Przykład aplikacji UDP

```
#include <sys/types.h>
#include <sys/socket.h>
#include <stdlib.h>          /* atoi */
#include <netinet/in.h>   /* htons, ntohs ... */
#include <string.h>         /* memset */
#include <arpa/inet.h>    /* inet_ntoa */
#include <stdio.h>          /* getchar */

int main(int argc,char **argv)
{
  struct sockaddr_in addr;
  int sock,ret,len;
  char c;

  if (argc<3)
  {
    printf("udp_app <port> n|o [ <ip_odbiorcy>
          <port_odbiorcy> ] \n\n");
    return 5;
  }
```

```
  memset(&addr,'\0',sizeof(addr));
  addr.sin_family = AF_INET;
  addr.sin_port = htons(atoi(argv[1]));
  addr.sin_addr.s_addr = inet_addr("192.168.1.1");

  sock = socket(AF_INET,SOCK_DGRAM,0);
  if (sock==-1) {
    printf("Nie moge utworzyc gniazdka!\n");
    return 2;
  }
  if (bind(sock,(struct sockaddr*)&addr, sizeof(addr))==-1)
  {
    printf("Nie moge dowiazac gniazdka!\n");
    return 3;
  }
```

Przykład aplikacji UDP cd.

```
switch (argv[2][0])
 {
  case 'o': printf("Odbiorca, czekam na dane:\n\n");
  while (1)
  {
   memset(&addr,0,sizeof(addr));
   ret = recvfrom(sock,&c,1,0,(struct sockaddr*) &addr, &len);
   printf("Otrzymalem znak %c\n",c);
   if (c=='q') break;
  }
    break;
  case 'n': printf("Nadawca, podaj znaki z klawiatury.
                    Znak 'q' konczy prace.\n\n");
    if (argc>=5)
    {
     memset(&addr,0,sizeof(addr));
     addr.sin_family = AF_INET;
     addr.sin_port = htons(atoi(argv[4]));
     addr.sin_addr.s_addr =  inet_addr(argv[3]);
    while (1)
    {
      c = (char)getchar();
      ret = sendto(sock,&c,1,0,
                          struct sockaddr*)&addr,
                          sizeof(addr));
      if (c=='q') break;
     }
    } else
      {
       printf("Nie podales adresu ip i portu
                    odbiorcy\n\n");
      }
            break;
    } /* switch */
     close(sock);
    } /* main */
```

Klient TCP

```
int main(int argc,char **argv)
{
  struct sockaddr_in addr,servaddr;
  int sock,ret,len;
  char c;

  if (argc<4)  {
    printf("tcp_app_c <port> <ip_serwera>
              <port_serwera>\n\n"); return 5;  }
  memset(&addr,'\0',sizeof(addr));
  addr.sin_family = AF_INET;
  addr.sin_port = htons(atoi(argv[1]));
  addr.sin_addr.s_addr = inet_addr("192.168.1.1");
  sock = socket(AF_INET,SOCK_STREAM,0);
  if (sock==-1) {
     printf("Nie moge utworzyc gniazdka!\n"); return 2;  }
  if (bind(sock,(struct sockaddr*)&addr, sizeof(addr))==-1) {
     printf("Nie moge dowiazac gniazdka!\n"); return 3;
  }
  memset(&servaddr,0,sizeof(servaddr));
  servaddr.sin_family = AF_INET;
  servaddr.sin_port = htons(atoi(argv[3]));
  servaddr.sin_addr.s_addr = inet_addr(argv[2]);
  if (connect(sock,(struct sockaddr*)&servaddr,
              sizeof(servaddr))==0)  {
     printf("Polaczenie nawiazane.
            Podaj znaki. Znak q konczy program.\n\n");
     while (1)   {
       c = (char)getchar();
       send(sock,&c,1,0);
       if (c=='q') break;  }
   }  else  { printf("Nie moge nawiazac polaczenia!\n");
   }
   close(sock);
}
```

