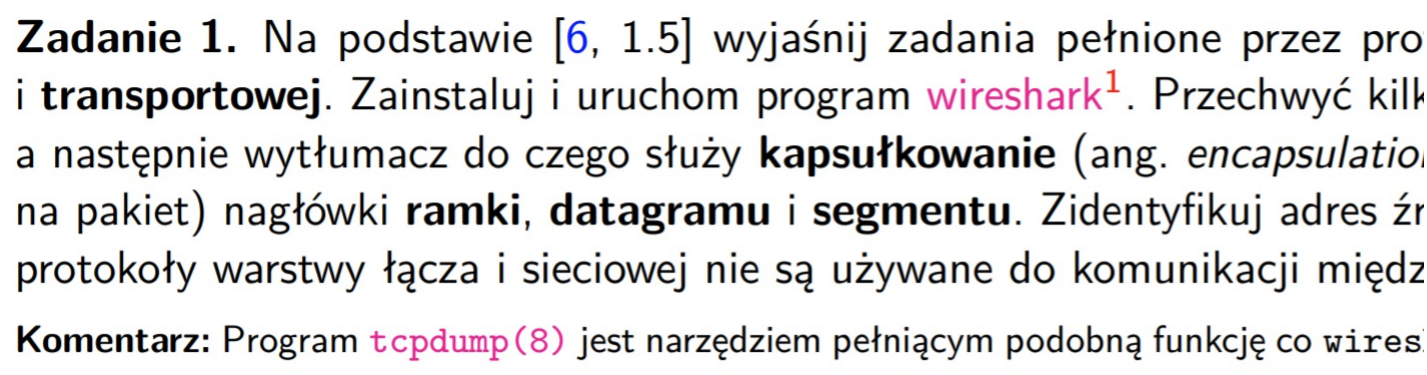
Lista 9

tags: SO

Zadanie 1



::: info **Zadania protokołów warstwy**:

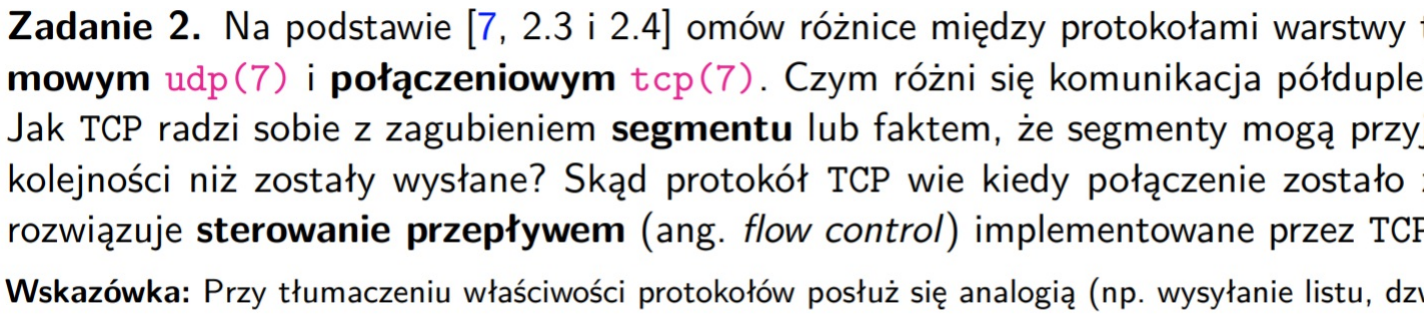
- **łącza** – na żądanie warstwy sieciowej przesyła pakiet pomiędzy parą węzłów (host/router)
- **sieciowej** – odpowiada za nawigację pakietów w sieci (tj. zna jej topologię)
- **transportowej** – odpowiada za transport segmentów danych z wykorzystaniem protokołów UDP i TCP

kapsułkowanie – pakiet z wyższej warstwy jest owijany w dodatkowe metadane z warstwy niższej :::

Czemu protokoły warstwy łącza i sieciowej nie są używane do komunikacji między procesami użytkownika?

W komunikacji międzyprocesowej nie jest nam potrzebna wiedza o strukturze ani sposobie nawigowania sieci.

