EE4204 Computer Networks I (Part 1)

Socket Programming Assignment
Lab Course Material

Lecturer : Assoc. Prof. Mohan Gurusamy, gmohan@nus.edu.sg

I. C programming in Linux

Editing:

You can use any editor in Linux to write your source code. Emacs and vi are possible choices of editors.

Compilation:

You can use the command-line compiler gcc for compilation. Suppose we have a simple source file called hello.c, and when the directory of hello.c is the current directory, you type:

gcc –g –Wall -o hello hello.c

This command-line tells the gcc compiler to read, compile and link the source file hello.c and write the executable code into the output file hello.

Like most Linux program, gcc has a number of command-line options (which starts with a "-"). The options above are used the most often:

-g	enables debugging
-Wall	turns on warnings
-o file	Place output in file *file*

Execution:

Type *./hello* under the directory of the executable file hello and there runs the program.

II. Socket Programming

Sockets are programming channels for processes to exchange data locally and across the network. A socket is an end point of a two-way communication channel.

Generally, an application program performs the following functions in order:

- Create and name sockets
- Make or accept socket connections
- Send and/or receive data
- Shut down socket operations.

Header files

The following header files contain data definitions, structures, constants, macros, and options used by socket subroutines. An application program should include the appropriate header files to make use of structures or other information a particular subroutine requires.

<sys/socket.h>	Contains data definitions and socket structures
<netinet/in.h>	Defines Internet constants and structures
<netdb.h>	Contains data defintions for socket subroutines
<sys/types.h>	Contain data type definitions
<unistd.h>	Contains standard symbolic constants and types
<netdb.h>	Contains data definitions for socket subroutines
<sys/errno.h>	Defines socket error codes for network communication errors.

Functions:

The following functions might be useful for your programming. A very brief introduction is given below. The students are strongly encouraged to turn to relevant documents for detailed information. "***man*** *command*" is a good method to get detailed help information for a command or function.

- *struct hostent* gethostbyname(const char * hostname);*

  This function converts a host name into a digital name and store it in a struct hostent variable.

- *int socket(int family, int type, int protocol);*

  This function creates a socket.

- *int bind (int sockfd, const struct sockaddr* myaddr, socklen_t taddrlen);*

This function allocates a local socket number. However, it is not necessary to call it in any cases. Call connect( ) or listen( ) after calling socket( ), the socket number will be allocated automatically, which is the usual way. bind( ) is usually used when the process intends to use a specific network address and port number.

- *int connect (int sockfd, const struct sockaddr* serv_addr, int addrlen);*

It is called by the client to connect to the server.

- *int listen (int sockfd, int backlog);*

It is used by the server side to wait for any connection request from the client side.

- *int accept (int sockfd, struct sockaddr* cliaddr, socklen_t* addrlen);*

This function synchronously extracts the first pending connection request from the connection request queue of the listening socket and then creates and returns a new socket.

- *pid_t fork( )*

It generates a new child process. Often used by the server side to generate a new process to handle a new connection request.

- *int send(int scokfd, char* buf, int len, int flags);*

Sends data in a connected socket.

- *int recv(int scokfd, char* buf, int len, int flags);*

Receives data in a connected socket.

- *int sendto((int scokfd, void * mes, int len, int flags, const struct sockaddr* toaddr, int * addrlen);*

Sends data to a specific socket.

- *int recvfrom(int scokfd, void * buff, int len, int flags, const struct sockaddr* fromaddr, int* addrlen);*

Receives data from a specific socket.

- *int close (sockfd);* /* close the socket */

- while(waitpid(-1,NULL,WNOHANG) > 0); /* clean up child processes*/

III. Brief Description of Example Programs

To help programming, the solutions for the first five problems are given, while another one is left for the students to develop and implement solutions. Brief explanation on the example programs are given below.

1. Ex1*: simple examples*

The programs show simple processes between a client and a server. The client will read a string from the screen input and send it to the server, and the server will receive it and print it on the screen. The server will exit when 'q' is inputted at the client end.

Tcp_ser.c and tcp_client.c are examples for a TCP-based process, while udp_ser.c and udp_client.c examples for a UDP-based process. ***_ser.c is the program for server side while ***_client.c is for the client side. The file naming except the suffix will remain in the following sections. For example, client1.c will mean the client program for Ex1.

2. **Ex2:** *sending large message in a single data-unit*

The example is to show how to transmit a large message using TCP. Here the large message is read from a file, which has nearly 50000 bytes in the test (if larger size is desired, the MAXLEN in *"headsock.h"* should be also modified). The file name is *"myfile.txt"*. The client end sends the entire message to the server in one data-unit (let us call it a packet).

TCP ensures that the entire message is received by the receiver correctly and in order with no errors. In the program, the function *"recv"* is called repeatedly in the serve side until all the data has been received. The received data is stored in file *"myTCPreceive.txt"*.

In this example, after the receiver received all the data, it will send back an acknowledgement to the sender and then the server can compute the message transfer time. Two packet structures, *"pack_so"* and "*ack_so*", are defined for data packets and acknowledgement information respectively.

3. Ex3: *sending a large message in small packets using TCP*

The example is to show how to send a large file using small packets. The large file to be sent is "*myfile.txt*", and the received data is stored in "*myTCPreceive.txt*".

The packet size is fixed at 100 bytes per packets. The sender does not stop its sending until all the data has been sent out. The program doesn't use the packet structure as last section. Since the transmitted file is an ASCII file, a '\0' is appended at the end of the file to indicate the end. The receiver will check the last byte in the packet it received to see whether the transmission has ended. If true, it sends back an acknowledgement to the sender (i.e. client). The sender will calculate the transfer time after receiving the acknowledgement. (In standard packet structure, some information to aid error /flow control is included. But in this program, there is no feedback information to the sender, we simply ignore the above information, and the packet contains the data to transmit only.)

IV. Some Tips on Linux Operation

1. **In the computer cluster, ONE user id can only login ONE machine at one time (Please check)**.
2. To return to the home directory, use "***cd ~***" or "***cd***". To go to the parent directory, use "***cd ..***".
3. To run the examples 1-3, you should keep the server and client program running simultaneously. They may be in two terminal windows in one computer, or they are in different computer. To run them, execute the server program first and keep it running, then execute the client program.

   When executing the client, please remember that there is a parameter in the command-line, which is the server's **ip address** or **host name.** Lack of this parameter will cause errors.
4. The host name of each machine is ***lx**.cadcam.nus.edu.sg***, (lx** differs from machines. It can be found on the tag attached to the computer).
5. The ip address of each machine can be found by keying the command ***"/sbin/ifconfig***".
6. You can test the communication between a client and a server within one computer, instead of two. Actually, 127.0.0.1 (or the string "localhost") stands for the address of the local host in TCP/IP. Therefore, to test, you can the run both the client and server programs in a single computer and let the client connect to the server at the address of 127.0.0.1 (or "localhost"). To measure the throughput in the network environment, client and server should be run on two computers.
7. To unzip a file, use the following command:
   "***unzip ***.zip"***
   where ***.zip is the file to be unzipped.

Note: You can run the example programs on a single computer (for example, your own laptop computer). If you do not have Linux OS, please follow the instructions in "**Install Ubuntu on Oracle VirtualBox"** posted as a separate document.

References

Introductory material on socket programming can be found in most of the text books on Computer Networks and Linux Programming. For details you can refer to books on TCP/UDP programming. You can also refer online resources. For the better understanding of the example programs, you can refer the ppt slides (EE4204-socket-programming…" posted.