**9. IPC: Sockets**

**9.4** Write two programs (server and client) to show how you canestablish a UDP socket connection using the above functions.

**OBJECTIVES:**

1. To learn about fundamentals of IPC through C socket programming.
2. Learn and understand the OS intraction with socket programming.
3. Use of system call and IPC mechanism to write effective application programs.
4. To know the port numbersing and process relation.
5. To knows the iterative and concurrent server concept.

**THEORY:**

JAVA SOCKET PROGRAMMING

TCP is a connection-oriented protocol layered on the top of IP of the TCP/IP stack with the ability to acknowledge receipt of packets at both ends. Acknowledgement ensures that the lost/corrupt packets can be retransmitted upon request. It also maintains a sequence in the sense that packets can be put back in the same order at the receiving end as they were transmitted. Although everything seems fair and advantageous at first look, it is also its weakness on occasion, because maintaining a guaranteed data transmission carries a fair amount of overhead (the header size of TCP packet is 20 bit whereas UDP header is 8 bit). In a situation where the order of the data is not that important or say, loss of a few packets does not matter to the verge of completely corrupting the data, TCP can be a real bottleneck. UDP is an unreliable connectionless protocol that neither guarantees that the packets will ever reach the destination nor that they will arrive in the same order they were sent. But, it works and surprisingly reaches the destination, without the slightest aura of "guarantee" or "reliability." TCP can be best suited for file transfer or the like where loss of bits is unacceptable. UDP, on the other hand, is best suited where a little loss in the transmission bits does not matter. For example, a few lost bits in video or audio signals are less severe without much quality degradation. Further, error correction in UDP can be built into data streams at the application level to account for missing information. So, UDP is not a total loss, after all.

**Program:**

**SERVER:**

#include <stdio.h>

#include <stdlib.h>

#include <unistd.h>

#include <errno.h>

#include <string.h>

#include <sys/types.h>

#include <sys/socket.h>

#include <netinet/in.h>

#include <arpa/inet.h>

#define MYPORT 4950

#define MAXBUFLEN 500

int main(int argc, char \*argv[])

{

int sockfd;

struct sockaddr\_in my\_addr;

struct sockaddr\_in their\_addr;

int addr\_len, numbytes;

char buf[MAXBUFLEN];

if((sockfd = socket(AF\_INET, SOCK\_DGRAM, 0)) == -1)

{

perror("Server-socket() sockfd error lol!");

exit(1);

}

else

printf("Server-socket() sockfd is OK...\n");

/\* host byte order \*/

my\_addr.sin\_family = AF\_INET;

my\_addr.sin\_port = htons(MYPORT);

my\_addr.sin\_addr.s\_addr = INADDR\_ANY;

memset(&(my\_addr.sin\_zero), '\0', 8);

if(bind(sockfd, (struct sockaddr \*)&my\_addr, sizeof(struct sockaddr)) == -1)

{

perror("Server-bind() error lol!");

exit(1);

}

else

printf("Server-bind() is OK...\n");

addr\_len = sizeof(struct sockaddr);

if((numbytes = recvfrom(sockfd, buf, MAXBUFLEN-1, 0, (struct sockaddr \*)&their\_addr, &addr\_len)) == -1)

{

perror("Server-recvfrom() error lol!");

exit(1);

}

else

{

printf("Server-Waiting and listening...\n");

printf("Server-recvfrom() is OK...\n");

}

printf("Server-Got packet from %s\n", inet\_ntoa(their\_addr.sin\_addr));

printf("Server-Packet is %d bytes long\n", numbytes);

buf[numbytes] = '\0';

printf("Server-Packet contains \"%s\"\n", buf);

if(close(sockfd) != 0)

printf("Server-sockfd closing failed!\n");

else

printf("Server-sockfd successfully closed!\n");

return 0;

}

**CLIENT:**

#include <stdio.h>

#include <stdlib.h>

#include <unistd.h>

#include <errno.h>

#include <string.h>

#include <netdb.h>

#define MYPORT 4950

int main(int argc, char \*argv[ ])

{

int sockfd;

struct sockaddr\_in their\_addr;

struct hostent \*he;

int numbytes;

if (argc != 3)

{

fprintf(stderr, "Client-Usage: %s <hostname> <message>\n", argv[0]);

exit(1);

}

if ((he = gethostbyname(argv[1])) == NULL)

{

perror("Client-gethostbyname() error lol!");

exit(1);

}

else

printf("Client-gethostname() is OK...\n");

if((sockfd = socket(AF\_INET, SOCK\_DGRAM, 0)) == -1)

{

perror("Client-socket() error lol!");

exit(1);

}

else

printf("Client-socket() sockfd is OK...\n");

their\_addr.sin\_family = AF\_INET;

printf("Using port: 4950\n");

their\_addr.sin\_port = htons(MYPORT);

their\_addr.sin\_addr = \*((struct in\_addr \*)he->h\_addr); memset(&(their\_addr.sin\_zero), '\0', 8

if((numbytes = sendto(sockfd, argv[2], strlen(argv[2]), 0, (struct sockaddr \*)&their\_addr, sizeof(struct sockaddr))) == -1)

{

perror("Client-sendto() error lol!");

exit(1);

}

else

printf("Client-sendto() is OK...\n");

printf("sent %d bytes to %s\n", numbytes, inet\_ntoa(their\_addr.sin\_addr));

if (close(sockfd) != 0)

printf("Client-sockfd closing is failed!\n");

else

printf("Client-sockfd successfully closed!\n");

return 0;

}

**Conclusion:**

1.UDP socket connetion using system calls in C studied

2.Client server connection established.

**References:**

[1] www.cs.cf.ac.uk/Dave/C/CE.html