Zadanie 2



udp (*user datagram protocol*):

- dane są właściwie wysyłane w jednorazowych paczkach zwanych datagramami. W ten sposób możemy użyć gniazda do nadesłania danych do pewnego serwera i natychmiast do innego, możemy również zbierać sygnały dla pojedynczego gniazda z wielu różnych klientów.
- nie dostajemy gwarancji, że datagram nie ulegnie duplikacji, dotrze do celu lub nie zostanie upuszczony gdzieś wewnątrz sieci.
- każdy datagram ma określony rozmiar, który jest przekazywany do adresata wraz z jego zwartością.

tcp (*transmission control protocol*):

- dane podróżują przez strumień. Utworzenie takiego strumienia wymaga nawiązania *połączenia* między klientem a serwerem. Jest to przypuszczalnie długotrwała relacja, która blokuje wykorzystywane gniazda po obu stronach na użytek tylko tego połączenia.
- protokół gwarantuje dostarczenie paczki lub informacje o porażce (mechanizm opiera się na wielokrotnym nadsyłaniu danych i oczekiwaniu na odpowiedź z serwera tak długo, aż nie zostanie przekroczony przybliżony czas oczekiwania, wtedy zwracany jest błąd – w ten sposób radzimy sobie chociażby z upuszczonymi segmentami albo martwym serwerem)
- przy transmisji tcp dzieli dane na segmenty, które ozdabia w metadane informujące o ich pozycji w pierwotnym ciągu (np. 1⌧1024 + 1025⌧2048) co pozwala odtworzyć ich zamierzoną kolejność.

Czym różni się komunikacja dupleksowa od półdupleksowej?

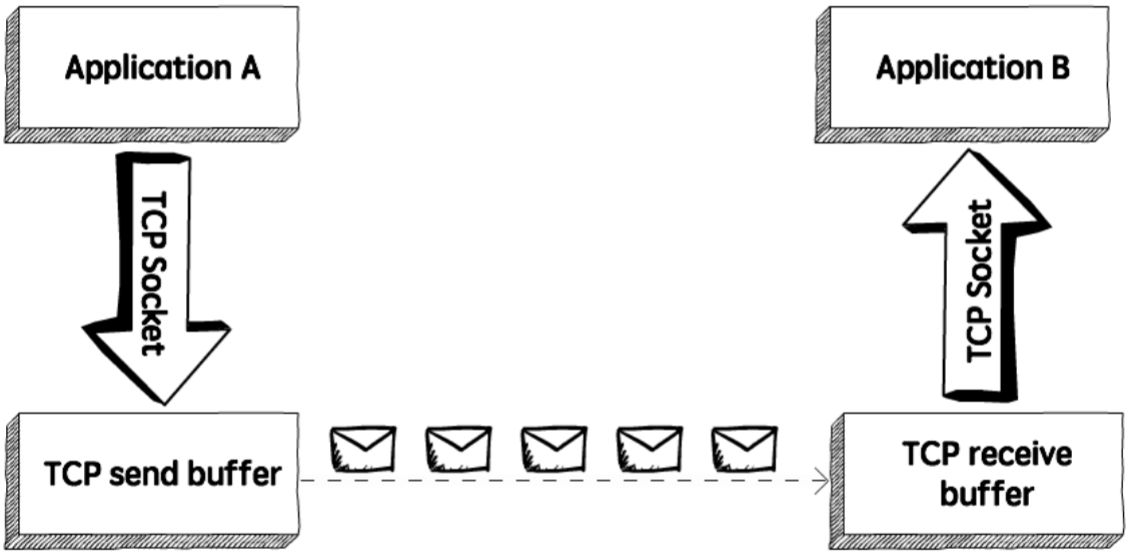
Komunikacja dupleksowa pozwala na równoległy przekaz w obie strony, w przeciwieństwie do półdupleksowej, która pozwala na obustronną komunikację, ale już nie na równoległą (tj. jeśli jeden koniec nadaje to drugi musi nasłuchiwać i vice versa).

Jaki problem rozwiązuje *sterowanie przepływem* imlementowane przez TCP?

The sender application writes data to a socket, the transport layer (in our case, `TCP`) will wrap this data in a segment and hand it to the network layer (e.g. `IP`), that will somehow route this packet to the receiving node.

On the other side of this communication, the network layer will deliver this piece of data to `TCP`, that will make it available to the receiver application as an exact copy of the data sent, meaning if will not deliver packets out of order, and will wait for a retransmission in case it notices a gap in the byte stream.

If we zoom in, we will see something like this.

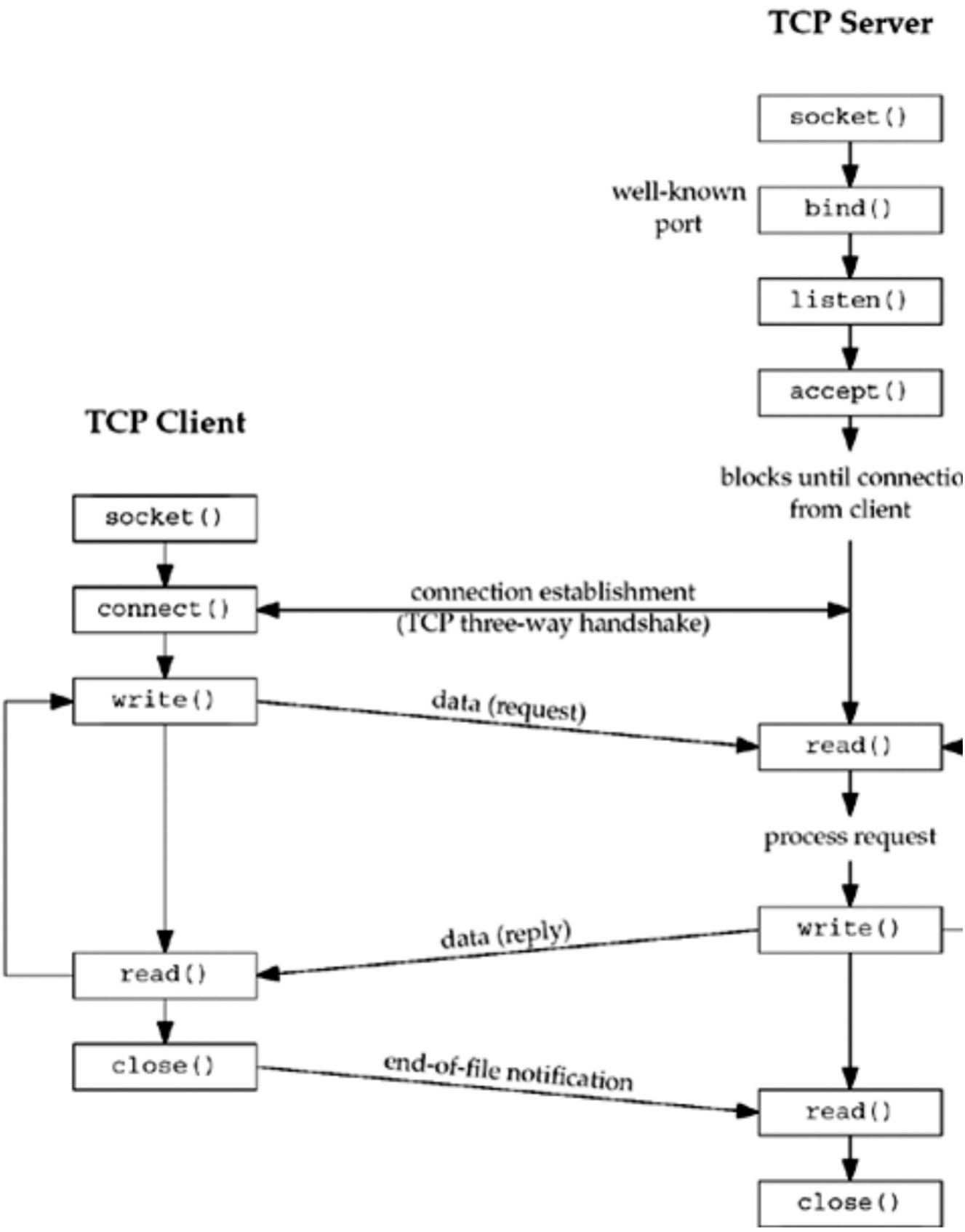


`TCP` stores the data it needs to send in the *send buffer*, and the data it receives in the *receive buffer*. When the application is ready, it will then read data from the receive buffer.

Flow Control is all about making sure we don't send more packets when the receive buffer is already full, as the receiver wouldn't be able to handle them and would need to drop these packets.

Zadanie 3

Zadanie 3. Omów diagram [7, 4.1] komunikacji **klient-serwer** używającej protoko
interfejsu **gniazd strumieniowych**. W którym momencie następuje związanie gnia
i zdalnym? Która ze stron komunikacji używa **portów efemerycznych** (ang. *ephemer*
argument wywołania systemowego `listen(2)`? Z jakim numerem portu jest związan
do i zwracane z `accept(2)`? Skąd serwer wie, że klient zakończył połączenie?



::: info

- **gniazdo strumieniowe** – dane przesyłane są segmentami, mamy również obsługę błędów i kończenia połączeń. Przeważnie wykorzystywane w protokole TCP
- **port efemeryczny** – port służący do wysyłania danych przez klienta (nie jest zarezerwowany dla żadnego protokołu) :::

well-known ports (aka, po prostu porty, które przyjmują dane poprzez odpowiadający im protokół)

Port	Protokół
53	DNS
20	FTP – przesyłanie oraz pobieranie plików i folderów
21	FTP – przesyłanie poleceń
67	DHCP – serwer
68	DHCP – klient
79	Finger
70	Gopher
80	HTTP, dodatkowe serwery, np. proxy, są najczęściej umieszczane na porcie 8080
443	HTTPS (HTTP na SSL)
143	IMAP
220	IMAP3
6661 – 6668	IRC
5222	XMPP – dla serwera sieci Jabber
389	LDAP
636	LDAPS (LDAP na SSL)
3306	MySQL
119	NNTP
110	POP3
995	POP3S (POP3 na SSL)
5432	PostgreSQL
873	Rsync
25	SMTP
22	SSH
514	Syslog
23	Telnet
69	TFTP
6000 – 6007	X11
161	SNMP
3389	RDP (Remote Desktop Connection)

socket - tworzy gniazdo do komunikacji sieciowej **bind** - wiąże gniazdo z lokalnym adresem IP (który hipotetycznie jest sposobem w jaki wiele procesów może przyjmować dane przez port wspólnego protokołu) **listen(sockfd, backlog)** - rozpoczyna nasłuchiwanie i kolejkowanie nadchodzących połączeń poprzez gniazdo *sockfd*, *backlog* definuje długość kolejki **accept(sockfd, ...)** - blokuje aż w *sockfd* pojawi się oczekujące połączenie, wtedy zostaje ono odebrane i zwrócony zostaje deskryptor nowego gniazda dla tego połączenia. **connect**(sockfd, *addr, addrlen*)* - nawiązuje połączenie z serwerem o adresie *addr* poprzez gniazdo *sockfd*. W tym celu jądro wybiera port efemeryczny dla tego połączenia. **close** - zamyka gniazdo kończąc połączenie, wysyła EOF do serwera.

Z jakim numerem portu jest związane gniazdo przyjmowane i zwracane przez accept()?

Zwracane gniazdo ma numer portu podniesiony z kolejki połączeń. Przyjmowane gniazdo ma port wybrany przy wywołaniu `socket`.

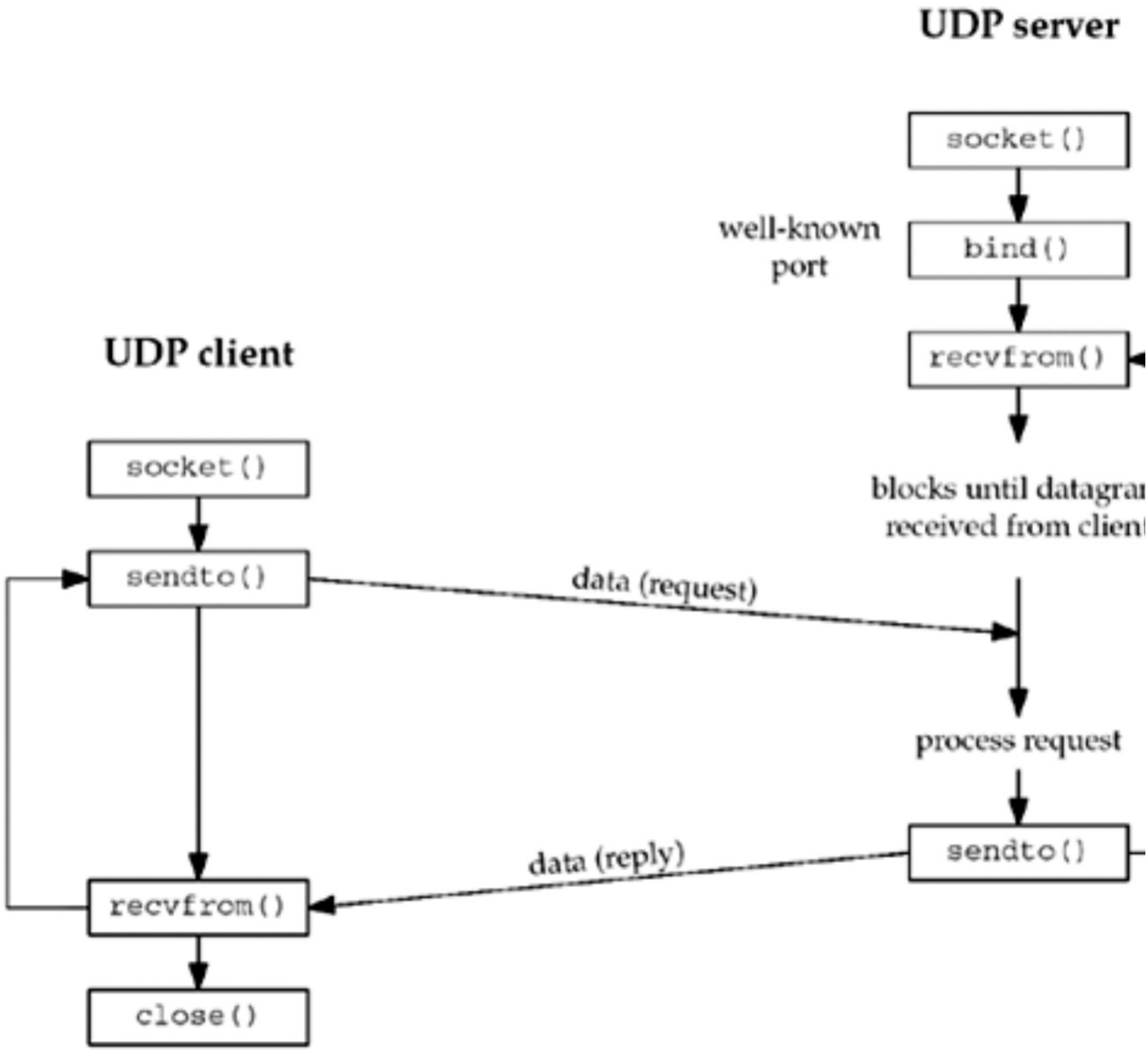
Skąd wiemy kiedy połączenie zostało zerwane?

Sygnał `FIN`

Zadanie 4

Zadanie 4. Omów diagram [7, 8.1] komunikacji klient-serwer używającej protokoł
interfejsu **gniazd datagramowych**. Czemu, w przeciwieństwie do TCP, serwer może
po wykonaniu funkcji `bind(2)`? Z jakiej przyczyny interfejs `read(2)` i `write(2)` p
być niewystarczający? Przedstaw semantykę operacji `recvfrom(2)` i `sendto(2)`. K
następuje związanie gniazda UDP z adresem lokalnym? Na podstawie [7, 8.11] zref
wykonanie `connect(2)` na gnieździe klienta. Jakie ograniczenia poprzednio wymie
poprawione przez wywołania `recvmsg(2)` i `sendmsg(2)`?

Figure 8.1. Socket functions for UDP client/server.



Gniazdo datagramowe - gniazdo bezpołączeniowe. Każdy pakiet jest indywidualnie kierowany do danego adresu. Nie ma gwarancji przesłania, ani zachowania kolejności.

Czemu w przeciwieństwie do TCP serwer może rozpocząć pracę zaraz po wykonaniu *bind()*?

Serwer nie musi czekać na nadejście połączenia.

Czemu read/write są niewystarczające?

Hipotetycznie dlatego, że read/write wymagają połączenia, tj. zakładamy, że zapis i odczyt odbywają się strumieniowo. Dla UDP dane przesyłane są w datagramach, dla których musimy z góry określić rozmiar przesyłu i adresata. Z tego powodu używamy `sendto` i `recvfrom`.

```
// przyjmujemy dane przez adres związany przez src_addr do bufora buf przez socket sockfd
ssize_t recvfrom(int sockfd, void *buf, size_t len, int flags,
                 struct sockaddr *src_addr, socklen_t *addrlen);

// wysyłamy dane przez sockfd na adres dest_addr
ssize_t sendto(int sockfd, const void *buf, size_t len, int flags,
               const struct sockaddr *dest_addr, socklen_t addrlen);
```

Zreferuj!

Już referuję... No więc dzieje się tak, że połączenie posiada już informacje o adresacie i adresie lokalnym zatem nie ma potrzeby jego precyzowania (możemy używać *send* i *recv*, które ich nie wymagają) i to jest tak pasjonujące, że aż warto mieć dedykowane temu pytanie.

Co jest fajnego w msg?

https://stackoverflow.com/questions/15874145/how-to-get-flags-returned-by-recvmsg-with-recvfrom Chyba chodzi o to, że działają niezależnie od typu gniazda, ale według tradycji tego przedmiotu nic nie jest pewne :)))

Zadanie 5

Zadanie 5. Przyjrzyjmy się warunkom brzegowym, które występują w trakcie używan
Kiedy `read(2)` i `write(2)` na gniazdach strumieniowych zwracają short counts? S
datagram UDP nie został obcięty przez jądro w trakcie kopiowania do przestrzeni
przyczyn należy być przygotowanym na to, że operacje na gniazdach zwrócą «EIN
klient spróbuje zapisać do gniazda powiązanego z połączeniem, które serwer zdążył
w kodzie funkcji «open_listenfd» użyto wywołania `setsockopt(2)` z opcją «SO_
stało gdyby programista o tym zapomniał?

Kiedy read i write zwracają short count?

Jeśli piszemy do gniazda, którego bufor jest już pełny lub jeśli bufor mieści tylko część danych, które chcielibyśmy zapisać. Short count może mieć również miejsce, gdy przekroczony zostaje czas oczekiwania na gniazdo zewnętrzne.

Skąd wiemy, że odebrany datagram nie został obcięty?

Datagramy zawierają informację o swoim rozmiarze w nagłówku, wystarczy zatem porównać ją z rozmiarem ładunku. Swoją drogą to pytanie jest całkowicie syntetyczne xD

Z jakich przyczyn należy być przygotowanym na to, że operacje zwrócą EINTR?

```
On  success,  these  system calls return a file descriptor
for the accepted socket (a nonnegative integer).
On error, -1 is returned, errno is set apropriately
```

Przyczyny Jeśli w trakcie wywołania systemowego jednej z takich procedur przyjdzie sygnał to zostaje ona przerwana i zwraca błąd.

EINTR The system call was interrupted by a signal that was caught before a valid connection arrived; see **signal**(7).

Co się stanie jak klient spróbuje pisać gniazda z połączeniem, które serwer już zamknął?

Klient dostanie `SIGPIPE`

Dlaczego w kodzie funkcji *open_listenfd* użyto wywołania *setsockopt* z opcją, której nie przepiszę?

```
/*
 * open_listenfd - open and return a listening socket on port
 *     Returns -1 and sets errno on Unix error.
 */

typedef struct sockaddr SA;

int open_listenfd(int port)
{
  int listenfd, optval=1;
  struct sockaddr_in serveraddr;

  /* Create a socket descriptor */
  if ((listenfd = socket(AF_INET, SOCK_STREAM, 0)) < 0)
    return -1;

  /* Eliminates "Address already in use" error from bind. */
  if (setsockopt(listenfd, SOL_SOCKET, SO_REUSEADDR,
                 (const void *)&optval , sizeof(int)) < 0)
    return -1;

  /* Listenfd will be an endpoint for all requests to port
     on any IP address for this host */
  bzero((char *) &serveraddr, sizeof(serveraddr));
  serveraddr.sin_family = AF_INET;
  serveraddr.sin_addr.s_addr = htonl(INADDR_ANY);
  serveraddr.sin_port = htons((unsigned short) port);

  if (bind(listenfd, (SA *) &serveraddr, sizeof(serveraddr)) < 0)
    return -1;

  /* Make it a listening socket ready to accept connection requests */
  if (listen(listenfd, LISTENQ) < 0)
    return -1;

  return listenfd;
}
```

::: danger Można łatwo zgadnąć, że chodzi tu o wtórne użycie lokalnego adresu jakiegoś gniazda, w normalnej sytuacji przy takiej próbie dostalibyśmy w twarz błędem (co ma sporo sensu) możemy jednak ten błąd zignorować (co nie ma już zbyt wiele sensu) używając `setsockopt` z SO_REUSEADDR. (kandydat na ***free question coupon***) :::

Zadanie 6

Ściągnij ze strony przedmiotu archiwum «so21_lista_9.tar.gz», następnie rozpakuj i zapoznaj si

UWAGA! Można modyfikować tylko te fragmenty programów, które zostały oznaczone w koment

Zadanie 6. Zmodyfikuj program «hostinfo.c» w taki sposób, aby wyświetlał dla danej nazwy serwera. Dodatkowo należy przekształcić nazwę usługi przekazane parametr programu na numer portu. Poniżej przykład:

```
# hostinfo www.google.com https
216.58.215.68:443
[2a00:1450:401b:803::2004]:443
```

Co należałoby zrobić, żeby program rozpoznawał usługę o nazwie «tftp»?

getaddrinfo(char *node, char *service, addrinfo *hints, addrinfo **res)

- node -- nazwa hosta (np. "www.google.com")
- service -- nazwa usługi (np. "https")
- hints -- wskaźnik na addinfo, której pola są wykorzystane jako kryteria wyszukiwania
- res -- tutaj zwracamy wartość, jako wskaźnik na listę struktur addrinfo, gdzie każda zawiera już przetłumaczony adres

getnameinfo(sockaddr *addr, socklen_t addrlen, char *host, socklen_t hostlen, char *serv, socklen_t servlen, int flags) -- funkcja odwrotna do `getaddrinfo` przyjmuje lokalny adres IP i numer portu (przez strukturę sockaddr). Jej wyniki czyli nazwa hosta i nazwa usługi są zwracane odpowiednio przez *host* i *serv*. Możemy również zmodyfikować jej zachowanie ustawiając odpowiednie flagi np. NI_NUMERICHOST i NI_NUMERICSERV drukują reprezentację "kropkowatą".

