Implementation of P2P file transfer protocol

Using Client-Server model

By:
Dhruv Patel – 200901130
Lalit Agarwal – 200901159

Under Guidance of
Prof. Sanjay Chaudhary,
DAIICT.

Definition:

Project is supposed to implement client to client file transfer utility connected to a common server. Here server provides communication among clients.

List of Programs developed: (Refer Appendix A)

server.c

This source file is implementation of server, which is bound to a known port and accepts client connection requests using Socket Programming (using bind(), listen(), and accept() functions). Then, it serves client requests such as keeping list of files of clients which are currently connected to server, sending list of files available on different clients to download, taking download request from one client and informing other client to send file to that client.

client.c

This source code is implementation of client, which is used to connect to server. It provides three options "l" to request list of files available from server on different clients, "d" to send download request to server which will then inform another client which is having file to send file to current client, "q" to quit.

unp.h

This header file contains list of header files use in different programs. It's just to facilitate including different header files.

ip.sh

This shell script is used by client.c program to get ip-address of system on which current client is running. This program uses /sbin/ifconfig command to get ipv4 address and stores it into a text file, which is then read by client.c and sent to server.

Concepts Used:

1. Socket Programming: It’s used extensively throughout the whole project.
2. Parent-child processes: Used to handle multiple clients on the server side.
3. Threading: For every child process running on server side, which is serving single client, there is one another thread is created. When download request comes from a client, this thread informs the sub-process and sub-process informs another client to send file to that client. Thread is also used in client.c to check asynchronous download request coming from server and then sends to requesting client.
4. Shared memory: Shared memory blocks are used to store list of files on different clients available (to store an array of structure “client”), no. of clients (integer variable cnt) and disconnected clients.
5. Semaphores: Semaphore is used in client.c to switch between Uploading file process and normal client facilities (like “l”, “d” and “q”).

APPENDIX A

Filename: - server.c

```
#include<sys/shm.h>
#include"unp.h"


int *downloadstatus;
int *cnt;
struct client *clients;


//client structure to store the list of files shared by each client.
struct client
{
          int id;
         int fileno;
         char ipaddress[20];
          char *name;
          char list[1000][30];
};


//Function to send the list of files shared by all the clients connected to the server.
void sendlist(void *fd)
{
         int confd=*(int *)fd;
         int err;
         char filename[30];
         char handshake[15];
         struct client *list;
         list=clients;
         strcpy(handshake,"CNTSTART");
```

```
        err=write(confd,(const void *)handshake,15);

        err=write(confd,(const void *)cnt,sizeof(int));

        if(err==-1) printf("Error in sending cnt.\n");

        err=write(confd,(const void *)list,*cnt*sizeof(struct client));

        if(err==-1) printf("error in sending list.\n");

        printf("List sent.\n");

}


//Function to check if any download request sent to any of the clients.

void idcheck(void *kuchbhi)

{

        int id=((int *)kuchbhi)[1];

        int confd=((int *)kuchbhi)[0];

        char request='d';

        while(*downloadstatus!=id);

        *downloadstatus=0;

        printf("Client %d accepted the download request.\n\n",id);

        write(confd,(const void *)&request,sizeof(request));

}


//Function to set the status of client to upload the file.

void download(int id,int did)

{



        *downloadstatus=did;

}


//Function to receive the request given by the client and take appropriate action.

int first(void* fd, int id)

{
```

```
int confd=*(int *)fd;
char *c;
int ret=0;
int did;
int array[2];
pthread_t tid;
array[0]=confd;
array[1]=id;
pthread_create(&tid,NULL,idcheck,(void *)array);
while(1)
{
        c=(char *)malloc(2*sizeof(char));
        do
        {
                ret=read(confd,(void *)c,2);
        }while(ret==-1);
if(c[0]=='l')
{
        printf("\nGot List Request from Client %d\n",id);
        sendlist(fd);
}
else if(c[0]=='d')
{
        sleep(1);
        read(confd,(void *)&did,sizeof(int));
        printf("\nGot Download Request from Client %d\n",did);
        write(confd,(void *)clients[did].ipaddress,20*sizeof(char));
        download(id,did);
}
else
```

```
		{
			printf("\nClient %d is exiting.\n",id);
			clients[id-1].fileno=0;
			pthread_exit();
			exit(0);
		}
		free(c);
		}
}

int main(void)
{
		int sockfd,j;
		int cntaddress,listaddress,daddress;
		socklen_t len;
		pid_t pid[1000];
		struct sockaddr_in servaddr,cliaddr;
		//Using Shared Memory.
		cntaddress=shmget(800,sizeof(int),IPC_CREAT | 0755);
		listaddress=shmget(900,15*sizeof(struct client),IPC_CREAT | 0755);
		daddress=shmget(1000,sizeof(int),IPC_CREAT | 0755);
		//cnt variable stores the number of clients connected to the server.
		cnt=(int *)shmat(cntaddress,NULL,0);
		clients=(struct client *)shmat(listaddress,NULL,0);
		downloadstatus=(int *)shmat(daddress,NULL,0);
		*downloadstatus=0;
		sockfd=socket(AF_INET,SOCK_STREAM,IPPROTO_TCP);
		bzero(&servaddr,sizeof(servaddr));
		servaddr.sin_family=AF_INET;
		servaddr.sin_addr.s_addr=htonl(INADDR_ANY);
		servaddr.sin_port=htons(13000);

		bind(sockfd,(SA *) &servaddr,sizeof(servaddr));

		*cnt=0;

		listen(sockfd,LISTENQ);

		 len=sizeof(cliaddr);

		 printf("\nServer Ready\n");

		 while(1)

		 {

				int confd,i;

				int ret=0;

				char ip[17];

				char filename[30];

				printf("Waiting for clients ..\n\n");

				 confd=accept(sockfd,(SA *) &cliaddr,&len);

				printf("Connection Established.\n");

				ret=read(confd,(void *)ip,17);

				if(ret==-1)

					printf("read error.\n");

				strncpy(clients[*cnt].ipaddress,ip,17);

				printf("IP Address of Client %d connected is
%s\n",(*cnt+1),clients[*cnt].ipaddress);

				strcpy(filename,"");

				clients[*cnt].id=*cnt+1;

				clients[*cnt].fileno=0;

				do

				{

					do

					{		ret=read(confd,(void *)filename,30);

					}while(ret==-1);

				printf("Listing the files of Client %d.\n",(*cnt+1));
```

```
                        }while(strcmp(filename,"STARTOFFILE")!=0);

                        do

                        {

                                clients[*cnt].fileno++;

                                do

                                {       ret=read(confd,(void *)filename,30);

                                }while(ret==-1);

                        if(strcmp(filename,"ENDOFFILE")==0) break;

                        strcpy(clients[*cnt].list[clients[*cnt].fileno],filename);

                        }while(1);

                        //Creating child server by using fork() command to handle the client
connected.

                        pid[*cnt]=fork();

                        if(pid[*cnt]==0)

                        {

                                close(sockfd);

                                first((void *)&confd, *cnt);

                        }

                        close(confd);

                        (*cnt)++;

                        printf("No. of Clients Connected: %d\n\n",*cnt);

        }

    return 0;

}
```

Filename: - client.c

```
#include"unp.h"

#include<errno.h>

#include<dirent.h>

#include<semaphore.h>
```

```
int cnt;
sem_t bin_sem;

//client structure to store the list of files shared by the client.
struct client
{
        int id;
        int fileno;
        char ipaddress[20];
        char *name;
        char list[1000][30];
};

//Function to send the list of files shared by this client to the server when it is connected
void sendlist(int *fd)
{
    int myfd= *fd;
    char path[30];
        char s[17];
    DIR *dir;
        FILE *f;
    struct dirent *dt;
    strcpy(path,"/home/");
    strcpy(path,strcat(path,getenv("USER")));
    strcpy(path,strcat(path,"/Sample/"));
    execl("./ip.sh", "ip.sh", NULL, (char *)0);
    f=fopen("./ipaddress","r");
    fscanf(f,"%s",s);
        printf("Client's IP Address is %s\n",s);
        write(myfd,(void *)s,17);
        dir=opendir(path);
        if(dir!=NULL)
        {
                        printf("Directory opened.\n");
                char filename[30];
                        strcpy(filename,"STARTOFFILE");
                        write(myfd,(const void *)filename,30);
                        printf("\nThe files shared by the client are: -\n\n");
                while((dt=readdir(dir))!=NULL)
                {
                        strcpy(filename,dt->d_name);
                                if(strcmp(filename,".")!=0 && strcmp(filename,"..")!=0)
                                {
                                        printf("%s\n",filename);
                                        write(myfd,(const void *)filename,30);
                                }
                }
                        printf("\n");
                strcpy(filename,"ENDOFFILE");
                        write(myfd,(const void *)filename,30);
        }
}

//Function to print the list of files of all clients connected.
void printlist(int *fd)
{
        int myfd=*fd;
        int i;
        char c[2];
```

```
int ret=0;
char handshake[15];
struct client *list;
c[0]='l';
c[1]='\0';
write(myfd,(const void *)c,2);
printf("Loading List from the Server. \nPlease Wait..\n");
sleep((cnt/3)+1);
do
{
        read(myfd,(void *)handshake,15);
}
while(strcmp(handshake,"CNTSTART")!=0);
do
{       ret=read(myfd,(void *)&cnt,sizeof(int));
} while(ret==-1);
list=(struct client *)malloc(cnt*sizeof(struct client));
do
{       ret=read(myfd,(void *)list,cnt*sizeof(struct client));
}while(ret==-1);

for(i=0;i<cnt;i++)
{
        int j;
        if(list[i].fileno != 0)
        printf("\nFiles of client %d: -\n",i+1);
        for(j=0;j<list[i].fileno;j++)
        {
                printf("%s\n",list[i].list[j]);
        }
```

```
            }
            printf("\n");
}

//Function to download the file from another client.
int download(void* confd)
{
             void *s;
             int f;
                        char c[2];
                        void *filebuf;
             char *filename;
                        char *path;
                        struct sockaddr_in servaddr;
                        char remoteip[20];
             int clientno,i=0;
                        int myfd=*(int *)confd;
                        int dfd,n;
                        int filecnt=30;
                        filebuf=(void *)malloc(4096);
                        path=(char *)malloc(30);
                        filename=(char *)malloc(filecnt);
                        c[0]='d';
            c[1]='\0';
            write(myfd,(const void *)c,2);
            printf("Enter client number:");
                        scanf("%d",&clientno);
                        printf("Enter filename:\n");
                        scanf("%s",filename);
                        printf("Downloading %s. Please Wait..\n",filename);
```

```
        s=(void*)malloc(1000);
        write(myfd,(const void *)&clientno,sizeof(int));
                sleep(1);
                read(myfd,(void *)remoteip,20*sizeof(char));
                //Using another port to initiate client-client file transfer.
                dfd=socket(AF_INET,SOCK_STREAM,IPPROTO_TCP);
                bzero(&servaddr, sizeof(servaddr));
            servaddr.sin_family=AF_INET;
                servaddr.sin_addr.s_addr=htonl(INADDR_ANY);
