COMPUTER NETWORKS PROJECT

GROUP CHAT IN UNIX ENVIRONMENT USING RASPBERRY PI 3 AS CONCURRENT SERVER

Date : 2nd June 2017

UNDER THE GUIDANCE OF

Dr. Narendran Rajagopalan
Assistant Professor, Dept. of CSE

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

NATIONAL INSTITUTE OF TECHNOLOGY PUDUCHERRY

MAY 2017

-Submitted By :

Rohan Singh Poona

Rajershi Gupta

Acknowledgement

The completion of this project could not have been possible without the basic foundation of Computer Networks given to us by our
Prof. Dr. Narendran Rajagopalan. This project is a result of a group of two people.

I would like to express my sincere gratitude to my supervisor
Dr. Narendran Rajagopalan providing his invaluable guidance, comments and suggestions throughout the course of the project.

Lastly, I would like to thank all those who helped me throughout the process of this project.

Abstract

Group chat between clients with Raspberry Pi acting as a concurrent server maintaining the chat records and allowing each client to send and receive data in its network.

The concurrent server was implemented in Raspberry Pi 3 running on UNIX Operating System remotely using Putty (SSH client). C programming language was used for implementing client and server in UNIX environment.

The Group chat allowed clients to communicate with each other, which we tested using 5 clients and raspberry pi 3 as a server.

Introduction

The project basically focuses on implementing a group chat between users in a UNIX environment connected through a concurrent server (Raspberry Pi) in a wireless local area network. Therefore, any data sent (or received) between the users goes via this central sever.

The server can be iterative i.e. it iterates each client and serves one request at a time. Moreover, a server can handle multiple clients at the same time in parallel and this type of server is called concurrent server.

The Raspberry Pi 3 acted as a concurrent server which was implemented remotely using PuTTY (SSH) in C programming Language.

PuTTY is a versatile tool for remote access to another computer. It is mostly used by people who want secure remote shell access to a UNIX system.

The client program was implemented in C which allows computers running in UNIX Environment to connect to the concurrent sever and chat with other clients.

Some of the networking functions that were used to implement this application:

- Socket function;
- Bind function;
- Listen function;
- Accept function;
- Fork function;
- Read function;
- Write function;

Many other user-defined functions were made and standard libraries were used which can be seen in the code portion of this document.

Algorithms are given before each of the concurrent server and client code to help a reader understand the code in a faster and convenient way.

Group Chat - Server

Algorithm

1. Server is made concurrent by calling fork() which returns two values (one for parent and one for child) and makes a child process which handles each client.
2. echo()
   - 2.1. It reads the uname sent by the client.
   - 2.2. Then reads the message sent by the client.
   - 2.3. If EMPTY => It appends the message with the **\nuname: (message\n)** to the file and writes the message **“WAIT…”** to the client .
   - 2.4. If NOT EMPTY=> It tries to read and check the uname of previous message.
     - 2.4.1. If it does not matches with its uname then reads back into the file to again read a uname and check for the match.
     - 2.4.2. If it matches with the uname then it moves the pointer to the next uname(\n).
   - 2.5. Now it reads from the SEEK_CUR to the EOF and writes it back to the client.
3. This echo() and fork() are called recursively by the server for each client.

