5. Opcje gniazd

```
int getsockopt(int socket,     // deskryptor gniazda
               int level,      // kto ma przetwarzać opcję
               int optName,    // nazwa opcji
               void *optVal,   // wartość opcji
               unsigned int *optLen);
                               // rozmiar zmiennej z opcją

int setsockopt(int socket,     // deskryptor gniazda
               int level,      // kto ma przetwarzać opcję
               int optName,    // nazwa opcji
               const void *optVal,  // wartość opcji
               unsigned int optLen);
                               // rozmiar zmiennej z opcją
```

W przypadku poprawnego wykonania funkcje zwracają 0, w przypadku błędu -1 i kod błędu w zmiennej `errno`.

- Opcje mogą dotyczyć różnych poziomów oprogramowania sieciowego (parametr `level`):
  `SOL_SOCKET` - oprogramowanie poziomu gniazd - dotyczy wszystkich gniazd
  `IPPROTO_IP` - oprogramowanie IPv4
  `IPPROTO_IPV6` - oprogramowanie IPv6
  `IPPROTO_TCP` - oprogramowanie TCP

- Opis opcji:
  ```
  man 7 socket
  man 7 tcp
  man 7 ip
  ```

- Dwa typy opcji:
  - opcje, które włączają lub wyłączają pewną właściwość
  - opcje, które pobierają lub przekazują specjalne wartości

Przykłady opcji

Nazwa		Typ	Wartość
Poziom **SOL_SOCKET**			
SO_BROADCAST	zezwolenie na wysyłanie w trybie rozgłaszania	int	0, 1
SO_KEEPALIVE	testowanie okresowe, czy połączenie żyje	int	0, 1
SO_LINGER	zwlekanie z zamykaniem, jeśli w buforze są dane do wysłania	struct linger	czas
SO_RVCBUF	rozmiar bufora odbiorczego	int	bajty
SO_SNDBUF	rozmiar bufora wysyłkowego	int	bajty
SO_RCVLOWAT	znacznik dolnego ograniczenia bufora odbiorczego	int	bajty
SO_SNDLOWAT	znacznik dolnego ograniczenia bufora wysyłkowego	int	bajty
SO_RCVTIMEO	czas oczekiwania na pobranie	struct timeval	czas
SO_SNDTIMEO	czas oczekiwania na wysłanie	struct timeval	czas
SO_REUSEADDR	zezwolenie współdzielenie przez dwa gniazda pary adres lokalny port	int	0, 1
SO_TYPE	pobranie typu gniazda (tylko getsockname))	int	liczba
SO_OOBLINE	wykorzystywane podczas przetwarzania danych poza pasmowych	int	0,1

- Gniazda połączone TCP dziedziczą niektóre opcje po gnieździe nasłuchującym. Należą do nich SO_KEEPALIVE, SO_LINGER, SO_RVCBUF, SO_SNDBUF.

