Computer Networks Project on Socket Programming.

Topic:

THE HANGMAN

Group members :
Charith Reddy (201201200)
Avinash Pericherla (201201236)

Abstract:

This project is basically implementation of a two player game **hangman** using the concept of single **peer-peer** communication.
Unlike the client-server model, in which the client makes a service request and the server fulfills the request, the peer to peer model allows each node in a peer-to-peer network to function as both a client and a server. Two computers are considered peers if they are communicating with each other and playing similar roles.

Functionalities included in this project:

- Hangman game algorithm – two different interface for EVEN and ODD clients.
- Usage of shared memory to differentiate the ODD and EVEN client and so the game can be played by many such incoming pairs of clients.
- Binary Semaphores in server to handle multiple clients bombarding on the server.

Server's side:

For the server side communication, it opens a public port which can be predefined or given as a command line argument and waits for its clients i.e. players of the game.
When a client (player) connects to the server -

➢ Case_1: There is no player waiting to play, then server takes client's information and waits until another player connects to the server and then pairs them up.

➢ Case_2: There is a player waiting to play , then server gives the information of the waiting player to the new visited player and there-by the players connect to each other and the further communication between them will happen directly without server's interference.

This process keeps on going. Every odd player connecting to the server is considered as a waiting client and even player as client to be connected with the client who is already waiting to play.

- We made the usage of semaphores and shared memory and thus server has the capacity of handling multiple clients and can scale them accordingly.

SERVER CODE:

```
#include <stdio.h>
#include <sys/shm.h>
#include <sys/stat.h>
#include <stdlib.h>

#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/sem.h>

#include <string.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>

#define BACKLOG 10

/* We must define union semun ourselves. */
union semun {
        int val;
        struct semid_ds *buf;
        unsigned short int *array;
        struct seminfo *__buf;
};
/* Initialize a binary semaphore with a value of 1. */
int binary_semaphore_initialize (int semid) {
        union semun argument;
        unsigned short values[1];
        values[0] = 1;
        argument.array = values;
        return semctl (semid, 0, SETALL, argument);
}
int binary_semaphore_wait (int semid) {
        struct sembuf operations[1];
        /* Use the first (and only) semaphore. */
        operations[0].sem_num = 0;
        /* Decrement by 1. */
        operations[0].sem_op = -1;
        /* Permit undo'ing. */
        operations[0].sem_flg = SEM_UNDO;
        return semop (semid, operations, 1);
}
/* Post to a binary semaphore: increment its value by 1.
This returns immediately. */
int binary_semaphore_post (int semid) {
        struct sembuf operations[1];
        /* Use the first (and only) semaphore. */
        operations[0].sem_num = 0;
        /* Increment by 1. */
        operations[0].sem_op = 1;
        /* Permit undo'ing. */
        operations[0].sem_flg = SEM_UNDO;
        return semop (semid, operations, 1);
}
void error(char *msg)
{
        perror(msg);
        exit(1);
}
        //--------------------------------------------------------------------------------
int main(int argc, char* args[]) {
        if (argc!=2) {
                printf("usage: ./s port\n");
                exit(-1);
        }

        int sockfd, newfd, sin_size, n;
        struct sockaddr_in my_addr, their_addr;

        sockfd = socket(PF_INET, SOCK_STREAM, 0);
        my_addr.sin_family = AF_INET;
        my_addr.sin_port = htons(atoi(args[1]));
        my_addr.sin_addr.s_addr = INADDR_ANY;
        memset(my_addr.sin_zero, '\0', sizeof my_addr.sin_zero);

        bind(sockfd, (struct sockaddr *)&my_addr, sizeof my_addr);

        listen(sockfd, BACKLOG);

        //--------------------------------------------------------------------------------
        int segment_id;
        char* shared_memory;
        const int shared_segment_size = 0x6400;
        /* Allocating a shared memory segment.  */
        segment_id = shmget (IPC_PRIVATE, shared_segment_size,
                     IPC_CREAT | IPC_EXCL | S_IRUSR | S_IWUSR);
        /* Attaching the shared memory segment.  */
        shared_memory = (char*) shmat (segment_id, 0, 0);

        int segment_id_2;
        char* shared_index;
        const int shared_segment_size_2 = 0x6400;
        /* Allocating a shared memory segment.  */
        segment_id_2 = shmget (IPC_PRIVATE, shared_segment_size_2,
                     IPC_CREAT | IPC_EXCL | S_IRUSR | S_IWUSR);
        /* Attaching the shared memory segment.  */
        shared_index = (char*) shmat (segment_id_2, 0, 0);
        sprintf(shared_index,"ODD");
        //--------------------------------------------------------------------------------
        int numsems = 1;
        int semaphore1 = semget(IPC_PRIVATE,numsems,0666 | IPC_CREAT);
        binary_semaphore_initialize(semaphore1);
        int semaphore2 = semget(IPC_PRIVATE,numsems,0666 | IPC_CREAT);
```

```
	binary_semaphore_initialize(semaphore2);
	binary_semaphore_wait(semaphore2);
	//------------------------------------------------------------------------------------
	char buffer[256];
	//------------------------------------------------------------------------------------
	while (1) {
		printf("....waiting to connect to a client....\n");
		sin_size = sizeof their_addr;
		newfd = accept(sockfd, (struct sockaddr *)&their_addr, &sin_size);
		if (newfd < 0)
			error("ERROR accepting");
		printf(":::Connection Accepted:::\n");

		if(fork() == 0) {
			binary_semaphore_wait(semaphore1);

			printf("CONNECTED\n");
			close(sockfd);

			n = write(newfd,shared_index,256);
			if (n < 0)
				error("ERROR writing to socket");

		//EVEN clients

			if (!strcmp(shared_index,"EVEN")) {
				printf("---Connected to EVEN_client---\n");
				//reading client_1 info stored in the shared memory
				n = write(newfd,shared_memory,256);
				if (n < 0)
					error("ERROR writing to socket");
				sprintf (shared_memory,"");
				sprintf(shared_index,"ODD");
				printf("/// Done with EVEN_client ///\n\n");
			}
			else if (!strcmp(shared_index,"ODD")){	//ODD clients
				printf("---Connected to ODD_client---\n");
				//writing client_1 info into the shared memory
				bzero(buffer, 256);
				n = read(newfd,buffer,256);
				if (n < 0)
					error("ERROR reading from socket");
				sprintf (shared_memory,"%s|%s",(char
*)inet_ntoa(their_addr.sin_addr),buffer);
				sprintf(shared_index,"EVEN");
				printf("/// Done with ODD_client ///\n\n");
			}
			else {
				printf("XXX---something is wrong---XXX\n");
			}
			binary_semaphore_post(semaphore2);
			exit(-1);
		}
```

```
                else {
                        binary_semaphore_wait(semaphore2);
                        close(newfd);
                        binary_semaphore_post(semaphore1);
                }
        }
}
```

