SOCKET EXERCISES

1. Title: Introduction to Sockets
   Project Tasks:
   a. Create three programs, two of which are clients to a single server. Client1 will send a character to the server process. The server will decrement the letter to the next letter in the alphabet and send the result to client2. Client2 prints the letter it receives and then all the processes terminate. Compile and run this exercise in both the UNIX and the Internet domains.

   b. Follow the same procedure as in part a except that the data type of the message should be integer and the server should decrement the integer before transmitting it to client2.

   c. Next write a socket program to enable client1 to send a float value to the server. The server process should increase the value of the number it receives by a power of 1.5. The server should print both the value it receives and the value that it sends. Client2 should print the value it receives from the server.

   d. Send a C structure that includes data of type character, integer and float from client1 to the server. The server should change the values so that client2 receives a structure with entirely different data. It is not permitted that the data should be converted to any other data type before transmission. Do this exercise in both the UNIX and the Internet domains.

   e. Repeat part 4 except that the machine architectures of the clients and server should be different. You are not permitted to convert the structures to stings before transmission. This is an exercise in the use of big endian and little endian representations of data. Explain what changes had to be made in order to accommodate the differences in data representations.

2. Title: Dining Philosophers Problem without shared memory

   Project Tasks:
   1. create five child processes for philosophers, and five child processes for each of the chopsticks

   2. initiate a pair of sockets for each connection to a server
   3. avoid possible deadlock
   4. close and unlink the sockets.

   Description: Use the fork( ) system call to create a child process. The parent should do nothing but create ten child processes. The requests for granting and releasing chopsticks should be done by each child process representing a philosopher through communication with the two processes representing the philosopher's left and right

hand chopsticks.

TCP/IP should be used for communication between philosophers and chopstick processes. The datagram type of sockets should be employed between the parent and child processes including both the philosopher and chopstick processes.

Use write( ) and read( ) system calls with the connection-oriented communication.

A solution that prevents deadlock or starvation should be implemented. For example the RightLeftDP algorithm (Lynch, 1996) is a suitable algorithm.

Five child processes are to be created for the five philosophers. The philosophers change their states among "thinking", "waiting", and "eating" until all philosophers fulfill their eating requirements (60 seconds total eating time).

Print the status of each philosopher whenever there is a change in the status.

3. Title: Performance Measurements

Project Tasks: Write a program using sockets to send a 1-byte message from a client to a server, and then have the server return the message to the client. This process should be repeated in a loop for 1000 iterations, and the average round trip latency should be calculated.

Follow the same procedure as above for messages of size 1 KB, 2 KB, 4KB, 8KB, and 16 KB. Determine the effective throughput for each message size.

4. Title: Multiplexing among multiple files or sockets
   Project Tasks: Using the "select" system call, write a program that checks whether a connection request has been made before calling the "accept" system call.

5. Title: Remote Login Client-Server Program
   Project Tasks:
   a. Establish a virtual circuit to the login server on the host given on the command line
   b. Use the getservbyname library routine to obtain the port number on the server
   c. Use the gethostbyname library routine to obtain the IP address of the remote host
   d. Create a daemon process that listens for login requests on the server that forks a separate

process to handle each client request

6. Title: Remote Arithmetic with Arrays and Error Detection
   Project Tasks:
   a. Send datagrams with two arrays of integers to a server
   b. The server should do a CRC check of the data
   c. The server sums the elements of each array and puts the sum in a third array that is returned to the client.
   d. The arrays should have at least 500 elements each

   Description:
   If the server discovers that the arrays have erroneous then the server does not reply. A timeout period should be established by the client such that retransmission occurs after the period expires. This will handle the cases of lost messages as well as of errors.

7. Title: Inventory Database
   Project Tasks:
   a. Create a database of parts for an automobile that includes part number, part name, part price, part quantity, account number of user, and a part description.
   b. Write procedures on the server that search for a part, obtain a part name from a part number, give the quantity of parts available, order a part, and obtain a list of subpart numbers
   c. Create a set of commands on the client that the server can interpret in order answer client inquiries.
8. Title: Remote Copying of Files
   Project Tasks:
   a. Copy a file of ASCII data to the server from a client.
   b. The server should check the file for errors with the CRC polynomial method.
   c. After the check, data should be sent to a second client

9. Title: Three-Way Talk: An Extension to UNIX Two-Way Talk

Description:

Talk is a visual form of write. Using sockets, a two way connection is set up between two people. With the aid of cursors, the screen is split into two windows, and each user's text is added to the window, one character at a time. Extend this program so that one user can talk to two other users instead of one. Usually, talkd is a server which listens at UDP port 517. The actual conversation takes place on a TCP connection that is established by negotiation. The Internet services daemon, inetd, handles the execution of talkd.

Your program needs to consider the following issues:

1. When a two-way talk is in progress, one of the participants may want to initiate a second talk or may receive a talk request from another user.

2. A participant involved in a two-way talk should be able to escape from talking and issue another talk request.

3. When a second talk connection is established, the screen of the participant who issued the second talk request should be split into three windows.

4. To make three-way talk realistic, even the screens of the other two participants must also be divided into three windows.

5. When one of the participants in a three-way talk hangs up, the remaining participants should be able to continue talking with their screens restored to two windows.

10. Title: A 911 Dispatcher

Description: This program will provide customer service by a 911 dispatcher. The dispatcher will have a database of fire, police, and ambulance crews that can respond to emergencies at remote sites. When an emergency call is received, the dispatcher evaluates it and requisitions personnel from the database. If insufficient personnel are available, the dispatcher will gather available people from other sites and, using TCP, transmit them to the customer location. When people have completed their emergency function, then they are returned to the dispatcher's available database by an agent at the site. The program should be written using C++ and network objects including agents. After an emergency is handled remotely, the remote host decides whether to retain some of the emergency personnel for longer than usual if the situation warrants it.

Implementation

1. Class Inheritance.

Since the code is written in object oriented C++, the project design requires a class structure. There are some communication functions that are common to the server, agent and client. These functions are placed in a base class that has Server, Agent, and Client sub-classes.

2. Message Broadcast Communication.

A message is broadcast so that agents and dispatcher receive it from any sender. The agent always asks for personnel from the server. If the server does not have personnel available, the agent sends the request to other agents already on duty. Based on the "Message To" field in the message structure, the agent or the dispatcher takes action.

3. Message Structure.

A message consists of six fields. The first field is the destination, the second field is the sender, the third field defines the type of message, while the last three fields depend on the type of the message.

to_id	from_id	type	f 1	f 2	f 3

4. Message Types.

Message Type	Code	Action	Location
sendworker	3	dealsendworker()	911agent.cpp
sendid	7	dealsendid()	911agent.cpp
requestworker	2	dealrequest()	911agent.cpp
jobcompletion	4	dealjobcompletion()	911agent.cpp
stringmesg	5	dealstringmesg()	911agent.cpp
emergcall	1	deal911call()	911center.cpp
requestworker	2	dealrequest()	911center.cpp
requestid	8	dealrequestid()	911.center.cpp
sendworker	3	dealsendworker()	911center.cpp
jobcompletion	4	dealjobcompletion()	911center.cpp
stringmesg	5	dealstringmesg()	911center.cpp

5. Use of UNIX System Calls.

To enable the client process to compile and execute the agent program, we use UNIX system calls. The exec system call is used to run the compiled agent file.

6. Some Major Functions.

In class entity
int setupcommunication(), creates a socket.
int closecommunication(), closes the socket.
int sendmesg(), sends the message structure.
int recvmesg(), receives the message structure.
int sendprogram(), sends the agent file to the client.
int recvprogram(), receives the agent file from the server.

In class mesg
void setempty(), clear all the strings.

In class despatcher
void manageagent(), sort the array duty_agent[].
void deal911call(), create agent id and fork a child to send.
void dealrequest(), handles the availability of personnel.
void dealsendworker(), puts back the returned personnel.

In class agent
void asknewid(), renew the inherited client_id.
void request(), if severe, more worker.

void listening(), wait for the message.
void dealrequest(), translate the request into mesg.
void dealsendworker(), receive the personnel from the server.
void dojob(), fork a child process to do the job.

In class specialagent
void working(), set up communication, get ID, decide if more worker is needed, etc.
void userinterface()

In class client
void whathapppen(), ask for the emergency type.
int writetofile(), write host_id to a file.
int compile(), compile the agent file.
int execute(), run the agent file.

.