TCP and UDP server using select

Prerequisites: TCP UDP

In previous articles we have seen a **TCP** server and a **UDP** server. But now we can combine our concurrent TCP echo server an ative UDP server into a single server that uses `select` to multiplex TCP and UDP socket.

Select function is used to select between TCP and UDP socket. This function gives instructions to the kernel to wait for any of multiple events to occur and awakens the process only after one or more events occur or a specified time passes.

Example – kernel will return only when one of these condition occurs

- Any Descriptor from {1, 2, 3} is ready for reading
- Any Descriptor from {4, 5, 6} is ready for writing
- Time 5sec have passed

The entire process can be broken down into following steps :
Server:

1. Create TCP i.e Listening socket
2. Create a UDP socket
3. Bind both socket to server address.
4. Initialize a descriptor set for select and calculate maximum of 2 descriptor for which we will wait
5. Call select and get the ready descriptor(TCP or UDP)
6. Handle new connection if ready descriptor is of TCP OR receive data gram if ready descriptor is of UDP

UDP Client:

1. Create UDP socket.
2. Send message to server.
3. Wait until response from server is recieved.
4. Close socket descriptor and exit.

TCP Client:

1. Create a TCP scoket.
2. Call connect to establish connection with server
3. When the connection is accepted write message to server
4. Read response of Server
5. Close socket descriptor and exit.

Necessary functions:

```
int select(int maxfd, fd_set *readsset, fd_set *writeset,
fd_set *exceptset, const struct timeval *timeout);
Returns: positive count of descriptors ready, 0 on timeout, -1 error
```

Arguments:

1. **maxfd:** maximum number of descriptor ready.
2. **timeout:** How long to wait for select to return.
3.
```
struct timeval{
long tv_sec;
long tv_usec;
};
if timeout==NULL then wait forever
if timeout == fixed_amount_time then wait until specified time
if timeout == 0 return immediately.
```
4. **readset:** Descriptor set that we want kernel to test for reading.
5. **writeset:** Descriptor set that we want kernel to test for writing.
6. **exceptset:** Descriptor set that we want kernel to test for exception conditions.

```
int read(int sockfd, void * buff, size_t nbytes);
Returns:  number of bytes read from the descriptor. -1 on error
```

Arguments:

1. **sockfd:** Descriptor which receives data.
2. **buff:** Application buffer socket descriptor data is copied to this buffer.
3. **nbytes:**Number of bytes to be copied to application buffer.

Server.c

```c
// Server program
#include <arpa/inet.h>
#include <errno.h>
#include <netinet/in.h>
#include <signal.h>
#include <stdio.h>
#include <stdlib.h>
#include <strings.h>
#include <sys/socket.h>
#include <sys/types.h>
#include <unistd.h>
#define PORT 5000
#define MAXLINE 1024
int max(int x, int y)
{
    if (x > y)
        return x;
    else
        return y;
}
int main()
{
    int listenfd, connfd, udpfd, nready, maxfdp1;
    char buffer[MAXLINE];
    pid_t childpid;
    fd_set rset;
    ssize_t n;
    socklen_t len;
    const int on = 1;
    struct sockaddr_in cliaddr, servaddr;
    char* message = "Hello Client";
    void sig_chld(int);

    /* create listening TCP socket */
    listenfd = socket(AF_INET, SOCK_STREAM, 0);
    bzero(&servaddr, sizeof(servaddr));
    servaddr.sin_family = AF_INET;
    servaddr.sin_addr.s_addr = htonl(INADDR_ANY);
    servaddr.sin_port = htons(PORT);

    // binding server addr structure to listenfd
    bind(listenfd, (struct sockaddr*)&servaddr, sizeof(servaddr));
    listen(listenfd, 10);

    /* create UDP socket */
    udpfd = socket(AF_INET, SOCK_DGRAM, 0);
    // binding server addr structure to udp sockfd
    bind(udpfd, (struct sockaddr*)&servaddr, sizeof(servaddr));

    // clear the descriptor set
    FD_ZERO(&rset);

    // get maxfd
    maxfdp1 = max(listenfd, udpfd) + 1;
    for (;;) {

        // set listenfd and udpfd in readset
        FD_SET(listenfd, &rset);
        FD_SET(udpfd, &rset);

        // select the ready descriptor
        nready = select(maxfdp1, &rset, NULL, NULL, NULL);

        // if tcp socket is readable then handle
        // it by accepting the connection
        if (FD_ISSET(listenfd, &rset)) {
            len = sizeof(cliaddr);
            connfd = accept(listenfd, (struct sockaddr*)&cliaddr, &len);
            if ((childpid = fork()) == 0) {
                close(listenfd);
                bzero(buffer, sizeof(buffer));
                printf("Message From TCP client: ");
                read(connfd, buffer, sizeof(buffer));
                puts(buffer);
                write(connfd, (const char*)message, sizeof(buffer));
                close(connfd);
                exit(0);
            }
            close(connfd);
        }
        // if udp socket is readable receive the message.
        if (FD_ISSET(udpfd, &rset)) {
            len = sizeof(cliaddr);
            bzero(buffer, sizeof(buffer));
            printf("\nMessage from UDP client: ");
            n = recvfrom(udpfd, buffer, sizeof(buffer), 0,
                         (struct sockaddr*)&cliaddr, &len);
            puts(buffer);
            sendto(udpfd, (const char*)message, sizeof(buffer), 0,
                   (struct sockaddr*)&cliaddr, sizeof(cliaddr));
        }
    }
}
```