CLIENT'S side:

For the client side communication, it connects to the server with the port and IP Address of the server which is given either as its command line arguments or predefined.

- The client's major role is associated with the game play.
- For instance, if one game is to be run between two players - run the client executable twice - one by one - i.e. first one should be executed and kept running and then the other client is to be executed. Likewise more and more games between two different players can be played using the same server.

Client Code :

```
#include <stdio.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <netdb.h>
#include <string.h>
#include <stdlib.h>

#define BACKLOG 10

void error(char *msg)
{
        perror(msg);
        exit(0);
}

char* player1();
void player2(int oppSockfd, char wordStr[256]);
void player12(int oppSockfd, char wordStr[256]);

int main(int argc, char *argv[])
{
        if (argc != 3) {
                fprintf(stderr,"usage: %s hostname port\n", argv[0]);
                exit(0);
        }

        int sockfd, portno, n;
        char buffer[256];
        struct sockaddr_in serv_addr;
        struct hostent *server;

        portno = atoi(argv[2]);
        sockfd = socket(AF_INET, SOCK_STREAM, 0);
        if (sockfd < 0)
                error("ERROR opening socket");
        server = gethostbyname(argv[1]);
        if (server == NULL) {
                fprintf(stderr,"ERROR, no such server\n");
                exit(0);
        }
        bzero((char *) &serv_addr, sizeof(serv_addr));
        serv_addr.sin_family = AF_INET;
        bcopy((char *)server->h_addr,
     (char *)&serv_addr.sin_addr.s_addr,
     server->h_length);
        serv_addr.sin_port = htons(portno);
        if ( connect(sockfd,(struct sockaddr *)&serv_addr,sizeof(serv_addr)) < 0){
                error("ERROR connecting");
        }
        printf("\n---Connected to the server---\n");

        char oppIndex[256];  // to know the info about odd or even client
        bzero(buffer, 256);
        n = read(sockfd,buffer,256);
        if (n < 0)
                error("ERROR reading from socket");
        bcopy(buffer,oppIndex,256);
        bzero(buffer,256);
        //-----------------------------------------------------------------------------------------------------------------
        if (!strncmp(oppIndex,"ODD",3)){      //ODD clients' module
                printf("Give a port no through which ur opponent can connect with you: ");
                bzero(buffer,256);
                fgets(buffer,255,stdin);
                n = write(sockfd,buffer,strlen(buffer)-1);
                if (n < 0)
                        error("ERROR writing to socket");
                close(sockfd);
                //-------------------END OF SERVER COMMUNICATION----------------------------
                int listeningSockfd, oppSockfd, sin_size, myportno;
                myportno = atoi(buffer);
                struct sockaddr_in my_addr, opp_addr;

                listeningSockfd = socket(PF_INET, SOCK_STREAM, 0);
                my_addr.sin_family = AF_INET;
                my_addr.sin_port = htons(myportno);
                my_addr.sin_addr.s_addr = INADDR_ANY;
                memset(my_addr.sin_zero, '\0', sizeof my_addr.sin_zero);
                bind(sockfd, (struct sockaddr *)&my_addr, sizeof my_addr);

                listen(listeningSockfd, BACKLOG);
                printf("\n---Waiting to connect to a opponent---\n");

                sin_size = sizeof opp_addr;
                oppSockfd = accept(listeningSockfd, (struct sockaddr *)&opp_addr, &sin_size);
                if (oppSockfd < 0)
                        error("ERROR accepting");
                printf("\nCONNECTED to opponent\n\n");
                //----------------CONNECTION ESTABLISHED---------------------------
                while (1) {
                        char* wordStr;
                        wordStr = (char*)malloc(sizeof(char));
                        char temp[256],temp2[256];
                        wordStr = player1(); // duplicate word appended to original word
                        bcopy(wordStr,temp,256);
                        bcopy(wordStr,temp2,256);
                        printf("\n...waiting for the result...\n\n");
                        n = write(oppSockfd,temp,256);
                        if (n < 0)
                                error("ERROR writing to socket");
                        player12(oppSockfd, temp2);
                        //--------------ONE TURN DONE------- Below is for next
                        printf("---Wait till your opponent gives a word to guess---\n");
                        char tempWordStr[256];
                        bzero(buffer,256);
