CHAUDHARY HAMDAN
1905387
Networks Lab 8
02/08/2021

1. Non blocking communication by IO Multiplexing.

Code (Server):

```
#include <stdio.h>        /* These are the usual header files */
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <string.h>
#include <errno.h>
#include <stdlib.h>
#include <sys/select.h>
#include <sys/time.h>
#include <unistd.h>
#include <fcntl.h>
#include <sys/ioctl.h>

#define BACKLOG 2   /* Number of connections in queue before being
rejected*/
#define MAXDATASIZE 1000
#define STDIN 0

int main(int argc, char *argv[])
{
        int listen_fd, conn_fd, i;

        int client_fd[FD_SETSIZE];
        int max_index;
        int sin_size;

        struct sockaddr_in server; /* server's address information */
        struct sockaddr_in client; /* client's address information */

        struct sockaddr_in temp;

        if (argc != 2) {            /* this is used because our program will need 1
argument (port) */
                printf("Usage: %s <port>\n", argv[0]);
```

```
		exit(-1);
	}

	if ((listen_fd = socket(AF_INET, SOCK_STREAM, 0)) == -1 ) {
		printf("socket() error\n");
		exit(-1);
	}

	server.sin_family = AF_INET;
	server.sin_port = htons(atoi(argv[1]));	/* Remember htons() from
"Conversions" section? =) */
	server.sin_addr.s_addr = INADDR_ANY;/* INADDR_ANY puts your
IP address automatically */
	bzero(&(server.sin_zero), 8);		/* zero the rest of the structure */

	if (bind(listen_fd, (struct sockaddr*)&server, sizeof(struct sockaddr))
== -1) {
		printf("bind() error\n");
		exit(-1);
	}

	if (listen(listen_fd, BACKLOG) == -1) {
		printf("listen() error\n");
		exit(-1);
	}
//-------------------------------------------------------------------------------------
---------------------------------
	fd_set read_set, write_set, all_set;
	struct timeval timeout;
	int ret_val;

	max_index = 0;
	for (i = 0; i < FD_SETSIZE; i++)
	{
		client_fd[i] = -1;
	}

	//highest-numbered file descriptor in any of the three sets.
	int max_fd = listen_fd;

	/*initialising fd sets that are storing status of various fd*/
	FD_ZERO(&read_set);
	FD_ZERO(&write_set);
	FD_ZERO(&all_set);

	//Make the standard input socket non-blocking
	//fcntl(STDIN, F_SETFL, O_NONBLOCK);

	FD_SET(listen_fd, &all_set);

	while (1)
```

```
	{
		read_set = all_set;
		write_set = all_set;

		timeout.tv_sec = 100;
		timeout.tv_usec = 0;

		ret_val = select(max_fd + 1, &read_set, NULL, NULL,
&timeout);
		/*if(ret_val == 0){
		  printf("\nTimeout occurred!  No data after 10 seconds.\n");
		}
		else*/
//_____
_____
_____

		if (ret_val == -1) {
			perror("select");
			exit(-1);
		}
		else if (FD_ISSET(listen_fd, &read_set)) { //server listen_fd is
set to accept new client connection
			int sin_size = sizeof(struct sockaddr_in);
			if ((conn_fd = accept(listen_fd, (struct sockaddr *)&client,
&sin_size)) == -1) {
				perror("accept");
				exit(-1);
			}
			printf("accepted a new connection\n");
			/*char temp_buf[INET_ADDRSTRLEN];

	inet_ntop(AF_INET,&(client.sin_addr),temp_buf,INET_ADDRSTRLE
N);
			if(temp_buf == NULL)
				printf("inet_ntop error\n");
			else
				printf("accepted connection from client with IP
Addr : %s and port Number :%u\n",temp_buf,ntohs(client.sin_port));*/

			//store the new connection fd in the array
			if (max_index == FD_SETSIZE)
				printf("too many clients to handle\n");
			else {
				client_fd[max_index] = conn_fd;
				max_index++;
			}
			//Set the conection fd to read and write data
			FD_SET(conn_fd, &all_set);
			if (conn_fd > max_fd)
				max_fd = conn_fd;
```

```
                } //end of FD_ISSET

//_____________________________________________________________________
_____________________________________________________________________
_____________________________________
                else { //Check the client for data
                        for (i = 0; i < FD_SETSIZE; i++) //loop checks all
possible sockets that are connected
                        {
                                sin_size = sizeof(struct sockaddr_in);
                                char temp_buf[INET_ADDRSTRLEN];
                                char buf[1000];
                                //checks if a particular socket is ready for reading
                                if (FD_ISSET(client_fd[i], &read_set))
                                {
                                        int numbytes;
                                        numbytes = recv(client_fd[i], buf,
MAXDATASIZE, 0);
                                        if (numbytes > 0)
                                        {

                                                printf("Message from the client
is %s\n", buf);

                                                /*buf[numbytes]='\0';
                                                if(getpeername(client_fd[i],(struct
sockaddr *)&temp,&sin_size)==-1)
                                                                {

        printf("Peername error\n");
                                                                        exit(-1);
                                                                }

        inet_ntop(AF_INET,&(temp.sin_addr),temp_buf,INET_ADDRSTRLE
N);
                                                                if(temp_buf ==
NULL)

        printf("inet_ntop error\n");
                                                                else

        printf("Message from client %s on
port %u :%s",temp_buf,ntohs(temp.sin_port),buf);*/
                                        }//if(numbytes>0)
                                        printf("Enter your message for the
client\n");
                                        fgets(buf, 1000, stdin);
                                        send(client_fd[i], buf, strlen(buf), 0);

                                        /*if(getpeername(client_fd[i],(struct
sockaddr *)&temp,&sin_size)==-1)
```

```
                                        {
                                                printf("Peername error\n");
                                                exit(-1);
                                        }


        inet_ntop(AF_INET,&(temp.sin_addr),temp_buf,INET_ADDRSTRLE
N);
                                        if(temp_buf == NULL)
                                                printf("inet_ntop error\n");
                                        else
                                                printf("My message to client %s on
port %u :",temp_buf,ntohs(temp.sin_port));
                                        fgets(buf,1000,stdin);
                                        send(client_fd[i],buf,strlen(buf),0);*/ /*
send message to client */
                                } //if(FD_ISSET(client_fd[i],&read_set))
                        }// end of for
                }//end of else
//_____________________________________________________________________
_____________________________________________________________________
_____________________
        }//end of while
}
```