Program

```
//groupserv.c

#include<stdio.h>

#include<stdlib.h>

#include<netinet/in.h>

#include<sys/socket.h>

#include<sys/types.h>

#include<string.h>
```

```
#include<arpa/inet.h>
#define MAX 25
void echo(int a);

int main()
{
	struct sockaddr_in serv,cli;
	int len,confd,sockfd;
	pid_t child;

	sockfd=socket(AF_INET,SOCK_STREAM,0);
	if(sockfd<0)
		printf("E_SOCKET");
	else
		printf("SOCKET_BUILD\n");

	bzero(&serv,sizeof(serv));
	serv.sin_family=AF_INET;
	serv.sin_port=htons(5555);
	serv.sin_addr.s_addr=htonl(INADDR_ANY);

	if(bind(sockfd,(struct sockaddr*)&serv,sizeof(serv)) < 0 )
		printf("E_BIND");
	else
		printf("BIND");

	listen(sockfd,5);
	len=sizeof(cli);
	while(1) {
		confd=accept(sockfd,(struct sockaddr*)&cli,&len);
		if(confd < 0)
```

```
                printf("E_ACCEPT");
        else {
                printf("ACCEPTED");

                if((child=fork())==0) {
                        close(sockfd);
                        echo(confd);
                        exit(0);
                }
                else
                        close(confd);
                }
        }
        return 0;
}

void echo(int sockfd)
{
        FILE *f;
        int n,i,flag,k;
        char buff[MAX],uname[MAX],rbuff[50],str[100];

         bzero(buff,sizeof(buff));
        read(sockfd,buff,sizeof(buff));
        strcpy(uname,buff);

        while(1) {
                flag=0;

                bzero(buff,sizeof(buff));
                n=read(sockfd,buff,sizeof(buff));
```

```
if(n>0)
printf("\n CLIENT: %s <end>",buff);
else if(n==0) {
        printf("read is returning 0\n");
}
else {
        printf("read failed\n");
        break;
}
if(strncmp(buff,"exit",4)==0)  {
         printf("\n Client wants to exit\n");
           break;
}
else {
        f=fopen("file.txt","a+");
        fseek(f,0,SEEK_SET);
        printf("Writing back to client : %s\n",buff);

         if(fgetc(f)!=EOF) {
                bzero(rbuff,sizeof(rbuff));
                fseek(f,-2,SEEK_END);

                for(i=0;i<200;i++) {
                        fseek(f,-2,SEEK_CUR);
                        if(fgetc(f)=='\n') {
                                        bzero(rbuff,sizeof(rbuff));
                                        fgets(rbuff,strlen(uname),f);
                                        printf("RBUFF_NAME
COMPARISION===%s",rbuff);
```

```
        if(strncmp(rbuff,uname,strlen(rbuff))!=0) {
                                                    if( ftell(f)<=7) {
                                                            fseek(f,0,SEEK_SET);
                                                            break;
                                                    }
                                                    fseek(f,-(strlen(uname)
+2),SEEK_CUR);
                                            }
                                                    else {
                                                            printf("\n NAME IS
MATCHED");
                                                    while(fgetc(f)!='\n') ;
                                                    break;
                                                    }
                                            }
                                    }
                                     fseek(f,1,SEEK_CUR);
                                    bzero(str,sizeof(str));
                                    while(fgets(rbuff,30,f)) {
                                     printf("\n\n RBUFF: %s",rbuff);
                                            strncat(str,rbuff,strlen(rbuff));
                                            bzero(rbuff,sizeof(rbuff));
                                    }
                                    printf("STRING:::%s",str);
                                    write(sockfd,str,strlen(str));
                                    flag=1;
                            }
                            else{printf("\nN=%d\n",n);}

                             fwrite("\n",strlen("\n"),1,f);
                            fwrite(uname,strlen(uname)-1,1,f);
```

```
                        fwrite(":",strlen(":"),1,f);
                        fwrite(buff,strlen(buff),1,f);
                        fclose(f);
                        if(flag!=1)
                                write(sockfd,"wait...",strlen("wait..."));
                }
        }
}
```

Group Chat – Client

Algorithm

1. Client inputs uname and writes it to server
2. Then it writes the chat message to the other client(s) via server.
3. Then read function is called which reads the message sent by the other client via concurrent server.
4. Step 2 and Step 3 are repeated until client writes EXIT.

Program

```
// client.c
 #include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include<arpa/inet.h>
#define MAX 100

int main()
{
        int sockfd;
        struct sockaddr_in servaddr;
```

```
        char buff[MAX];

        sockfd=socket(AF_INET,SOCK_STREAM,0);
        bzero(&servaddr,sizeof(servaddr));

        servaddr.sin_family=AF_INET;
        if(inet_pton(AF_INET,"192.168.225.229",&servaddr.sin_addr)<=0)
                printf("E_INET_PTON");
        servaddr.sin_port=htons(5555);

        if(connect(sockfd,(struct sockaddr *) &servaddr,sizeof(servaddr))<0)
                printf("Error_in_CONNECTING\n");

        bzero(buff,sizeof(buff));
        printf("\nEnter uname: ");
        fgets(buff,sizeof(buff),stdin);
        write(sockfd,buff,strlen(buff));
        while(1) {
                printf("Enter the data to be send: ");
                bzero(buff,sizeof(buff));
                fgets(buff,sizeof(buff),stdin);
                write(sockfd,buff,strlen(buff));

                if(strncmp(buff,"exit",4)==0)
                        break;
                bzero(buff,sizeof(buff));
                read(sockfd,buff,sizeof(buff));
                printf("\n %s \n",buff);
        }
        close(sockfd);
        return 0;
}
```

Conclusion

It was a wonderful learning experience for me while working on this project. This project took me through various phases of project development and gave me a real insight into the concept of Computer Networking. I enjoyed each and every bit of work that I had put into this project.

Hence, the group chat was successfully implemented and verified on the terminals of many (upto 5) client users connected in a wireless local area network.

By writing a frontend code for chat, my code can be used for making a full-fledged working application for users to chat in a local area network. Thus, this project is further extendable.

References

1. UNIX® Network Programming Volume 1, Third Edition: The Sockets Networking API
By W. Richard Stevens, Bill Fenner, Andrew M. Rudoff

2. UNIX Concepts and Applications Fourth Edition - By Sumitabha Das