                        n = read(oppSockfd,buffer,256);
                        if (n < 0)
                                error("ERROR reading from socket");
                        bcopy(buffer,tempWordStr,256);
                        player2(oppSockfd, tempWordStr);
                }
        }
        //------------------------------------------------------------------------------------------------------------
        else if (!strncmp(oppIndex,"EVEN",4)) {        //EVEN clients' module
```

```
bzero(buffer,256);
n = read(sockfd,buffer,256);
if (n < 0)
            error("ERROR reading from socket");
close(sockfd);
//--------------------------------------------------------------------------
int i,tempIdx,oppPortno;
char ip_addr[256];
char port_num[256];
for (i=0;i<strlen(buffer);i++) {
            if (buffer[i] == '|') {
                        ip_addr[i]='\0';
                        tempIdx = i+1;
                        break;
            }
            ip_addr[i] = buffer[i];
}
for (i=tempIdx;i<strlen(buffer);i++) {
            port_num[i-tempIdx] = buffer[i];
}
port_num[strlen(buffer)-(strlen(ip_addr)+1)] = '\0';
oppPortno = atoi(port_num);
//--------------------END OF SERVER COMMUNICATION------------------------------
printf("\n---Establishing connection with a player---\n");
int oppSockfd;
struct sockaddr_in opp_addr;
struct hostent *opponent;

oppSockfd = socket(AF_INET, SOCK_STREAM, 0);
if (oppSockfd < 0)
            error("ERROR opening socket");
opponent = gethostbyname(ip_addr);
if (opponent == NULL) {
            fprintf(stderr,"ERROR, no such opponent\n");
            exit(0);
}
bzero((char *) &opp_addr, sizeof(opp_addr));
opp_addr.sin_family = AF_INET;
bcopy((char *)opponent->h_addr,
    (char *)&opp_addr.sin_addr.s_addr,
    opponent->h_length);
opp_addr.sin_port = htons(oppPortno);
if ( connect(oppSockfd,(struct sockaddr *)&opp_addr,sizeof(opp_addr)) < 0){
            error("ERROR connecting");
}
printf("\nCONNECTED to opponent\n\n");
//----------------CONNECTION ESTABLISHED---------------------------
while (1) {
				printf("---Wait till your opponent gives a word to guess---\n");
				char tempWordStr[256];
				bzero(buffer,256);
				n = read(oppSockfd,buffer,256);
				if (n < 0)
					error("ERROR reading from socket");
				bcopy(buffer,tempWordStr,256);
				player2(oppSockfd, tempWordStr);
				//-----ONE TURN DONE---------------------------
				char* wordStr;
				wordStr = (char*)malloc(sizeof(char));
				char temp[256],temp2[256];
				wordStr = player1();
				bcopy(wordStr,temp,256);
				bcopy(wordStr,temp2,256);
				printf("\n...waiting for the result...\n\n");
				n = write(oppSockfd,temp,256);
				if (n < 0)
					error("ERROR writing to socket");
				player12(oppSockfd, temp2);
			}
		}
		else {
			printf("XXX---something is wrong---XXX\n");
		}
		return 0;
}

char* player1() {
		char word[256];
		char wordStr[256];
		char *actualWordStr = (char*)malloc(sizeof(char));
		printf("---Your turn to give a word to your opponent---\n");
		bzero(wordStr,256);
		printf("Give your word: ");
		scanf("%s",word);
		strcat(wordStr,word);
		strcat(wordStr,"|");
		printf("Your word in string format (for ex:- APPLE -> A_P_E): ");
		scanf("%s",word);
		strcat(wordStr,word);
		actualWordStr = &wordStr[0];
// word stored as Networking|N _ t w _ r _ _ n g
		return actualWordStr;
}

void player2(int oppSockfd, char wordStr[256]) {
        char man[7][15]={{' ',' ',' ',' ','_','_','_','\n'},{' ',' ',' ',' ',' ','|','\n'},{' ',' ',' ',' ',' ','0','\n'},{' ',' ',' ',' ',' ','|','\n'},{' ',' ',' ',' ','/','|','\\','\n'},{' ',' ',' ',' ',' ','|','\n'},{' ',' ',' ','_','/',' ','\\','_','\n'}};
//char array man used to print the hanging man
        int i, j, temp, n;
        char actualWord[256],dupWord[256];
        printf("\n\tGo ahead...guess the word........\n");
        for (i=0;i<strlen(wordStr);i++) {
                if (wordStr[i]=='|') {
                        actualWord[i] = '\0';
                        temp = i+1;
                        break;
                }
                actualWord[i] = wordStr[i];
        }
        for (i=temp;i<strlen(wordStr);i++) {
                dupWord[i-temp] = wordStr[i];
        }
        dupWord[strlen(wordStr)-(strlen(actualWord)+1)] = '\0';

// actual word and duplicate word separated
        printf("\n\t\t");
        for (i=0;i<strlen(dupWord);i++) {
                printf("%c ",dupWord[i]);
        }
        printf("\n\n");

        char letter;
        char buffer[256];
        int count=0;
        while (count<7) {
                if (!strcmp(actualWord,dupWord)) {
                        printf("You WON against your opponent!! :)\n\n");
                        break;
                }
                printf("your letter: ");
                letter = getc(stdin);
