**Week 13:Client Server application using Multiprotocol server**

Here we combine our concurrent TCP echo server and iterative 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 the multiple events to occur and awakens the process only after one or more events occur or a specified time passes.

**Server.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**

|  |
| --- |
| // 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**

|  |
| --- |
| // 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)