Serwer TCP

```
int main(int argc,char **argv)
{
 struct sockaddr_in addr,servaddr;
 int sock,ret,len,sock2;
 char c;
 if (argc<2)  { printf("tcp_app_s <port> \n\n");return 5;  }
 memset(&servaddr,0,sizeof(servaddr));
 servaddr.sin_family = AF_INET;
 servaddr.sin_port = htons(atoi(argv[1]));
 servaddr.sin_addr.s_addr = inet_addr("192.168.1.1");
 sock = socket(AF_INET,SOCK_STREAM,0);
 if (sock==-1) {
  printf("Nie moge utworzyc gniazdka!\n");return 2;  }
 if (bind(sock,(struct sockaddr*)&servaddr,
          sizeof(servaddr))==-1) {
   printf("Nie moge dowiazac gniazdka!\n");return 3;  }
 if (listen(sock,5)==-1) {
   printf("Nie moge ustawic gniazdka w tryb nasluchu!\n");
   return 4; }
 printf("Serwer. Oczekuje na polaczenie\n\n");
 memset(&addr,0,sizeof(addr));
 len=0;
 sock2 =
 accept(sock,(structsockaddr*)&addr,&len);
 if (sock2>0)
 {
 printf("Polaczenie nawiazane z %s.\n\n",
          inet_ntoa(addr.sin_addr));
  while (1)  {
   recv(sock2,&c,1,0);
   if (c=='q') break;
   printf("Otrzymalem znak %c\n",c); }
 close(sock2);
  } else  {
    printf("Blad: %s\n",strerror(errno));
  }
 close(sock); }
```

Interfejs gniazd – zagadnienia dodatkowe

- Dane priorytetowe – MSG_OOB – realizacja i wykorzystanie
- Tryb nieblokujący: *fcntl (sock, F_SETFL, O_NONBLOCK | fcntl(sock,F_GETFL,0))*
- Przeglądanie zbioru deskryptorów:

select (int numfds, fd_set *readfs, fd_set *writefs, fd_set *exceptfs,struct timeval *timeout)

FD_ZERO(fd_set *set) - wyczyść zbiór
FD_SET(int fd, fd_set *set) - dodaj deskryptor do zbioru
FD_CLR(int fd, fd_set *set) - usuń deskryptor ze zbioru
FD_ISSET(inf fd, fd_set *set) - sprawdź, czy deskryptor pozostał w zbiorze

struct timeval { int tv_sec; int tv_usec; }

- Serwery iteracyjne
- Serwery współbieżne

Funkcje pomocnicze

- int **getsockname**(int fd, struct sockaddr* addr, socklen_t *len) - adres lokalny
- int **getpeername**(int fd, struct sockaddr* addr, socklen_t *len) - adres zdalny
- struct hostent ***gethostbyname**(const char *name) - adres wg nazwy

```
struct hostent {   char *h_name;
                   char **h_aliases;
                   int h_addrtype; /* AF_INET */
                   int h_length;
                   char **h_addr_list; }
#define h_addr h_addr_list[0];
```

- int **shutdown**(int fd, int howto)

  SHUT_RD, SHUT_WR, SHUT_RDWR
- int **getsockopt**(int fd, int level, int optname, void *optval, int *optlen)
- int **setsockopt**(int fd, int level, int optname, void *optval, int optlen)

  level = SOL_SOCKET, IPPROTO_IP, IPPROTO_TCP,

Interfejs gniazd – wybrane opcje

SOL_SOCKET:

- SO_BROADCAST
- SO_ERROR (pobierz błąd i wyjdź)
- SO_KEEPALIVE (2h!)
- SO_LINGER (struct linger {l_onoff, l_time})
- SO_OOBINLINE
- SO_TYPE
- SO_RCVBUF
- SO_SNDBUF
- SO_RCVLOWAT (min. dla select)
- SO_SNDLOWAT
- SO_REUSEADDR

IP_PROTOIP:

- IP_HDRINCL
- IP_TTL
- IP_TOS

IP_PROTOTCP:

- TCP_KEEPALIVE
- TCP_MAXSEG
- TCP_MAXRT
- TCP_NODELAY