TCP_Client.c

```c
// TCP Client program
#include <netinet/in.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <sys/socket.h>
#include <sys/types.h>
#define PORT 5000
#define MAXLINE 1024
int main()
{
    int sockfd;
    char buffer[MAXLINE];
    char* message = "Hello Server";
    struct sockaddr_in servaddr;

    int n, len;
    // Creating socket file descriptor
    if ((sockfd = socket(AF_INET, SOCK_STREAM, 0)) < 0) {
        printf("socket creation failed");
        exit(0);
    }

    memset(&servaddr, 0, sizeof(servaddr));

    // Filling server information
    servaddr.sin_family = AF_INET;
    servaddr.sin_port = htons(PORT);
    servaddr.sin_addr.s_addr = inet_addr("127.0.0.1");

    if (connect(sockfd, (struct sockaddr*)&servaddr,
                sizeof(servaddr)) < 0) {
        printf("\n Error : Connect Failed \n");
    }

    memset(buffer, 0, sizeof(buffer));
    strcpy(buffer, "Hello Server");
    write(sockfd, buffer, sizeof(buffer));
    printf("Message from server: ");
    read(sockfd, buffer, sizeof(buffer));
    puts(buffer);
    close(sockfd);
}
```

UDP_client.c

```c
// UDP client program
#include <arpa/inet.h>
#include <netinet/in.h>
#include <stdio.h>
#include <stdlib.h>
#include <strings.h>
#include <sys/socket.h>
#include <sys/types.h>
#define PORT 5000
#define MAXLINE 1024
int main()
{
    int sockfd;
    char buffer[MAXLINE];
    char* message = "Hello Server";
    struct sockaddr_in servaddr;

    int n, len;
    // Creating socket file descriptor
    if ((sockfd = socket(AF_INET, SOCK_DGRAM, 0)) < 0) {
        printf("socket creation failed");
        exit(0);
    }

    memset(&servaddr, 0, sizeof(servaddr));

    // Filling server information
    servaddr.sin_family = AF_INET;
    servaddr.sin_port = htons(PORT);
    servaddr.sin_addr.s_addr = inet_addr("127.0.0.1");
    // send hello message to server
    sendto(sockfd, (const char*)message, strlen(message),
           0, (const struct sockaddr*)&servaddr,
           sizeof(servaddr));

    // receive server's response
    printf("Message from server: ");
    n = recvfrom(sockfd, (char*)buffer, MAXLINE,
                 0, (struct sockaddr*)&servaddr,
                 &len);
    puts(buffer);
    close(sockfd);
    return 0;
}
```

Steps to compile and run the above codes:

1. Compile the server program (`gcc server.c -o ser`)
2. Run server using (`./ser`)
3. On another terminal, compile tcp client program (`gcc tcp_client.c -o tcpcli`)
4. Run tcp client (`./tcpcli`)
5. On another terminal, compile udp client program (`gcc udp_client.c -o udpcli`)
6. Run udp client (`./udpcli`)

Output of the above codes: