Wykład #4

Awaryjne kończenie klienta – sygnał **SIGPIPE**

- Jeśli do gniazda zamkniętego druga strona coś przyśle, to gniazdo odpowie segmentem RST.
- Jeśli proces oczekuje na odbiór danych i gniazdo otrzyma segment RST to funkcja czytająca wyjdzie z błędem ECONNRESET.
- **SIGPIPE – jest wysyłany jeśli proces wyśle coś do gniazda , które odebrało segment RST**
- Proces nie jest informowany o segmencie RST jeśli jest zablokowany na wprowadzaniu danych przez użytkownika
- Rozwiązaniem jest użycie funkcji select()/poll()/epoll()
- Jeśli serwer załamie się, to klient dowie się o tym fakcie dopiero po wysłaniu danych (a opóźnienie może być duże) – rozwiązaniem jest włączenie opcji SO_KEEPALIVE/TCP_KEEPALIVE gniazda

Trzy zasady obowiązujące przy pisaniu serwera sieciowego:

- Jeśli tworzymy procesy potomne to musimy obsłużyć sygnał SIGCHLD
- Jeśli przechwytujemy sygnały, to musimy obsługiwać zdarzenia spowodowane przerwanym działaniem funkcji systemowych
- Aby nie pozostawiać po sobie procesów w stanie „zombie”, musimy poprawnie zdefiniować procedurę obsługi sygnału SIGCHILD, używając w niej funkcji waitpid() (funkcja wait() nie spełnia wszystkich wymagań w obsłudze sygnałów)

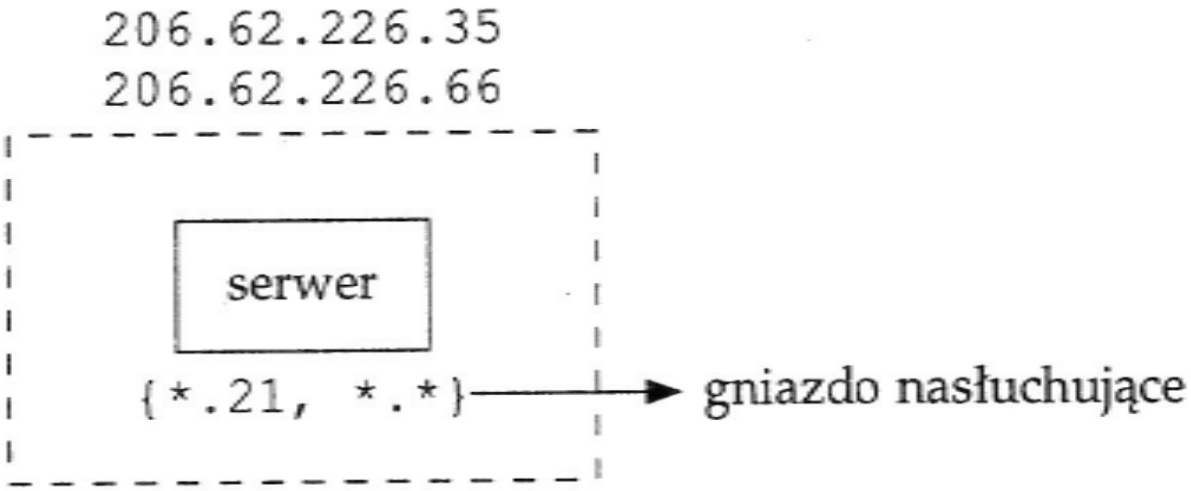
Sygnał SIGURG

- Odbiór danych pozapasmowych (OOB)
- Domyślnie sygnał SIGURG nie jest dostarczany, a bez procedury obsługi sygnału jest ignorowany
- Do odebrania wymagane dodatkowe operacje:
  - Ustawienie właściciela gniazda – operacja F_SETOWN funkcji fcntl. Nowo tworzone gniazdo nie ma właściciela. W systemie LINUX gniazdo połączone nie dziedziczy właściciela po gnieździe nasłuchującym.
  - Ustanowienie funkcji obsługi sygnału.
- Wysyłanie danych pozapasmowych – wymagane użycie flagi MSG_OOB w funkcji wysyłającej (jakie funkcje możemy do tego użyć?)

TCP
Numery Portów dla serwerów współbieżnych

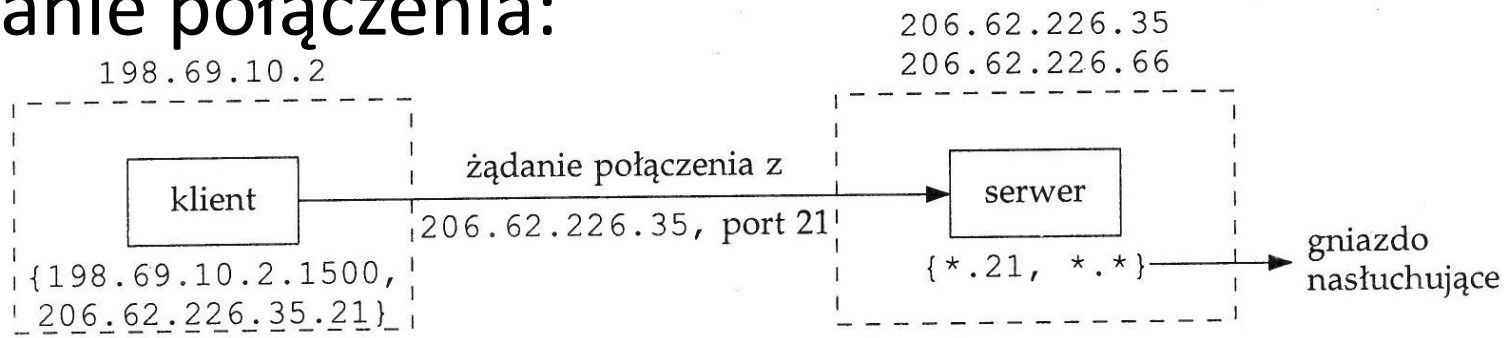
Numery portów TCP a serwery współbieżne

- Serwery współbieżne i iteracyjne
- Identyfikacja połączenia dla serwerów współbieżnych odbywa się na podstawie pary gniazdowej
- Bierne otwarcie: para gniazdowa {*:21, *:*}

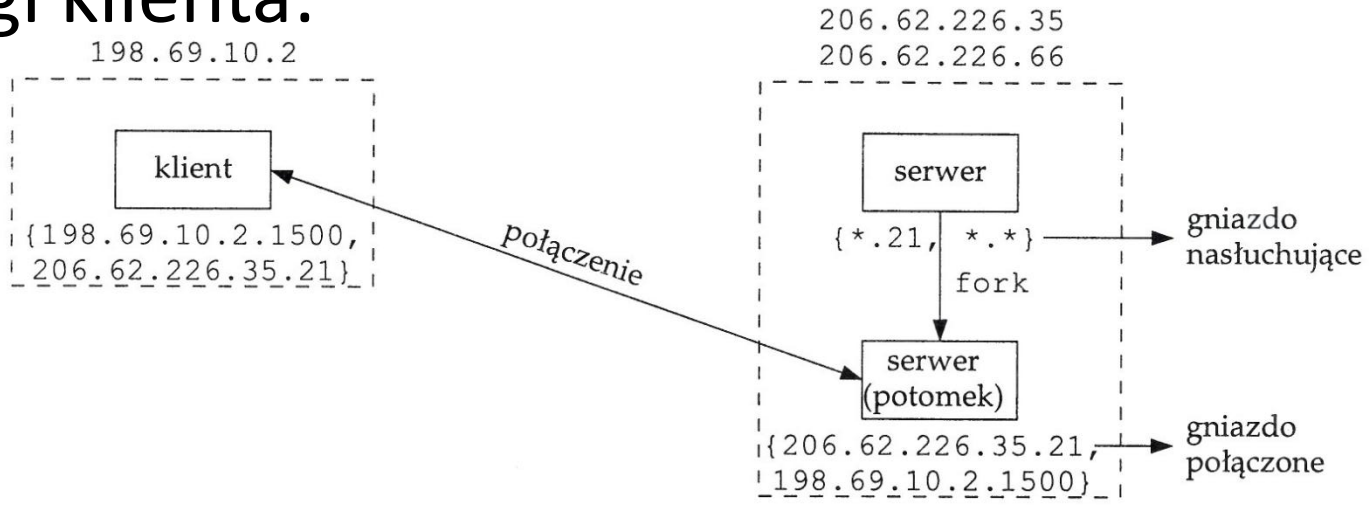


Numery portów TCP a serwery współbieżne (2)

- Żądanie połączenia:

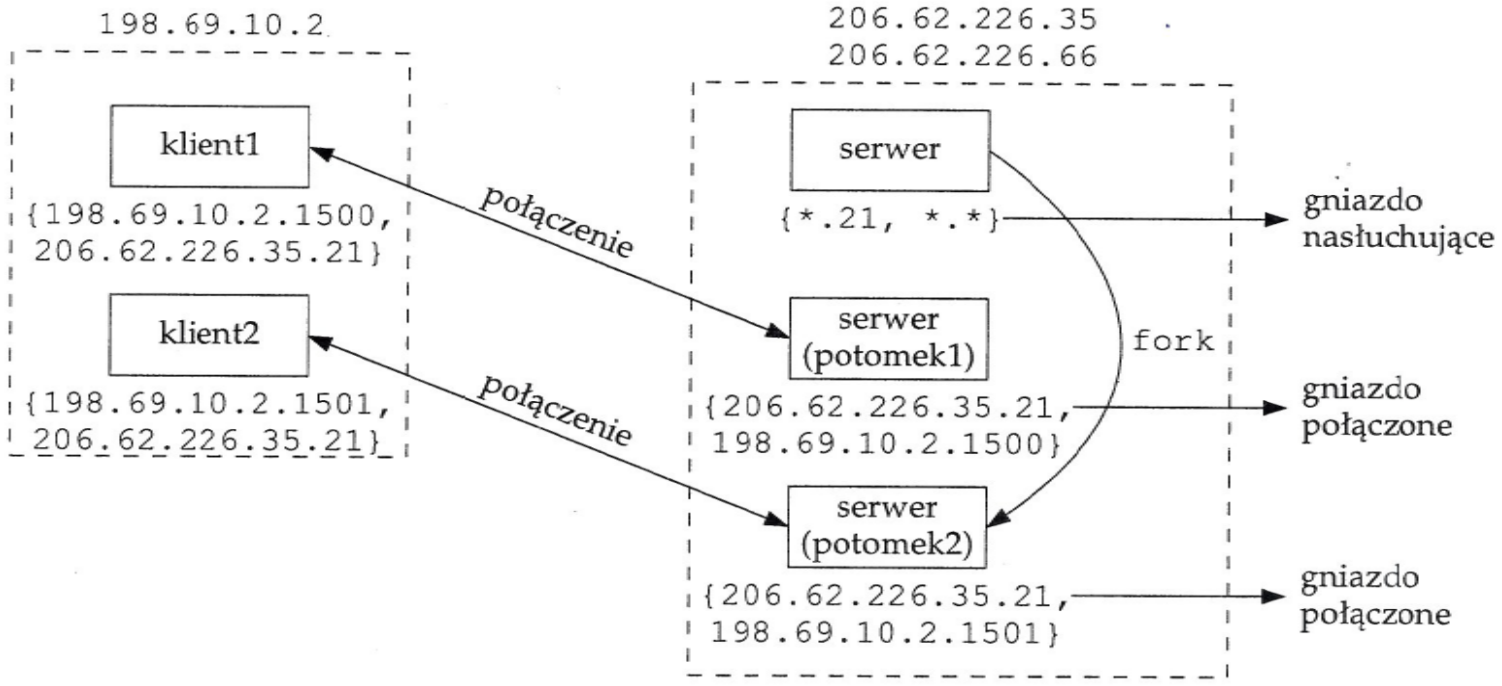


- Połączenie – tworzony jest potomek dla obsługi klienta:



Numery portów TCP a serwery współbieżne (3)

- Połączenie drugiego klienta:



- Identyfikacja procesu obsługującego na podstawie pary gniazdowej

Funkcje **wielowejściowe** (ang. **reentrant**)

- Funkcje, które mogą być bezpiecznie wywoływane jednocześnie w procesie i w procedurze obsługi sygnałów.
- Nie korzystają ze zmiennych statycznych/globalnych, które są powodem problemów
- Przykład funkcji, które nie są (nie muszą być) wielowejściowe: gethostbyname(), gethostbyname2(), getservbyport(), gethostbyaddr(), getservbyname()
- Funkcje, które nie są wielowejściowe nie powinny być używane także w wątkach
- Zmienna **errno** – zmienna globalna, która powoduje problemy dla wielowejściowości

Funkcje wielowejściowe

- Należy korzystać z odpowiedników wielowejściowych funkcji: np. dla funkcji gethostbyname(), gethostbyname2(), getservbyport(), gethostbyaddr(), getservbyname() - są to funkcje z taką samą nazwą rozszerzoną o końcówkę ‘_r’
- Własne funkcje należy pisać tak, aby **nie korzystać ze zmiennych statycznych i globalnych**

Funkcje wielowejściowe – jak używać zmiennej **errno** w obsłudze sygnału

```
1. void sig_alrm(int signo)
2. {
3.     int errno_save;
4.     errno_save = errno;/*save its value on entry */
5.     if (write( ... ) != nbytes)
6.         fprintf (stderr, „write error,
7.                          errno = %d/n”,errno);
8.     errno = errno_save; /* restore its value on
                            return */
9. }
```

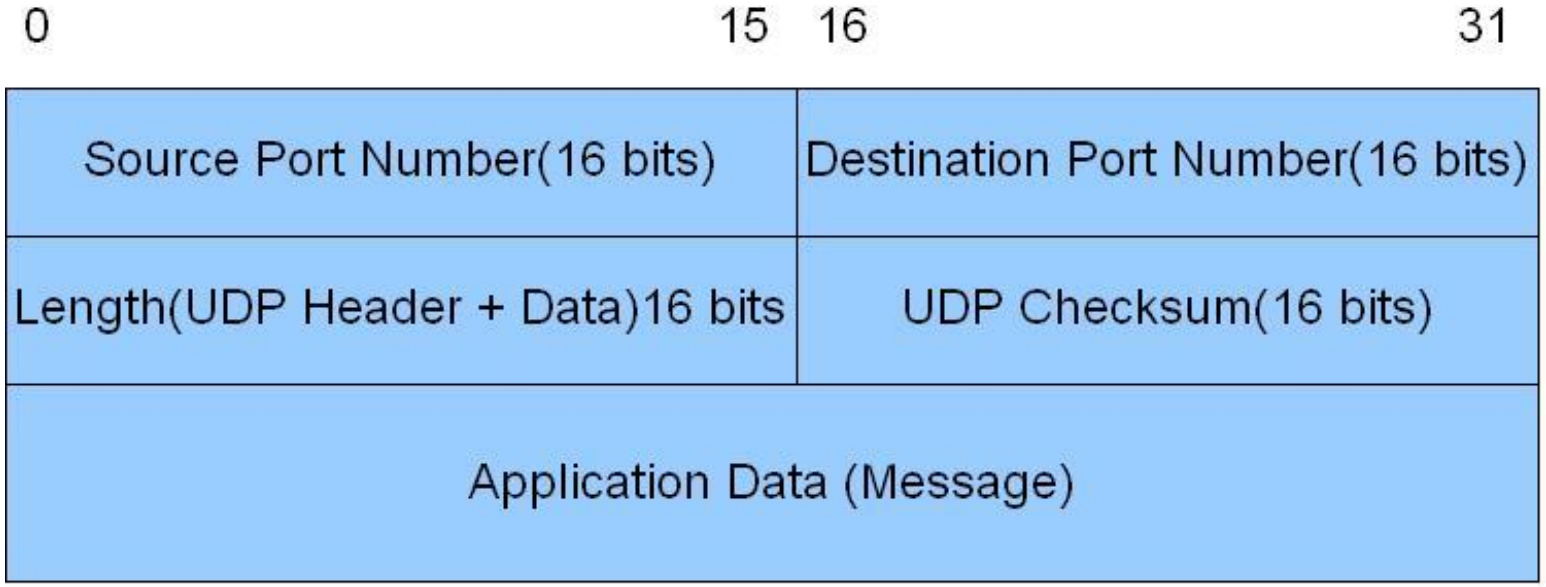
Nie powinno się używać (należy uważać) funkcji należących do standardowej biblioteki wejścia-wyjścia (wiele wersji tej biblioteki nie zapewnia wielowejściowości – ew. sprawdzenie w pliku nagłówkowym) w procedurze obsługi sygnału

API gniazd UDP

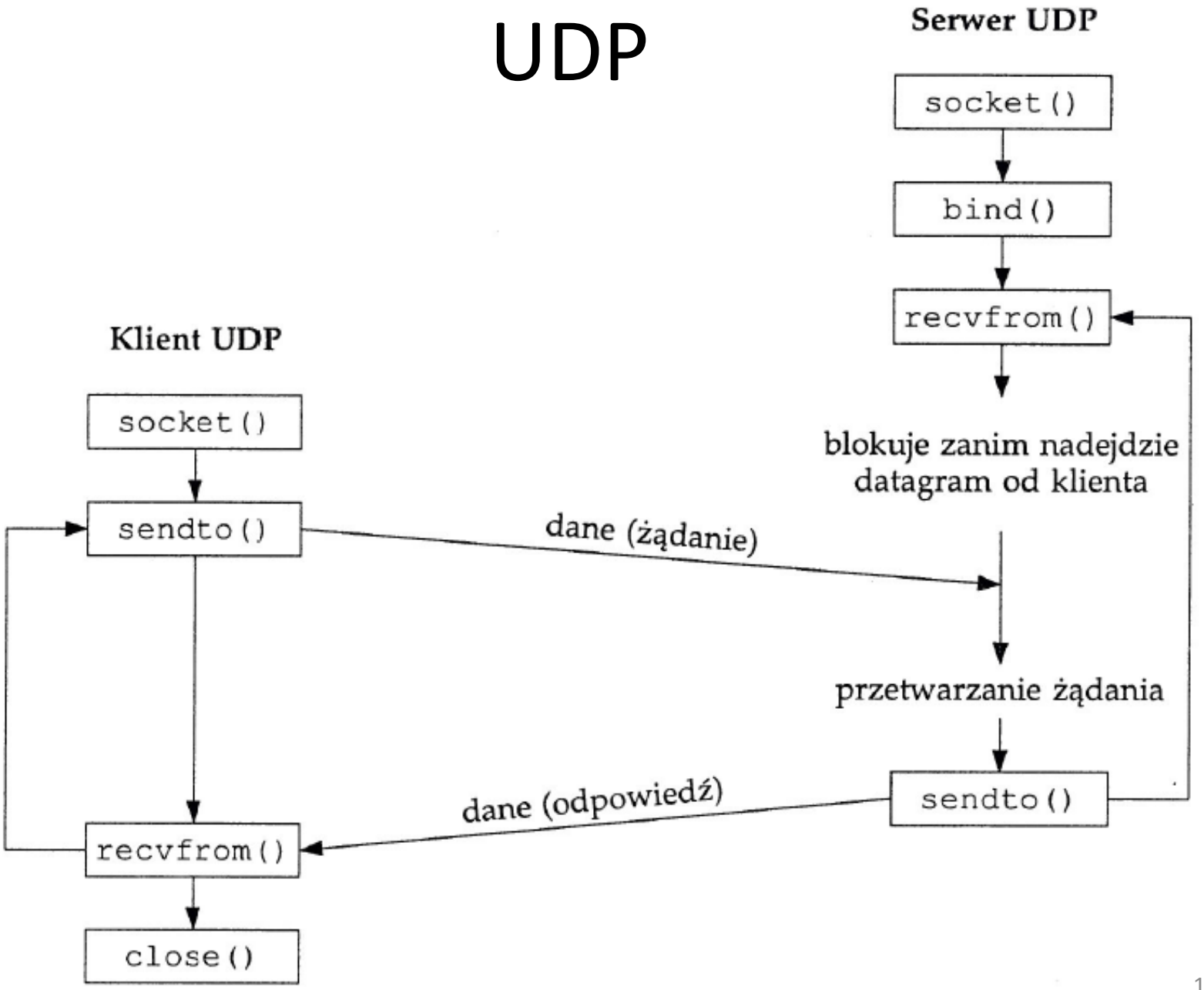
Wykład #4

Protokół UDP

- **RFC 768** (3 strony)
- Usługa bezpołączeniowa
- Przesyłanie datagramów – dane podzielone na paczki danych
- Zawodny – możliwa utrata pakietów bez żadnej informacji
- Na jednym gnieździe możliwa jednoczesna komunikacja z wieloma systemami zdalnymi



UDP


Serwer UDP
socket()
bind()
recvfrom()
blokuje zanim nadejdzie datagram od klienta
przetwarzanie żądania
sendto()
Klient UDP
socket()
sendto()
dane (żądanie)
recvfrom()
dane (odpowiedź)
close()

Opcje gniazd UDP

- Funkcje getsockopt() i setsockopt()
- Poziom - IPPROTO_UDP
- Opcja: UDP_CORK – jeśli ta opcja jest uaktywniona dane wysyłane przez gniazdo są gromadzone, a nie wysyłane. Wysłanie następuje po wyłączeniu tej opcji. Opcja charakterystyczna dla systemu LINUX od wersji jądra 2.5.44.
- Domyślne opcje protokołu UDP można także zmieniać przez zmienne w systemie /proc/sys/net/ipv4

UDP – funkcje odbierające

- read – nie otrzymamy informacji o adresie nadawcy i nie będziemy mogli odesłać informacji
- recv – podobnie jak read() ale ma dodatkowe flagi
- recvfrom() – podobnie jak funkcja accept() zwraca adres nadawcy
- recvmsg() – jak recvfrom() i dodatkowo można odczytywać opcje z nagłówka protokołów IP

UDP – funkcje odbierające

ssize_t recvfrom(int sockfd, void *buf, size_t len, int flags, struct sockaddr *src_addr, socklen_t *addrlen);

- Pierwsze trzy argumenty jak w funkcji read()
- Flagi – następny slajd
- Adres nadawcy (możemy nadać tej wartości NULL)
- Rozmiar adresu nadawcy (musimy podać rozmiar pamięci zaalokowanej na adres sieciowy), natomiast funkcja zwraca rozmiar odebranego adresu
- Funkcja może zwrócić 0 – jest to normalne zachowanie
- Może być użyta dla protokołu TCP
- Opcja gniazd: SO_RECVBUF - jeśli ustawimy zbyt mały bufor funkcja zwróci błąd EMSGSIZE

Funkcja recvfrom() - flagi

- **MSG_DONTWAIT** – żądanie operacji nieblokującej (bez potrzeby ustawienia opcji gniazda)
- **MSG_OOB** – żądanie otrzymania danych pozapasmowych (**tylko TCP**)
- **MSG_PEEK** – podgląd nadchodzącego komunikatu
- **MSG_WAITALL** – czytanie wszystkich danych, zgodnie z parametrem count
- **MSG_TRUNC** – informowanie, że dane zostały obcięte z powodu zbyt małego bufora odbiorczego – funkcja zwraca liczbę faktycznie odebranych danych przez gniazdo a nie liczbę danych skopiowanych do bufora odbiorczego

Funkcja recvfrom() – przypadek użycia

```
int gniazdo;
char bufor[MAX_MSG_LEN];
struct sockaddr_in repl_addr;
socklen_t len = sizeof(repl_addr);

if (recvfrom(gniazdo, bufor, MAX_MSG_LEN, 0,
   (struct sockaddr*)&repl_addr, &len) < 0)
{
        perror("recvfrom error");
}
```

Funkcja recvmsg()

- Funkcja umożliwia odbieranie zwykłych danych oraz dodatkowych informacji związanych z odbieranym datagramem
- Można używać zarówno dla UDP jak i TCP
- Jednym z argumentów jest struktura typu msghdr, która służy do przechowywania zarówno danych jak i dodatkowych informacji

Funkcja recvmsg()

- ssize_t recvmsg(int sockfd, struct msghdr *msg, int flags);

- Argumenty
  - Deskryptor gniazda
  - Wskaźnik do struktury typu msghdr, za pomocą której przekazywane są dane i dodatkowe informacje
  - Flagi

Flagi w funkcji recvmsg()

- Obowiązują flagi dla funkcji recvfrom()
- MSG_ERRQUEUE (od Linux 2.2) – pobieranie nieobsłużonych błędów z gniazda przez dane dodatkowe
- MSG_CMSG_CLOEXEC – zamknięcie deskryptora
- MSG_CTRUNC – zwracanie rzeczywistego rozmiaru danych dodatkowych, gdy bufor w aplikacji jest mniejszy niż dane dodatkowe

Struktury msghdr i iovec

```
struct msghdr {
        void        *msg_name;        /* opcjonalny adres*/
        socklen_t    msg_namelen;     /* rozmiar adresu*/
        struct iovec *msg_iov;        /* tablica z danymi */
        size_t       msg_iovlen;      /* liczba element.  msg_iov */
        void        *msg_control;     /* dane dodatkowe (cmsghdr), */
        size_t       msg_controllen;  /* rozmiar  danych dodatk.*/
        int          msg_flags;       /* flagi  odbiorcze*/
};

struct iovec {                        /* Element tablicy z danymi*/
        void  *iov_base;              /* Adres początkowy*/
        size_t iov_len;               /* Rozmiar danych*/
};
```

Flagi struktury msghdr dla funkcji recvmsg()

- MSG_TRUNC – dane zostały obcięte, ponieważ datagram był większy niż bufor odbiorczy
- MSG_CTRUNC – dane dodatkowe zostały obcięte, ponieważ datagram był większy niż bufor odbiorczy dla danych dodatkowych
- MSG_OOB – zostały odebrane dane pozapasmowe
- MSG_ERRQUEUE – dane nie zostały odebrane, ale zostały odebrane błędy z gniazda
- MSG_EOR - zakończenie rekordu danych (dla gniazd typu SOCK_SEQPACKET)

Struktura cmsghdr

```
struct cmsghdr {
    socklen_t     cmsg_len;     /* data byte count, including hdr */
     int          cmsg_level;       /* originating protocol */
     int          cmsg_type;        /* protocol-specific type */
    unsigned char cmsg_data[]; /* followed by */
 };
```

cmsg_level – Dla IP: IPPROTO_IP lub IPPROTO_IPV6

cmsg_type – rodzaj opcji, np. IPV6_PKTINFO

Operacje na strukturze cmsghdr

- CMSG_FIRSTHDR() - zwraca wskaźnik do pierwszej struktury cmsghdr w buforze danych informacyjnych (sygnalizacyjnych) przekazanych w strukturze msghdr.
- CMSG_NXTHDR() - zwraca następną poprawną strukturę cmsghdr za przekazaną strukturą cmsghdr. Zwraca NULL jeśli w buforze nie ma wystarczającego miejsca.
- CMSG_ALIGN() - podaje wyrównanie na podstawie podanej długości. Wartość stała. Służy do oszacowania wymaganego miejsca w pamięci.
- CMSG_SPACE() - zwraca liczbę bajtów zajmowanych przez dany element informacyjny.
- CMSG_DATA() - zwraca wskaźnik do danych w strukturze cmsghdr.
- CMSG_LEN() - zwraca liczbę bajtów, ile należy wpisać do zmiennej cmsg_len (składowa struktury cmsghdr) uwzględniając możliwe wyrównanie bajtów. Jako argument podaje się rozmiar danych.

Opcje gniazd

- Opcje gniazd ustawiamy za pomocą funkcji setsockopt() – jeśli ustawimy daną opcję, to wszystkie datagramy wysyłane z tego gniazda będą miały ustawioną tą opcję
- Opcje gniazd pobieramy za pomocą funkcji getsockopt() – funkcja zwraca dokładnie to (albo prawie dokładnie), co ustawiła funkcja setsockopt() z jednym wyjątkiem – dla TCP dla opcji **source route** pobierane jest to co przyszło w datagramie od drugiej strony połączenia
- Do pobierania opcji z nagłówków datagramów przychodzących służy funkcja recvmsg(), ale wcześniej należy powiadomić jądro, że ma dołączać daną opcję do odbieranych danych za pomocą funkcji setsockopt()

Co należy zrobić, aby odczytać informacje z nagłówka datagramu UDP (dotyczy także gniazd surowych – typu RAW):

1. Ustawić opcje, które chcemy odbierać funkcją setsockopt()
2. Wywołać funkcję recvmsg(), która oprócz danych datagramu przekazuje żądane informacje
3. Odczytać za pomocą makr żądaną opcję

Przykład pobierania adresu docelowego z przychodzącego datagramu dla IPv4 (multihoming)

- Ustawiamy opcje gniazd
  - Wywołujemy funkcję setsockopt() z opcją IP_PKTINFO
- Odbieramy datagram za pomocą funkcji recvmsg()
- Odszukujemy opcję IP_PKTINFO za pomocą makr i odczytujemy adres ze struktury in_pktinfo

Struktura **in_pktinfo** dla opcji **IP_PKTINFO**

```
struct in_pktinfo {
    unsigned int     ipi_ifindex;  /* Interface index */
    struct in_addr     ipi_spec_dst; /* Local address */
    struct in_addr    ipi_addr;     /* Header Destination
                                       address */
};
```

Przykład

```
struct msghdr msg;      struct cmsghdr *cmsg;
struct in_addr  addr;      int on=1,  sd;
……
setsockopt(sd, IPPROTO_IP, IP_PKTINFO, &on, sizeof(on))
……
recvmsg(sd, &msg, flags);
……
for(cmsg = CMSG_FIRSTHDR(&msg);     cmsg != NULL;
    cmsg = CMSG_NXTHDR(&msg, cmsg)) {
            if (cmsg->cmsg_level == IPPROTO_IP && cmsg->cmsg_type ==
    IP_PKTINFO) {
                        addr = ((struct in_pktinfo*)CMSG_DATA(cmsg))->ipi_addr;
                        printf("message received on address %s\n",
                                    inet_ntoa(addr));
 }   }
```

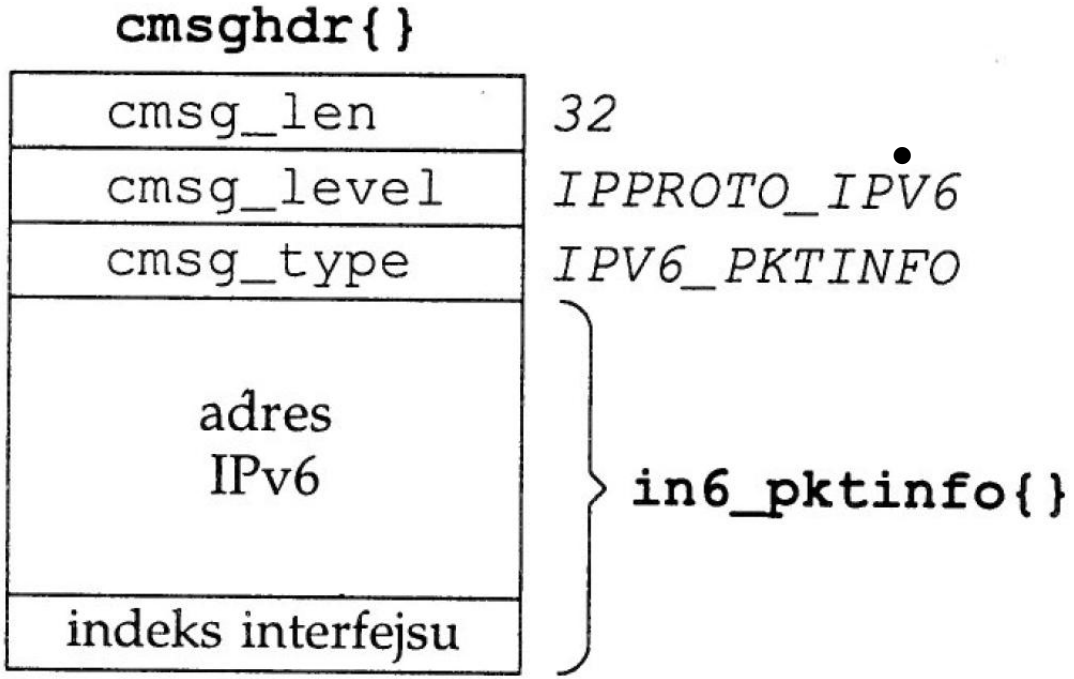
Opcje pobierane z datagramu IP

- **IP_PKTINFO** (since Linux 2.2) Pass an IP_PKTINFO ancillary message that contains a pktinfo structure that supplies some information about the incoming packet. This only works for datagram oriented sockets. The argument is a flag that tells the socket whether the IP_PKTINFO message should be passed or not. The message itself can only be sent/retrieved as control message with a packet using recvmsg(2) or sendmsg(2).
- **IP_RECVOPTS** (since Linux 2.2) Pass all incoming IP options to the user in a IP_OPTIONS control message. The routing header and other options are already filled in for the local host. Not supported for SOCK_STREAM sockets.
- **IP_RECVORIGDSTADDR** (since Linux 2.6.29) This boolean option enables the IP_ORIGDSTADDR ancillary message in recvmsg(2), in which the kernel returns the original destination address of the datagram being received. The ancillary message contains a struct sockaddr_in.
- **IP_RECVTOS** (since Linux 2.2) If enabled the IP_TOS ancillary message is passed with incoming packets. It contains a byte which specifies the Type of Ser vice/ Precedence field of the packet header. Expects a boolean integer flag.
- **IP_RECVTTL** (since Linux 2.2) When this flag is set, pass a IP_TTL control message with the time to live field of the received packet as a byte. Not sup ported for SOCK_STREAM sockets.

Przykład pobierania adresu docelowego z przychodzącego datagramu dla IPv6

- Ustawiamy opcje gniazd
  - Wywołujemy funkcję setsockopt() i ustawiamy opcję **IPV6_RECVPKTINFO**
- Odbieramy datagram za pomocą funkcji recvmsg(sd, msg, flags)
- Odszukujemy opcję IPV6_PKTINFO ze struktury msg za pomocą makr i odczytujemy adres ze struktury in6_pktinfo

Struktura pola msg_control dla opcji IPV6_PKTINFO

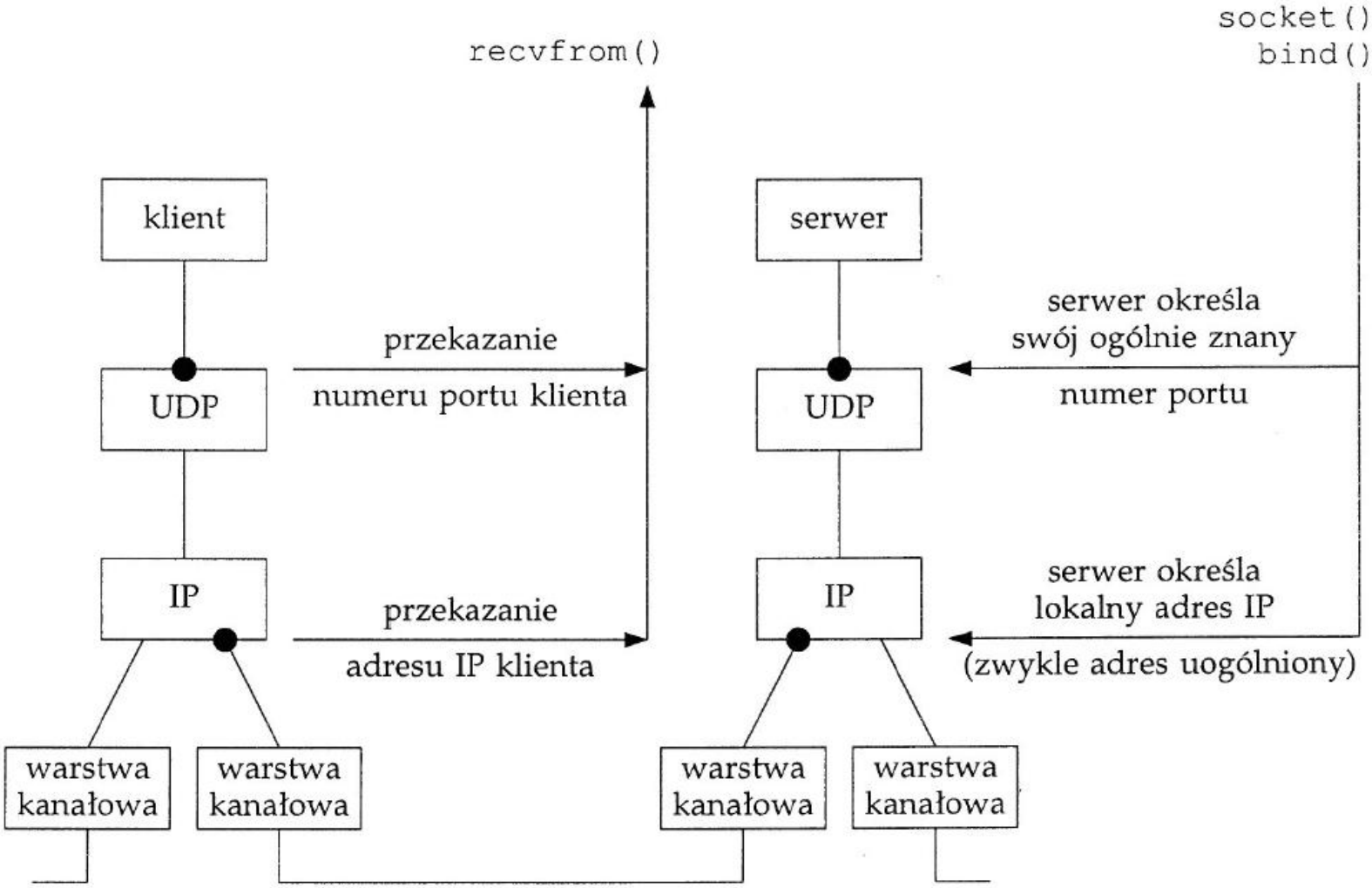



```
struct in6_pktinfo
{
    struct in6_addr
    ip6_addr;
    int ipi6_ifindex;
};
```

Opcje pobierane z datagramu IPv6

- **IPV6_RECVPKTINFO** (od Linux 2.6.14) Ustawia możliwość odbierania informacji sterujących typu IPV6_PKTINFO dla przychodzących datagramów. Format informacji sterujących jest zdefiniowany przez strukturę in6_pktinfo (RFC 3542). Opcja dostępna tylko dla gniazd SOCK_DGRAM lub SOCK_RAW.
- **IPV6_RTHDR, IPV6_AUTHHDR, IPV6_DSTOPTS, IPV6_HOPOPTS, IPV6_FLOWINFO, IPV6_HOPLIMIT** – powyższe opcje umożliwiają uzyskanie informacji zawartych w nagłówkach rozszerzeń z otrzymywanych pakietów. Opcja dostępna tylko dla gniazd SOCK_DGRAM lub SOCK_RAW.

UDP
Uzyskiwanie informacji o połączeniu



Informacje o adresie i porcie uzyskiwane przez **serwer**

Informacje z datagramów IP	Serwer TCP	Serwer UDP (gnia. niepołączone)
Adres źródłowy	accept() getpeername()	recvfrom()
Numer portu źródłowego	accept() getpeername()	recvfrom()
Adres docelowy	getsockname()	recvmsg()
Numer portu docelowego	getsockname()	getsockname()

UDP – funkcje wysyłające

- Gniazdo niepołączone - musimy określić adres pod który wysyłamy
- sendto()
- sendmsg()
- send() – tylko w połączeniu z funkcją connect()
- write() – tylko w połączeniu z funkcją connect()
- Można wysyłać datagramy UDP o rozmiarze 0.

UDP – funkcja sendto()

```
ssize_t sendto(int s, const void *buf, size_t len, int
flags, const struct sockaddr *to, socklen_t tolen);
```

- Argumenty funkcji :
  - int sockfd - deskryptor gniazda, do którego chcemy wysłać
  - const void * buf – bufor z danymi, które będą wysłane
  - size_t len – rozmiar danych do wysłania
  - int flags – flagi
  - const struct sockaddr * dest_addr - wskaźnik do struktury, w której znajdują się informacje o przeznaczeniu pakietu (adres IP oraz port)
  - socklen_t addrlen - wielkość struktury sockaddr addrlen
- Wartość zwracana :
  - w przypadku powodzenia - liczba wysłanych bajtów
  - w przypadku błędu – -1
- Jeśli rozmiar danych jest większy od (MTU – rozmiar nagłówków IP i UDP) nastąpi fragmentacja datagramu

UDP – funkcja sendto() wybrane flagi

- **MSG_CONFIRM** – dla gniazd DGRAM i RAW, wymuszenie uaktualniania odwzorowania adresów IP<->MAC
- **MSG_DONTROUTE** – pakiet przeznaczony dla sieci lokalnej
- **MSG_DONTWAIT** – operacja nieblokowalna (bez potrzeby ustawienia opcji gniazda) – zwracane błędy EAGAIN lub EWOULDBLOCK
- **MSG_OOB** – dane pozapasmowe
- **MSG_MORE** – (LINUX) - jeśli ta flaga jest uaktywniona dane wysyłane do gniazda są gromadzone, a nie wysyłane. Wysłanie następuje po wyłączeniu tej fagi.

UDP – funkcja sendto() - przykład użycia :

```
int gniazdo;
char bufor[MAX_MSG_LEN];
struct sockaddr_in serwer;
…….
socklen_t len = sizeof(serwer);
int flags =  MSG_DONTWAIT | MSG_CONFIRM;

if( sendto(gniazdo, bufor,
          strlen(bufor), flags, (struct
    sockaddr*)&serwer, len) < 0 {
    perror(„sendto error”);
}
```

Funkcja sendmsg()

- ssize_t **sendmsg**(int sockfd, const struct msghdr *msg, int flags);

- Umożliwia wysyłanie dodatkowych informacji sterujących (szczególnie ważne dla UDP i IPv6)

Parametry:

- Deskryptor gniazda
- Wskaźnik do struktury typu msghdr – dane przechowywane w msg->msg_iov
- Flagi - obowiązują te same co dla funkcji sendto()

Opcje – ustawianie za pomocą funkcji sendmsg

- Możliwość ustawiania opcji dla każdego wychodzącego datagramu (w odróżnieniu od opcji dla gniazda)
- Inicjalizacja struktury msghdr i cmsghdr (odpowiedni przydział pamięci) z pomocą makr
- Ustawienie danej opcji: np. IPV6_HOPLIMIT, IPV6_FLOWINFO
- Wysłanie datagramu funkcją sendmsg()

Przykład ustawiania opcji – adres źródłowy

```
struct msghdr msg;
struct cmsghdr *cmsg;
struct in_pktinfo *pktinfo;
// after initializing msghdr & control data to
    CMSG_SPACE(sizeof(struct in_pktinfo))
cmsg = CMSG_FIRSTHDR(&msg);
cmsg->cmsg_level = IPPROTO_IP;
cmsg->cmsg_type = IP_PKTINFO;
cmsg->cmsg_len = CMSG_LEN(sizeof(struct in_pktinfo));
pktinfo = (struct in_pktinfo*) CMSG_DATA(cmsg);
pktinfo->ipi_ifindex = src_interface_index;
pktinfo->ipi_spec_dst = src_addr;
bytes_sent = sendmsg(sd, &msg, flags);
```

Adres w funkcji sendmsg()

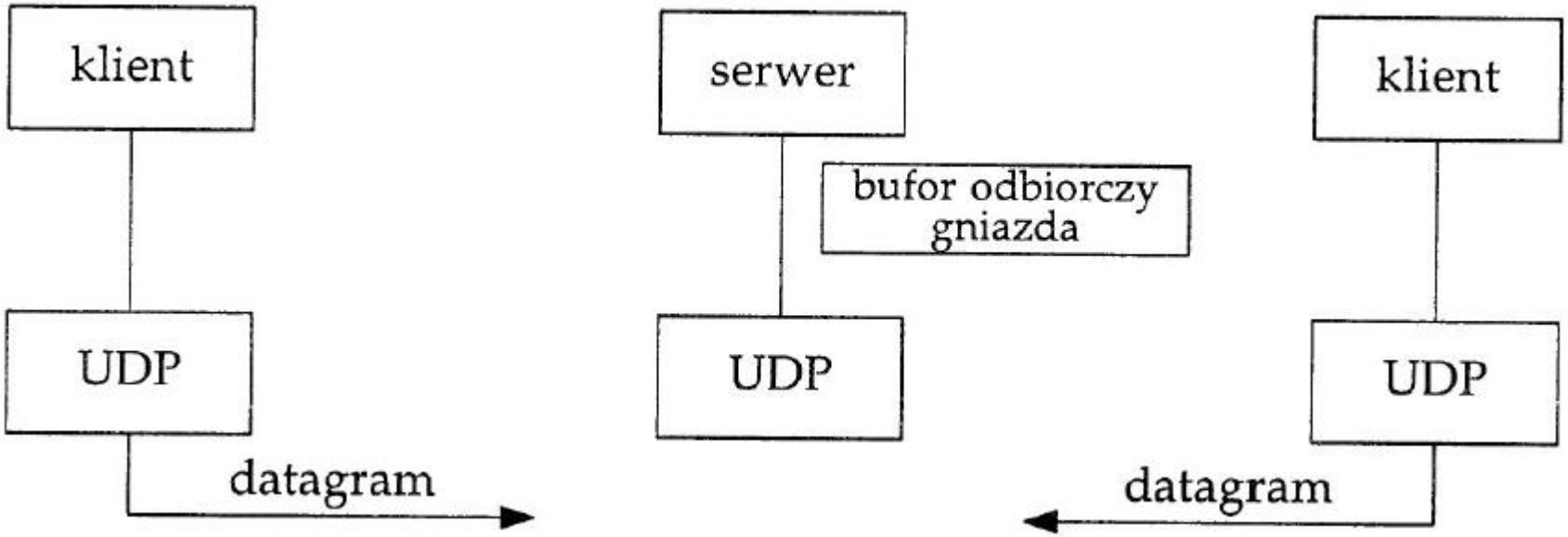
- Dla UDP obowiązkowy jeśli nie używamy funkcji connect() w programie
- Dla TCP powinien być równy NULL, w przeciwnym razie może zostać zwrócony błąd EISCONN (zależne od implementacji)
- Dla gniazda TCP, które nie jest połączone będzie zwrócony błąd ENOTCONN

Adresy w funkcjach wysyłających

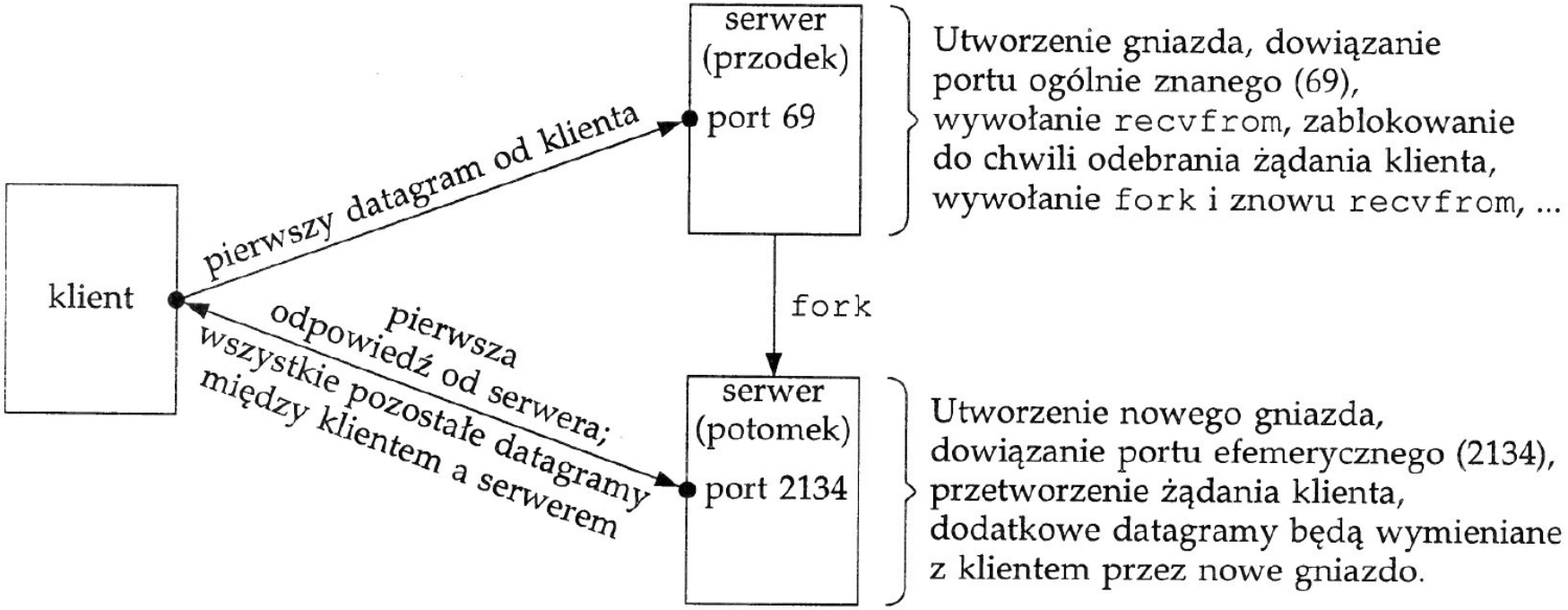
Rodzaj gniazda	write() lub send()	sendto() bez określonego adresu docelowego	sendto() z określonym adresem docelowym
Gniazdo TCP	OK	OK	EISCONN
Gniazdo UDP połączone	OK	OK	EISCONN
Gniazdo UDP niepołączone	EDESTADDRREQ	EDESTADDRREQ	OK

Usługa bezpołączeniowa – możliwe użycie **jednego gniazda** do obsługi wielu połączeń

- serwer współbieżny - problem śledzenia klientów



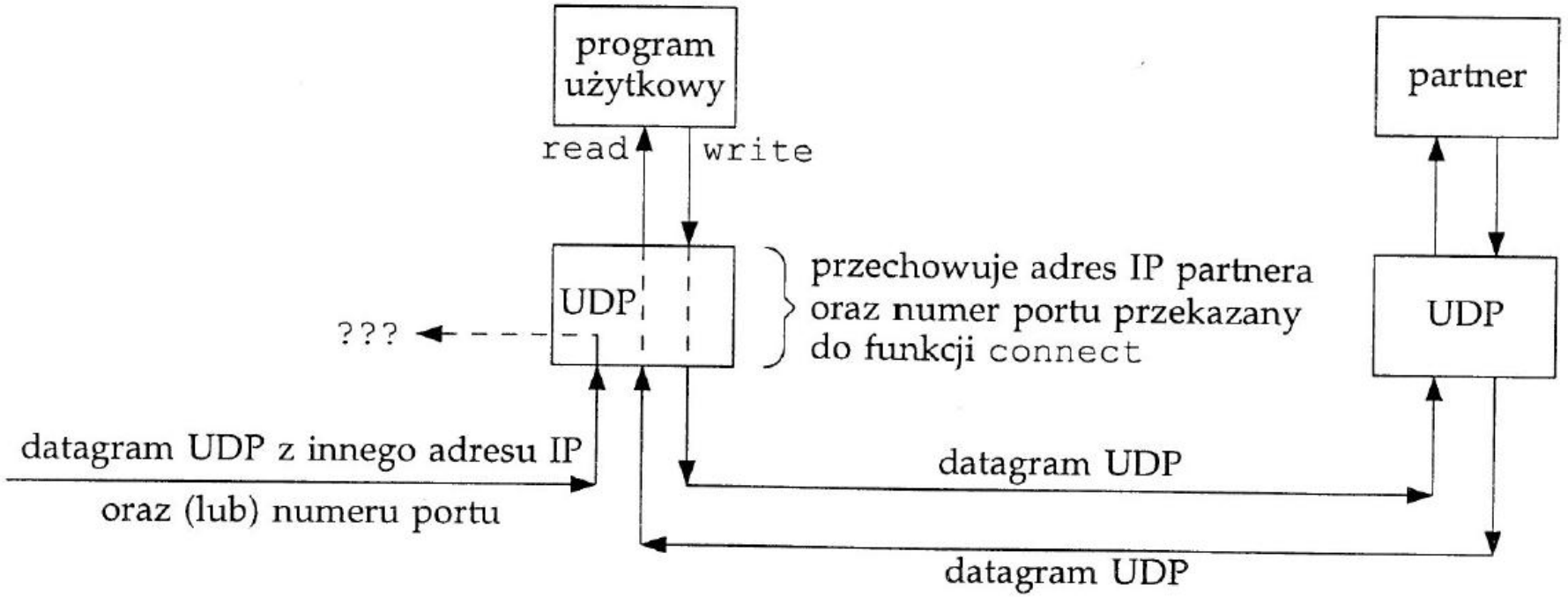
Współbieżne serwery UDP



UDP – funkcja connect() – gniazdo połączone

- Brak prawdziwego połączenia (tylko jedna strona ma gniazdo połączone)
- Można wymieniać datagramy wyłącznie z jednym systemem zdalnym
- Przy wysyłaniu datagramów nie musimy podawać adresu (można użyć funkcji write() lub send())
- Adres źródłowy i docelowy datagramów przychodzących można pobrać funkcją odpowiednio **getpeername()** i **getsockname()**
- Błędy asynchroniczne są przekazywane do aplikacji (np. brak serwera na danym porcie) - **bez gniazda połączonego błędy asynchroniczne nie są przekazywane do gniazd UDP**

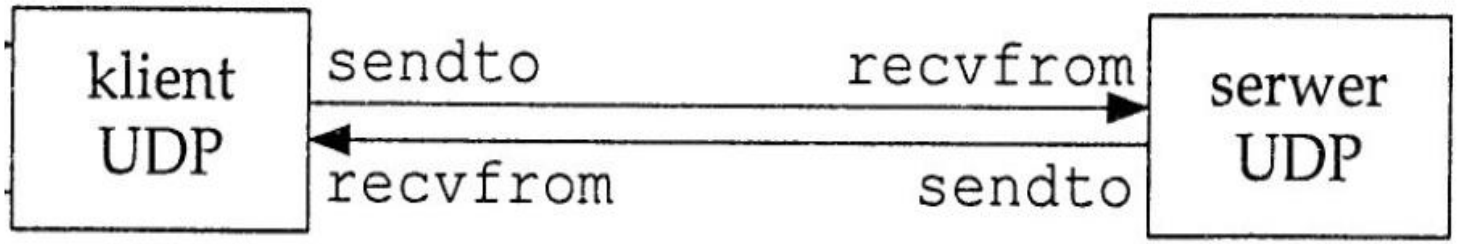
UDP – funkcja connect() – gniazdo połączone



Opcja protokołu IP dla gniazda typu UDP lub RAW tworząca kolejkę błędów

- **IP_RECVERR / IPV6_RECVERR – tworzy kolejkę błędów dla gniazda niepołączonego i możliwość ich pobierania za pomocą danych dodatkowych funkcji recvmesg().**

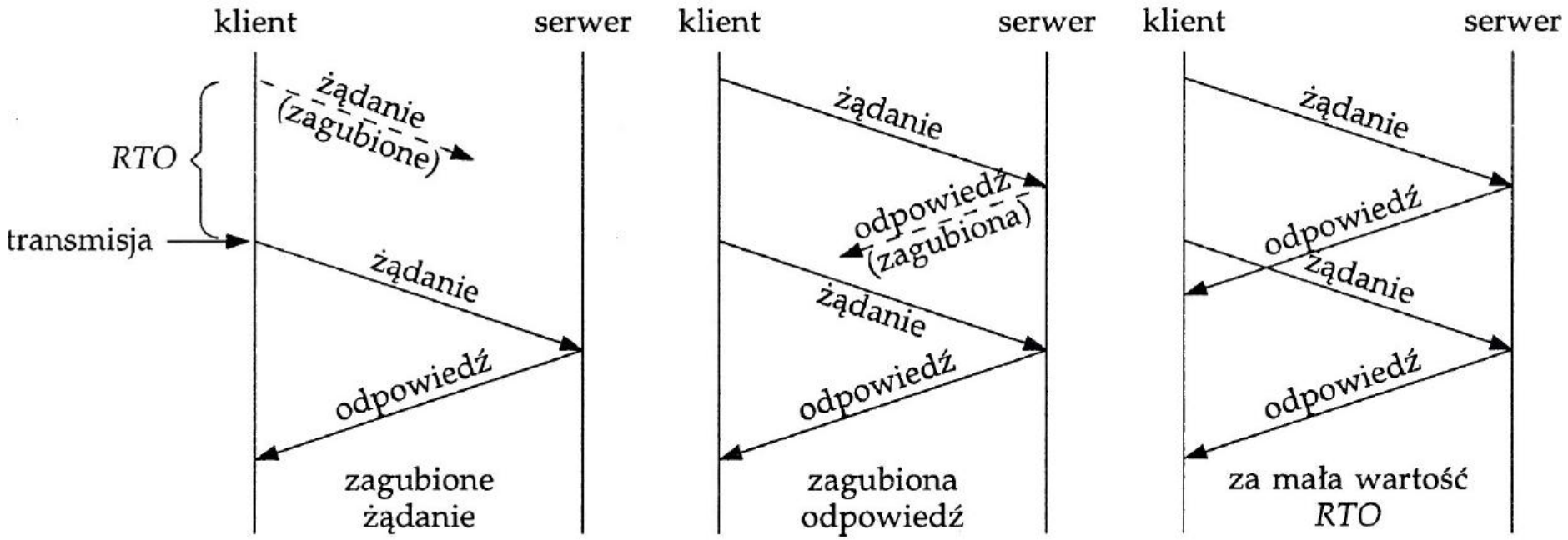
UDP – przykład serwera i klienta czasu dobowego z funkcjami sendto() i recvfrom()



daytimeudpcliv6.c

daytimeudpsrvv6.c

Zapewnienie niezawodności dla UDP – problem aplikacji/programisty



Estymacja jednostki czasu oczekiwania na ponowienie transmisji RTO

- Jacobsen [1] zaproponował następujący algorytm estymacji RTO (retransmission timeout):

  delta = zmierzony_RTT – srtt

  *srtt =srtt + g * delta*

  rttvar = rttvar + h(|delta| - rttvar)

  *RTO = srtt + 4 * rttvar*

gdzie: *g=1/8, h=1/4, srtt* – wygładzony parametr szacunkowego RTT, *rttvar* – wygładzony parametr szacunkowego odchylenia średniego

- Wymaga zaimplementowania numerów sekwencyjnych lub przesyłania czasu
- Po nieotrzymaniu odpowiedzi RTO należy zastosować wykładnicze wydłużenie czasu oczekiwania (*expotential backoff*)

Estymacja jednostki czasu oczekiwania na ponowienie transmisji RTO - referencja

[1] V. Jacobson, „Congestion Avoidance and Control”, Originally Published in: Proc. SIGCOMM '88, Vo118 No. 4, August 1988

Przykład – prosty sposób zapewnienia niezawodności dla klienta czasu dobowego

- Ustawiamy czas oczekiwania dla gniazda opcją SO_RCVTIMEO
- Wysyłamy datagram, oczekujemy na odpowiedź funkcją recvfrom() i sprawdzamy status powrotu
- Jeśli funkcja zwraca błąd związany z upływem czasu oczekiwania (EAGAIN | EWOULDBLOCK) ponawiamy wysyłanie założoną liczbę razy

UDP - podsumowanie

- Bezpołączeniowy (zawodny, brak sterowania przepływem danych)
- Funkcje do wysyłania: sendto(), sendmsg()
- Funkcje do odbierania: recvfrom(), recvmesg()
- Użycie funkcji connect():
  - Gniazdo połączone UDP (filtr, kolejka błędów asyn.)
  - Optymalizacja wydajności
- Ustawianie i pobieranie informacji z nagłówków protokołów IPv4 i IPv6
- Podstawowy protokół dla rozgłaszania (broadcast) i rozgłaszania grupowego (multicast)