Code (Client):

```
#include <stdio.h>
#include <sys/types.h>
#include <errno.h>
#include <stdlib.h>
#include <string.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <netdb.h>       /* netbd.h is needed for struct hostent =) */
#include <fcntl.h>

#define MAXDATASIZE 100   /* Max number of bytes of data */
#define STDIN 0

int main(int argc, char *argv[])
{
 int fd, numbytes, nval;       /* files descriptors */
```

```
  char buf[MAXDATASIZE];  /* buf will store received text */

  struct hostent *he;        /* structure that will get information about remote
host */
  struct sockaddr_in server;  /* server's address information */

  if (argc !=3) {              /* this is used because our program will need 2
arguments (IP,port) */
    printf("Usage: %s <IP Address> <port>\n",argv[0]);
    exit(-1);
  }

  if ((fd=socket(AF_INET, SOCK_STREAM, 0))==-1){  /* calls socket() */
    printf("socket() error\n");
    exit(-1);
  }

  server.sin_family = AF_INET;
  server.sin_port = htons(atoi(argv[2])); /* htons() is needed again */
  //server.sin_addr = *((struct in_addr *)he->h_addr);  /*he->h_addr passes
"*he"'s info to "h_addr" */
  server.sin_addr.s_addr = inet_addr(argv[1]);
  bzero(&(server.sin_zero),8);

  if(connect(fd, (struct sockaddr *)&server,sizeof(struct sockaddr))==-1){ /*
calls connect() */
    printf("connect() error\n");
    exit(-1);
  }

static char buf1[1000];
//defining sets to hold the fd for using them with select function. Master is
necessary because these sets will change once they are sent to the select
function
fd_set rset,master,wset;
        struct timeval timeout;
/*initialising fd sets that are storing status of fd*/
        FD_ZERO(&rset);
        FD_ZERO(&master);
        //FD_ZERO(&wset);
//adding fd to master set
        FD_SET(fd,&master);
        FD_SET(STDIN,&master);

do
{
        rset=master;
        //wset=master;

//time tells select to monitor for that much amount of time.
        timeout.tv_sec=100;
```

```
        timeout.tv_usec=0;

        //select system call takes two sets to check if fd is ready for readio or
writeio
        nval=select(fd+1,&rset,NULL,NULL,&timeout);//using select to verify
if it is possible to
        //nval=select(fd+1,&rset,&wset,NULL,NULL);//using select to verify if
it is possible to

        //if some data waiting to be read. then read it
        if(FD_ISSET(fd,&rset))
        {
            numbytes=recv(fd,buf,MAXDATASIZE,0);
            buf[numbytes]='\0';
            printf("Server Message: %s\n",buf);
        }
        //printf("My Message to server:");
        //if socket is ready to be written then scan data from user and send it
        if(FD_ISSET(STDIN,&rset))
        {
                fgets(buf1,1000,stdin);
                send(fd,buf1,strlen(buf1),0);
        }

}while(strncmp(buf,"exit",4)!=0);

    close(fd);   /* close fd =) */

}
```

Output: