent *hp;
    struct sockaddr_in serveraddr;

    if ((clientfd = socket(AF_INET, SOCK_STREAM, 0)) < 0)
    return -1; /* check errno for cause of error */

    /* Fill in the server's IP address and port */
    if ((hp = gethostbyname(hostname)) == NULL)
    return -2; /* check h_errno for cause of error */
    bzero((char *) &serveraddr, sizeof(serveraddr));
    serveraddr.sin_family = AF_INET;
    bcopy((char *)hp->h_addr_list[0],
      (char *)&serveraddr.sin_addr.s_addr, hp->h_length);
    serveraddr.sin_port = htons(port);

    /* Establish a connection with the server */
    if (connect(clientfd, (SA *) &serveraddr, sizeof(serveraddr)) < 0)
    return -1;
    return clientfd;
}
/* $end open_clientfd */
```

Part B

Write a second program that accepts a port number as a command line argument, and starts an HTTP server. This server should constantly accept() connections, read requests of the form

GET /path HTTP/1.1\r\n\r\n

read the file indicated by /path, and send it over the “connect” file descriptor returned by the call to accept().

<Server program>

server1.c client1.c test.c

```c
#include "csapp.h"

void echofile(int connfd);

int main(int argc, char **argv)
 {
  int listenfd, connfd, port, clientlen;
  struct sockaddr_in clientaddr;
  struct hostent *hp;
  char *haddrp;
  if (argc != 2)
   {
    fprintf(stderr, "usage: %s <port>\n", argv[0]);
    exit(0);
   }
  port = atoi(argv[1]);

  listenfd = Open_listenfd(port);
  printf("Server Listening : %d\n",port);

  while(1)
  {
    connfd = Accept(listenfd, (SA *) &clientaddr, &clientlen);
    if (connfd < 0) {
      perror("Connection failed");
      exit(1);
    }

    echofile(connfd);
    Close(connfd);
    exit(0);

  }


 }
```

1) accepts a port number as a command line argument

```c
if (argc != 2)
 {
  fprintf(stderr, "usage: %s <port>\n", argv[0]);
  exit(0);
 }
port = atoi(argv[1]);
```

2) create a socket descriptor and start server

```c
listenfd = Open_listenfd(port);
printf("Server Listening : %d\n",port);
```

```c
int open_listenfd(int port)
{
    int listenfd, optval=1;
    struct sockaddr_in serveraddr;

    /* Create a socket descriptor */
    if ((listenfd = socket(AF_INET, SOCK_STREAM, 0)) < 0)
    return -1;

    /* Eliminates "Address already in use" error from bind. */
    if (setsockopt(listenfd, SOL_SOCKET, SO_REUSEADDR,
          (const void *)&optval , sizeof(int)) < 0)
    return -1;

    /* Listenfd will be an endpoint for all requests to port
       on any IP address for this host */
    bzero((char *) &serveraddr, sizeof(serveraddr));
    serveraddr.sin_family = AF_INET;
    serveraddr.sin_addr.s_addr = htonl(INADDR_ANY);
    serveraddr.sin_port = htons((unsigned short)port);
    if (bind(listenfd, (SA *)&serveraddr, sizeof(serveraddr)) < 0)
    return -1;

    /* Make it a listening socket ready to accept connection requests
    if (listen(listenfd, LISTENQ) < 0)
    return -1;
    return listenfd;
}
/* $end open_listenfd */
```

3) echofile : read the request message until client closes connection

```c
#include "csapp.h"

void echofile(int connfd)
 {
  size_t n;

  char name[MAXLINE];
  char* hostname;
  char* filename;

  rio_t rio;

  Rio_readinitb(&rio, connfd);

   bzero(name,MAXLINE);

   n = read(connfd,name,MAXLINE);


   filename = strtok(name," ");
   hostname = strtok(NULL,"");

    printf("Here is the message\n");
    printf("GET /");
    printf("%s",filename);
    printf(" HTTP/1.1\\r\\n");
    printf("\n");
    printf("Host: ");
    printf("%s",hostname);
    printf("\\r\\n\n\\r\\n\n");
```

<Programming Environment>

```
server: server1.c csapp.o echo1.o
        $(CC) -o server server1.c echo1.o csapp.o $(LDFLAGS) $(LDLIBS)
client: client1.c csapp.o
        $(CC) -o client client1.c csapp.o $(LDFLAGS) $(LDLIBS)
```

how to run/test the program

prepare two terminal -> server terminal : $ make server / client terminal : $ make client

./server portnum

./client host port file

<Test Result>

Case #1

host : localhost /port : 88 / file : index.html

(server)

```
(base) n3-22-73:networks yenalee$ ./server 88
Server Listening : 88
Here is the message
GET /index.html HTTP/1.1\r\n
Host: localhost\r\n
\r\n
```

(client)

```
(base) n3-22-73:networks yenalee$ make client
gcc -o client client1.c csapp.o  -lpthread
(base) n3-22-73:networks yenalee$ ./client localhost 88 index.html
Request the message
```

Case #2

host : 10.250.22.73 /port : 100 / file : abc.txt

(server)

```
(base) n3-22-73:networks yenalee$ ./server 100
Server Listening : 100
Here is the message
GET /abc.txt HTTP/1.1\r\n
Host: 10.250.22.73\r\n
\r\n
(base) n3-22-73:networks yenalee$
```

(client)

```
(base) n3-22-73:networks yenalee$ ./client 10.250.22.73 100 abc.txt
Request the message
```