            servaddr.sin_port=htons(15000);
                sleep(3);
        if(connect(dfd,(SA *) &servaddr,sizeof(servaddr))<0)
                         printf("Connection problem.\n");
                write(dfd,(const void *)filename,30);
                strcpy(path,"/home/");
        strcpy(path,strcat(path,getenv("USER")));
        strcpy(path,strcat(path,"/Downloads/"));
        strcpy(path,strcat(path,filename));
                f=open(path,O_CREAT | O_RDWR, 0755);
   while(1)
   {
                n=read(dfd,filebuf,4096);
        write(f,filebuf,4096);
        cnt++;
        if(n<=0) break;
   }
   free(filebuf);
   close(f);
   close(dfd);
   printf("\nDownload finished.\n\n");
```

```
        return 1;
}


//Function to upload the file requested by another client.
void downloadrequestcheck(int *fd)
{
            int myfd=*(int *)fd;
            int sockfd,n;
            char c;
            void *filebuf;
            int f;
            struct sockaddr_in servaddr,cliaddr;
            char filename[30];
            char *path;
        int len,confd;
            filebuf=(void *)malloc(4096);
            path=(char *)malloc(30);
            read(myfd,(void *)&c,sizeof(char));
            printf("Receive Download request.\n");
            sem_wait(&bin_sem);
                        //Using another port for file transfer, this client acting as a server.
                        sockfd=socket(AF_INET,SOCK_STREAM,IPPROTO_TCP);
            bzero(&servaddr,sizeof(servaddr));
            servaddr.sin_family=AF_INET;
            servaddr.sin_addr.s_addr=htonl(INADDR_ANY);
            servaddr.sin_port=htons(15000);
            bind(sockfd,(SA *) &servaddr,sizeof(servaddr));
            listen(sockfd,LISTENQ);
                        len=sizeof(cliaddr);
                        confd=accept(sockfd,(SA *) &cliaddr,&len);
```

```
                read(confd,(void *)filename,30);
                strcpy(path,"/home/");
            strcpy(path,strcat(path,getenv("USER")));
        strcpy(path,strcat(path,"/Sample/"));
                strcpy(path,strcat(path,filename));

                f=open(path,O_RDONLY);
         if(f<0)
         {
             printf("File couldn't be opened");
             return;
         }
         printf("Uploading %s\n",path);
                while(1)
         {
             n=read(f,filebuf,4096);
             cnt++;
             write(confd,(const void *)filebuf,4096);
             if(n<=0) break;
         }
         printf("Uploading Finished\n");
         free(filebuf);
         close(f);
         close(confd);
                printf("\nEnter choice:\n'l' to get list of files\n'd' to download file\n'q' to quit\n>");

        sem_post(&bin_sem);
}

int main(void)
```

```
{
	int myfd;
	void *c;
	pthread_t tid;
	socklen_t len;
	FILE* f;
	int dreturn;
	int ret=0;
	char *answer;
	char path[100];
	struct sockaddr_in servaddr,myaddr;
	c=(void*)malloc(1000);
	sem_init(&bin_sem, 0, 0);
	myfd=socket(AF_INET,SOCK_STREAM,0);
	bzero(&servaddr, sizeof(servaddr));
	bzero(&myaddr,sizeof(myaddr));
	servaddr.sin_family=AF_INET;
	servaddr.sin_addr.s_addr=htonl(INADDR_ANY);
	servaddr.sin_port=htons(13000);
	bind(myfd,(SA *) &myaddr,sizeof(myaddr));
	len=sizeof(servaddr);
	if(connect(myfd,(SA *) &servaddr,len)!=0)
	{	printf("Connection problem.\n");
		exit(-1);
	}
	printf("\nConnection Established !!\n");
	sendlist(&myfd);
	answer=(char *)malloc(5*sizeof(char));
	while(1)
	{
```

```
{

pthread_create(&tid,NULL,downloadrequestcheck,&myfd);
sem_post(&bin_sem);
printf("Enter choice:\n'l' to get list of files\n'd' to download file\n'q' to quit\n>");
scanf("%s",answer);
pthread_cancel(tid);
sem_wait(&bin_sem);
if(strcmp(answer,"l")==0)
{
	printf("\nList Request Sent to the Server.\n");
	printlist(&myfd);
}
else if(strcmp(answer,"d")==0)
{
	printf("\nDownload Request Sent to the Server.\n");
	dreturn=download(&myfd);
}
else if(strcmp(answer,"q")==0)

{
	char c[2];
	c[0]='q';
	c[1]='\0';
	write(myfd,(const void *)c,2);
	exit(0);
}
else
{
	printf("Invalid choice. Please enter your choice again\n");
	fflush(stdin);
}
```

```
        }
                return 0;
}
```

Filename: - ip.sh

```
set `ifconfig`

shift 6

k=`expr length $1`

g=`expr $k - 5`

echo $g

s=${1:5:g}

echo $s > ipaddress
```