- **Opcja `SO_BROADCAST`**

```
# Nadawca

#include <stdio.h>      /* printf(), fprintf() */
#include <sys/socket.h> /* socket(), bind() */
#include <arpa/inet.h>  /* sockaddr_in */
#include <stdlib.h>     /* atoi() */
#include <string.h>     /* memset() */
#include <unistd.h>     /* close() */


int main(int argc, char *argv[])
{
  int gniazdo;
  struct sockaddr_in rozglAdr;
  char *rozglIP;
  unsigned short rozglPort;
  char *tekst;
  int rozglaszanie;
  unsigned int tekstDl;

  if (argc < 4) {
     fprintf(stderr,"Uzycie:  %s <Adres IP> <Port> <Tekst>\n",
                              argv[0]);
     exit(1);
  }

  rozglIP = argv[1];          /* adres rozgloszeniowy */
  rozglPort = atoi(argv[2]);  /* port rozgloszeniowy */
  tekst = argv[3];            /* tekst rozglaszany */

  if ((gniazdo= socket(PF_INET, SOCK_DGRAM, IPPROTO_UDP)) < 0)
    { perror("socket()"); exit(1); }

  rozglaszanie = 1;
  if (setsockopt(gniazdo, SOL_SOCKET, SO_BROADCAST,
                 &rozglaszanie, sizeof(rozglaszanie)) < 0)
    { perror("setsockopt()"); exit(1); }

  memset(&rozglAdr, 0, sizeof(rozglAdr));
  rozglAdr.sin_family = AF_INET;
  rozglAdr.sin_addr.s_addr = inet_addr(rozglIP);
  rozglAdr.sin_port = htons(rozglPort);

  tekstDl= strlen(tekst);

for (;;)
  {
   /* Rozglaszaj co 3 sekundy */
    if (sendto(gniazdo,tekst,tekstDl,0,
               (struct sockaddr *)&rozglAdr,
                sizeof(rozglAdr)) != tekstDl) {     perror("sendto() wyslal
inna liczbe bajtow niz powinien");
      exit(1); }
      sleep(3);
  }
}
```

```
# Odbiorca
#
#include <stdio.h>      /* printf(), fprintf() */
#include <sys/socket.h> /* socket(), connect(), sendto(), recvfrom() */
#include <arpa/inet.h>  /* sockaddr_in, inet_addr() */
#include <stdlib.h>     /* atoi() */
#include <string.h>     /* memset() */
#include <unistd.h>     /* close() */

#define MAXTEKST 255 /* najdluszy odbierany tekst */

int main(int argc, char *argv[])
{
  int gniazdo;
  struct sockaddr_in rozglAdr;
  unsigned int rozglPort;
  char tekst[MAXTEKST+1];
  int tekstDl;

  if (argc != 2)    {
     fprintf(stderr,"Uzycie: %s <Port rozgloszeniowy>\n",
             argv[0]);
     exit(1);
  }

  rozglPort = atoi(argv[1]);   /* port rozgloszeniowy */
  if ((gniazdo= socket(PF_INET, SOCK_DGRAM, IPPROTO_UDP)) < 0)
  { perror("socket()"); exit(1); }

  memset(&rozglAdr, 0, sizeof(rozglAdr));
  rozglAdr.sin_family = AF_INET;
  rozglAdr.sin_addr.s_addr = htonl(INADDR_ANY);
  rozglAdr.sin_port = htons(rozglPort);

  if (bind(gniazdo,(struct sockaddr *)&rozglAdr,
                   sizeof(rozglAdr)) < 0)
  { perror("bind()"); exit(1); }

  if ((tekstDl = recvfrom(gniazdo, tekst, MAXTEKST, 0,
                          NULL, 0)) < 0)
  { perror("recvfrom()"); exit(1); }

  tekst[tekstDl] = '\0';
  printf("Otrzymano : %s\n", tekst);

  close(gniazdo);
  exit(0);
}
```

- **Opcja `SO_REUSEADDR`**

```
  int serwGniazdo, wynik;
  struct sockaddr_in serwAdr;
  unsigned short serwPort;
  int opcja;

  serwGniazdo = socket(PF_INET, SOCK_STREAM, 0);

  opcja=1;
  wynik = setsockopt(sock,SOL_SOCKET,SO_REUSERADDR,
                   (void *)&opcja, sizeof(opcja));

  memset(serwAdr, 0, sizeof(serwAdr));
  echoSerwAdr.sin_family = AF_INET;
  echoSerwAdr.sin_addr.s_addr = htonl(INADDR_ANY);
  echoSerwAdr.sin_port = htons(serwPort);

/* Przypisz gniazdu lokalny adres */
  wynik =bind(serwGniazdo,(struct sockaddr *) &serwAdr,
         sizeof(serwAdr));
```

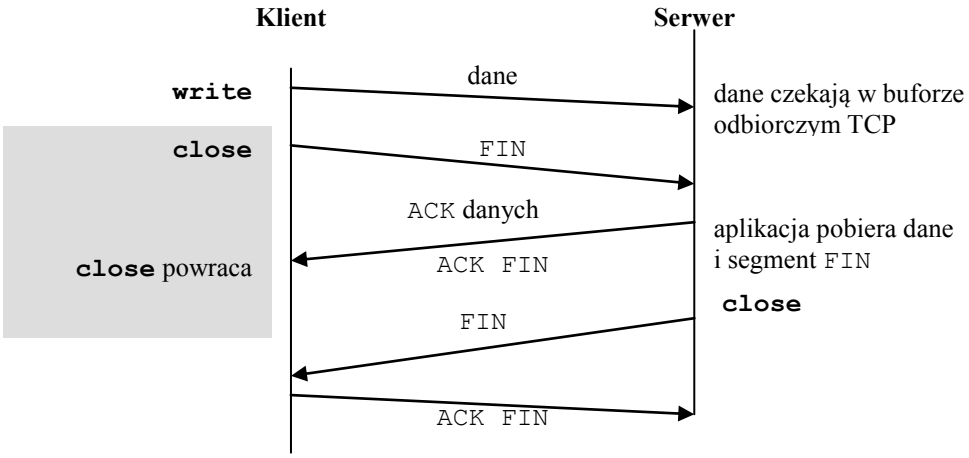
- **Opcja `SO_LINGER`**

```
struct linger {
  int l_onoff;   /* 0 - wyłączone, niezero - włączone */
  int l_linger;  /* czas zwlekania */
};
```

Jeśli:

- `l_onoff` jest równe 0 - ignorowana jest druga składowa i działanie funkcji `close` pozostaje niezmienione
- `l_onoff` jest różne od 0, `l_linger` jest równe 0 - połączenie zostanie natychmiast zerwane przez warstwę TCP
- `l_onoff` jest różne od 0, `l_linger` jest różne od 0 - proces będzie uśpiony dopóty, dopóki albo wszystkie dane będą wysłane i nadejdzie potwierdzenie od partnera, albo upłynie czas zwlekania (ang. *linger*).
- Jeśli wróci się z funkcji w wyniku upłynięcia czasu zwlekania, to zwrócony będzie kod błędu `EWOULDBLOCK` i wszystkie dane pozostawione w buforze wysyłkowym zostaną zniszczone.

- Włączone zwlekanie



Otrzymaliśmy potwierdzenie danych i przesłanego do partnera segmentu FIN. Nadal nie wiemy, czy aplikacja partnera przeczytała dane. Jak uzyskać tę informację?

- **Opcje `SO_RVCBUF` i `SO_SNDBUF`**

- Każde gniazdo ma bufor wysyłkowy i odbiorczy.
- Bufor odbiorczy wykorzystywany jest przez oprogramowanie warstwy TCP i UDP do przechowywania danych zanim przeczyta je aplikacja.
  - Wielkość bufora odbiorczego TCP jest równa rozmiarowi okna oferowanego partnerowi..
  - W przypadku UDP jeśli datagram nie mieści się w buforze odbiorczym gniazda, zostanie odrzucony.
- Każde gniazdo TCP ma bufor wysyłkowy. Do niego kopiowane są dane z bufora użytkowego aplikacji. Jeśli gniazdo jest gniazdem blokującym (ustawienie domyślne), powrót z funkcji `write` będzie oznaczał, że wszystkie dane z bufora aplikacji zostały umieszczone w tym buforze. Dane są usuwane z tego bufora dopiero po otrzymaniu potwierdzenia ACK.
- Gniazdo UDP nie ma bufora wysyłkowego. Posługuje się tylko jego rozmiarem do określenia maksymalnego rozmiaru datagramu, który można wysłać poprzez to gniazdo.

- Przykład: chcemy zwiększyć rozmiar bufora odbiorczego gniazda

```
int rvcBufferSize;
int sockOptSize;
sockOptSize=sizeof(rvcBufferSize);
if (getsockopt(sock,SOL_SOCKET,SO_RCVBUF,&rvcBufferSize, &sockOptSize) < 0)
  error("getsockopt");
printf("Poczatkowa wielkosc bufora: %d\n", rvcBufferSize);

/* Podwajamy wielkość bufora */
rvcBufferSize *=2;
if (setsockopt(sock,SOL_SOCKET,SO_RCVBUF,
               &rvcBufferSize,sizeof(rcvBufferSize)) < 0)
  error("setsockopt");
```

- **Opcje `SO_RCVLOWAT` i `SO_SNDLOWAT`**

- Funkcja `select` do stwierdzenia gotowości gniazda do czytania lub pisania wykorzystuje znaczniki dolnego ograniczenia bufora wysyłkowego i odbiorczego (ang. *low-water mark*).
- Znacznik dolnego ograniczenia bufora odbiorczego jest to niezbędna liczba bajtów w buforze odbiorczym gniazda potrzebna do tego aby `select` przekazała informację, że gniazdo nadaje się do pobierania danych.
- Znacznik dolnego ograniczenia bufora wysyłkowego jest to niezbędna wielkość dostępnej przestrzeni w buforze wysyłkowym gniazda potrzebna do tego aby `select` przekazała informację, że gniazdo nadaje się do wysyłania danych.
- W przypadku UDP znacznik ten oznacza górną granicę maksymalnego rozmiaru datagramów UDP, które można odsyłać do tego gniazda. Gniazdo to nie ma bufora wysyłkowego, ma tylko rozmiar bufora wysyłkowego.

- Przykład: chcemy otrzymać 48 bajtów, zanim nastąpi powrót z operacji czytania

```
int lowat;
int lowatSize;
sockOptSize=sizeof(rvcBufferSize);
int wynik;

lowat=48;
wynik=setsockopt(sock,SOL_SOCKET,SO_RCVLOWAT,&lowat,sizeof(rcvBufferSize));
```

- **Opcja `SO_KEEPALIVE`**
  - Opcja włącza i wyłącza sondowanie połączenie TCP (ang. keepalive probe). Sondowanie polega na wysyłaniu segmentu ACK, na który partner musi odpowiedzieć.