Operating Systems

Lab 03：Socket Programming, IPC

Goals:

Learn to work with Socket to perform inter-process communication.

Policies:

It should go without saying that all the work that you will turn in for this lab will be yours. Do not surf the web to get inspiration for this assignment, do not include code that was not written by you. You should try your best to debug your code on your own, but it's fine to get help from a colleague as long as that means getting assistance to identify the problem and doesn't go as far as receiving source code to fix it (in writing or orally).

Contents:

1. What is socket?

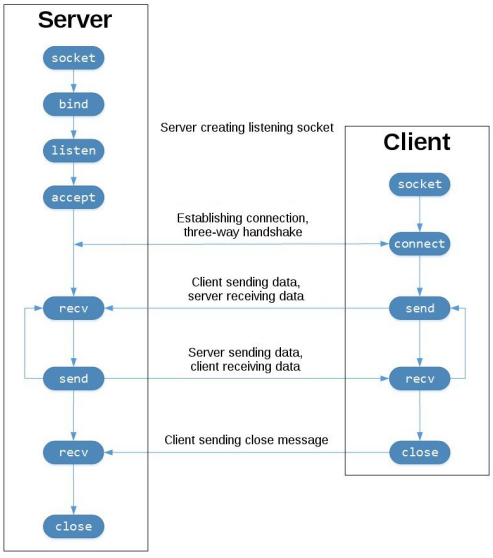
A socket is identified by: an IP + a TCP (or UDP) port – a unique number to differentiate a network service (or process) on a host.

Two processes (client and server) establish a connection: the server has a socket, and the client has a socket. (socket_server, socket_client) uniquely identifies a connection.

2. What is socket programming?

Socket programming is a way of connecting two processes on a network to communicate with each other. One socket listens on a particular port at an IP, while other socket reaches out to the other to form a connection. Server forms the listener socket while client reaches out to the server.

3. State diagram for server and client model



Stages for server

1. **Socket creation**

```
int sockfd = socket(domain, type, protocol)
```

sockfd: socket descriptor, an integer (like a file-handle)

domain: communication domain, an integer, e.g., AF_INET (IPv4 protocol), AF_INET6 (IPv6 protocol) is used for communication.

type: communication type:

SOCK_STREAM: TCP (reliable, connection oriented)

SOCK_DGRAM: UDP (unreliable, connectionless)

protocol: protocol value for Internet Protocol (IP), which is 0.

2. **Bind:**

```
int bind(int sockfd, const struct sockaddr *addr,
                        socklen_t addrlen);
```

After creation of the socket, bind function binds the socket to the IP address and port number specified in **addr**.

Function of Bind: socket + IP + Port.

If the IP address is IPv4: IPv4 is represented by structure **sockaddr_in**, including 16-bits port number and 32-bits IP address

If the IP address is IPv6: IPv6 is represented by structure **sockaddr_in6**, including 16-bits port number and 128-bits IP address

sockaddr_in:

```
struct sockaddr_in{
      sa_family_t sin_family; //Address Family: AF_INET, or AF_INET6
      uint16_t sin_port; //16-bits port
      struct in_addr sin_addr; //32-bits IP address
      char sin_zero[8]; //generally not used, filled with 0
};
```

```
//Create socket:

int serv_sock = socket(AF_INET, SOCK_STREAM, 0);

//Create sockaddr_in structure variable

struct sockaddr_in serv_addr;

memset(&serv_addr, 0, sizeof(serv_addr)); //ach byte is filled with zeros

serv_addr.sin_family = AF_INET; // IPv4 is used

serv_addr.sin_addr.s_addr = inet_addr("127.0.0.1"); //IP address of server

serv_addr.sin_port = htons(1234); //port number

bind(serv_sock, (struct sockaddr*)&serv_addr, sizeof(serv_addr));
```

3. Listen:

```
int listen(int sockfd, int backlog);
```

It puts the server socket in a passive mode, where it waits for the client to approach the server to make a connection. The backlog, defines the maximum length to which the queue of pending connections for sockfd may grow. If a connection request arrives when the queue is full, the client may receive an error with an indication of ECONNREFUSED.

4. Accept:

```
int new_socket= accept(int sockfd, struct sockaddr *addr,
socklen_t *addrlen);
```

It extracts the first connection request on the queue of pending connections for the listening socket, sockfd, creates a new connected socket, and returns a new file descriptor referring to that socket. At this point, connection is established between client and server, and they are ready to transfer data.

Stages for Client

1. Connect:

```
int connect(int sockfd, const struct sockaddr *addr,
                        socklen_t addrlen);
```

The connect() system call connects the socket referred to by the file descriptor sockfd to the address specified by addr. Server's address and port is specified in addr.

Write:

```
ssize_t write(int fd, const void *buf, size_t count);
```

write() writes up to count bytes to the file referenced by the file descriptor fd from the buffer starting at buf

On success, the number of bytes written are returned (zero indicates nothing was written). On error, -1 is returned, and errno is set appropriately.

Read:

```
ssize_t read(int fd, void *buf, size_t count);
```

read() attempts to read up to count bytes from file descriptor fd into the buffer starting at buf.

On success, the number of bytes read is returned (zero indicates end of file)

What should you do:

The client program has been given: **client.c**.

Task 1: please write the server program, named as: **server.c**

Task 2:

However, the above implementation can only perform single-process communication, that is to say, each time can only make a client connect to server for data communication, which

obviously does not meet the basic requirements of the server. We can modify the code on the server side by creating a child process each time after a client connection is successfully accepted by the server, and letting the child process handle the read and write data while the parent process continues to listen and accept.

Results of Task 1:

```
zaobohe@ubuntu:~/Desktop$ ./server
Socket successfully created..
Socket successfully binded..
Server listening..
server acccept the client...
From client: Hi
         To client : Hello
From client: exit
         To client : exit
Server Exit...
zaobohe@ubuntu:~/Desktop$
```

```
zaobohe@ubuntu:~/Desktop$ ./client
Socket successfully created..
connected to the server..
Enter the string : Hi
From Server : Hello
Enter the string : exit
From Server : exit
Client Exit...
zaobohe@ubuntu:~/Desktop$
```

2. What should you submit:

1) **Please submit a zip file which includes the C code of server.c (for both task 1 and task 2):**
2) **Your zip file should be named as: "姓名_lab3"**
3) **Please submit your zip file by sending email to me as attachment: my email address is: hzb564@jnu.edu.cn**