TFTP (Trivial File Transfer Protocol) -- a simple lockstep File Transfer Protocol which allows a client to get a file from or put a file onto a remote host. One of its primary uses is in the early stages of nodes booting from a local area network. TFTP has been used for this application because it is very simple to implement.

```
struct addrinfo {
    int ai_flags;
    int ai_family;
    int ai_socktype;
    int ai_protocol;
    socklen_t ai_addrlen;
    struct sockaddr *ai_addr;
    char *ai_canonname;
    struct addrinfo *ai_next;
};
```

```
int main(int argc, char **argv) {
  struct addrinfo *p, *listp, hints;
  char buf[MAXLINE];
  char puf[MAXLINE];
  int rc, flags;

  if (argc != 2 && argc != 3)
    app_error("usage: %s <domain name>\n", argv[0]);

  /* Get a list of addrinfo records */
  memset(&hints, 0, sizeof(struct addrinfo));

  //- hints.ai_family = AF_INET; /* IPv4 only */
  hints.ai_family = AF_UNSPEC;   /* IPv4 + IPv6 */

  /* SOCK_DGRAM jeśli marzy nam się obsługiwać tylko UDP
  lub możemy pozostawić to pole niewypełnione co oznacza obsługę dowolnego typu gniazda */
  hints.ai_socktype = SOCK_STREAM;

  /* Connections only */
  //- if ((rc = getaddrinfo(argv[1], NULL, &hints, &listp)) != 0)
  char *service = argc == 3 ? argv[2] : NULL;
  if ((rc = getaddrinfo(argv[1], service, &hints, &listp)) != 0)
    gai_error(rc, "getaddrinfo");

  /* Walk the list and display each IP address */
  flags = NI_NUMERICHOST;  /* Display address string instead of domain name */
  flags |= NI_NUMERICSERV; /* Display service string instead of service name */
  for (p = listp; p; p = p->ai_next) {
    Getnameinfo(p->ai_addr, p->ai_addrlen, buf, MAXLINE, puf, MAXLINE, flags);

    if (p->ai_family == AF_INET)
      printf("%s", buf);
    else
      printf("[%s]", buf);

    if (service != NULL)
      printf(":%s", puf);

    printf("\n");

  }

  /* Clean up */
  freeaddrinfo(listp);

  return EXIT_SUCCESS;
}
```

Zadanie 7

Zadanie 7. Zapoznaj się z kodem źródłowym serwera «echoserver.c» i klienta «
podobnej do «echo». Twoim zadaniem jest taka modyfikacja serwera, by po odebr
wydrukował liczbę bajtów odebranych od wszystkich klientów, po czym zakończył s

Używając programu «watch» uruchom polecenie «netstat -ptn», aby obserwować s
Wystartuj po jednym procesie serwera i klienta używając wybranego portu (np. 777
końce połączenia należące do serwera i klienta. Następnie wystartuj drugą instan
zachowuje się ona tak samo jak pierwsza? Co zmieniło się na wydruku z «netstat»

Uruchom program wireshark. Na interfejsie sieciowym loopback każ mu nasł
przychodzących na port, który wybrano do komunikacji między klientem i serwerem. W
w trakcie otwierania i zamykania połączenia oraz w trakcie przesyłania danych międ

Zadanie 8

Zadanie 8. Serwer z poprzedniego zadania nie radził sobie ze współbieżną obsługą
z pliku «echoclient-fork.c» naprawia to poważne ograniczenie z użyciem wywoł
głównego procesu jest odbieranie połączeń i delegowanie ich obsługi do podprocesó

Proces serwera musi zliczać liczbę bajtów odebranych od klientów. W tym celu prz
anonimową, w której przechowuje tablicę «client». Przy starcie podprocesu umieszcz
wpis za pomocą procedury «addclient». Żeby uniknąć wyścigów każdy podproce
nik «nread». Po zakończeniu podprocesu należy wywołać procedurę «delclient»
prywatnego licznika klienta, do globalnego licznika serwera.

W dowolnym momencie działanie serwera może zostać przerwane przy pomocy syg
wtedy poczekać na zakończenie podprocesów i wydrukować zawartość globalnego li
przykładowy wydruk z sesji serwera:

```
# ./echoserver-fork 8000
[9047] Connected to localhost:36846
[9105] Connected to localhost:36850
[9047] Disconnected!
^C
Server received quit request!
[9105] Disconnected!
Server received 22 bytes
#
```

Zadanie 9

Zadanie 9 (bonus). Zapoznaj się z procedurami «echo» w pliku «echoclien
i «echoserver-fork.c». Usuń komentarz otaczający wywołanie funkcji exit(3
cie i serwerze. Zaobserwuj występujące usterki, a następnie wytłumacz ich źródło. Ja
by nie ulegały awarii?