                if (letter == '\n') letter = getc(stdin);

                buffer[0]=letter;
                buffer[1]='\0';
                n = write(oppSockfd,buffer,256);
                if (n < 0)
                         error("ERROR writing to socket");

                int ch=0;
                for (i=0;i<strlen(actualWord);i++) {
                        if (actualWord[i]==letter && dupWord[i]!=letter) {
                                dupWord[i] = actualWord[i];
                ch=1;
            }
        }
// Updating the duplicate word according to guesses.
        if(ch==0) { count++; }
        printf("\n\t\t");
        for (i=0;i<strlen(dupWord);i++) {
            printf("%c ",dupWord[i]);
        }
        printf("\n\n");
        for (i=0;i<count;i++){
            for (j=0;j<sizeof(man[i]);j++){
                printf("%c",man[i][j]);
            }
        }
        printf("\n\n");
    }
    if (count>=7) {
        printf("You LOST against your opponent! :(\n\n");
    }
}

void player12(int oppSockfd, char wordStr[256]) {
    char man[7][15]={{' ',' ',' ',' ','_','_','_','\n'},{' ',' ',' ',' ',' ','|','\n'},{' ',' ',' ',' ',' ','0','\n'},{' ',' ',' ','
','|','\n'},{' ',' ',' ',' ','/','|','\\','\n'},{' ',' ',' ',' ',' ','|','\n'},{' ',' ',' ','_','/',' ','\\','_','\n'}};
    int i, j, temp;
    char actualWord[256],dupWord[256];
    printf("\n\tYour opponent is guessing the word........\n");
    for (i=0;i<strlen(wordStr);i++) {
        if (wordStr[i]=='|') {
            actualWord[i] = '\0';
            temp = i+1;
            break;
        }
        actualWord[i] = wordStr[i];
    }
    for (i=temp;i<strlen(wordStr);i++) {
        dupWord[i-temp] = wordStr[i];
    }
    dupWord[strlen(wordStr)-(strlen(actualWord)+1)] = '\0';
    printf("\n\t\t");
    for (i=0;i<strlen(dupWord);i++) {
        printf("%c ",dupWord[i]);
    }
    printf("\n\n");

    char letter;
    int count=0, n;
```

```
        char buffer[256];
        while (count<7) {
                if (!strcmp(actualWord,dupWord)) {
                        printf("Your opponent WON against you! :(\n\n");
                        break;
                }

                bzero(buffer,256);
                n = read(oppSockfd,buffer,256);
                if (n < 0)
                         error("ERROR reading from socket");
                letter = buffer[0];
                printf("Letter choosen: %s\n",buffer);

                int ch=0;
                for (i=0;i<strlen(actualWord);i++) {
                        if (actualWord[i]==letter && dupWord[i]!=letter) {
                                dupWord[i] = actualWord[i];
                                ch=1;
                        }
                }
                if(ch==0) { count++; }
                printf("\n\t\t");
                for (i=0;i<strlen(dupWord);i++) {
                        printf("%c ",dupWord[i]);
                }
                printf("\n\n");
                for (i=0;i<count;i++){
                        for (j=0;j<sizeof(man[i]);j++){
                                printf("%c",man[i][j]);
                        }
                }
                printf("\n\n");
        }
        if (count>=7) {
                printf("Your opponent LOST against you!! :)\n\n");
        }
}
```

Future scope & Applications:

- Many such **similar peer to peer games** can be developed which have a great demand and are driving crazy the present generation.
- The present single peer to peer can be modeled into multi p2p file sharing like torrents etc.

- The present peer to peer can have many major applications.

Few examples below:

<u>1) SEARCH ENGINES:</u>

In P2P search i.e. distributed search, each individual connected to the network serves its local index as a source of search. Instead of having a central company and a central server, each participant of the network is a search repository.

<u>2) VIDEO & AUDIO CASTING :</u>

The impact of video and audio web sites on the Internet has been very large over the last couple of years - and will only increase. Therefore there has been talk of moving video streaming to P2P networks, to lessen the load on the Internet. A P2P approach for video streaming would be to hold a copy of a file in different parts of the world and serve it from multiple points to users.

<u>3) E-COMMERCE :</u>

Consumer to Consumer e-commerce is one of the most popular services on the Internet. A centralized trading platform (such as eBay) enables consumers to trade, buy or sell their goods. However in a centralized system, there is always a possibility of a failure - such as the server goes down or is busy. P2P enabled e-commerce can remove the centralized system and so lessen the possibility of failures.
And Mobile P2P applications etc.