Systemy Operacyjne

Gniazda i komunikacja sieciowa

Dr hab. inż. Krzysztof Rzecki, prof. AGH

Komunikacja między procesami

- System komputerowy: IPC (ang. *Interprocess Communication*):
  - Standardy: POSIX, System V
  - Rodzaje: pipe, kolejki, pamięć wspólna, semafory, rygle, etc.
- Sieci komputerowe:
  - Warstwa łącza: **Ethernet**, SLIP, PPP
  - Protokoły sieciowe: **IPv4**, IPv6, ICMP
  - Protokoły transportowe: **TCP**, **UDP**
  - Protokoły aplikacji: DNS, HTTP, SSH, etc.

Programowanie usług sieciowych

… czyli aplikacji:

- wieloprocesowych,
- komunikujących się przez interfejs sieciowy,
- tworzących formę (architekturę) klient-serwer.

Zapotrzebowanie

Komputer użytkownika **klient**

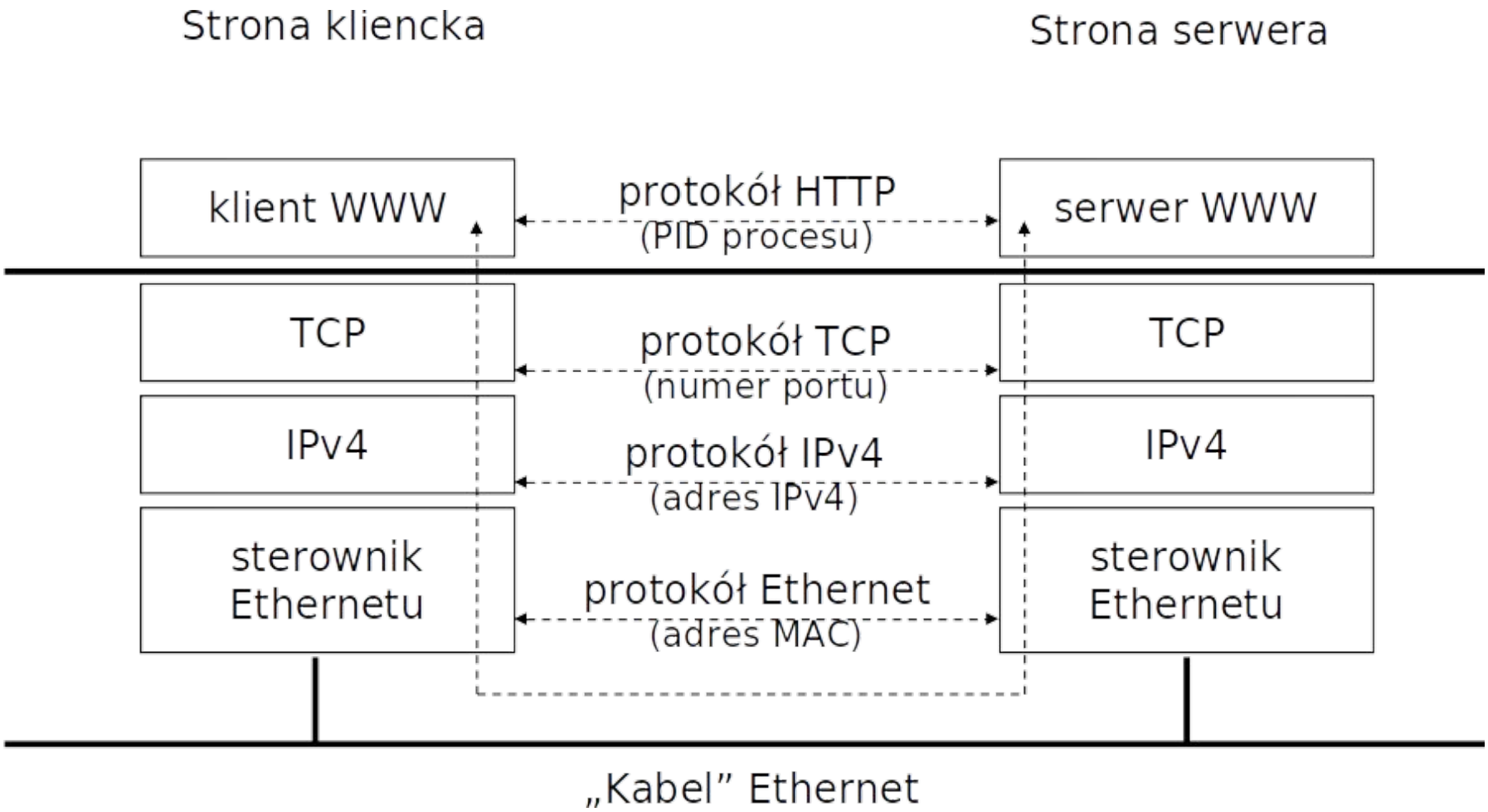
Komputer z zasobami **serwer**

Zasadniczo słabo wyposażony komputer z własnym systemem operacyjnym i oprogramowaniem komunikacyjnym.

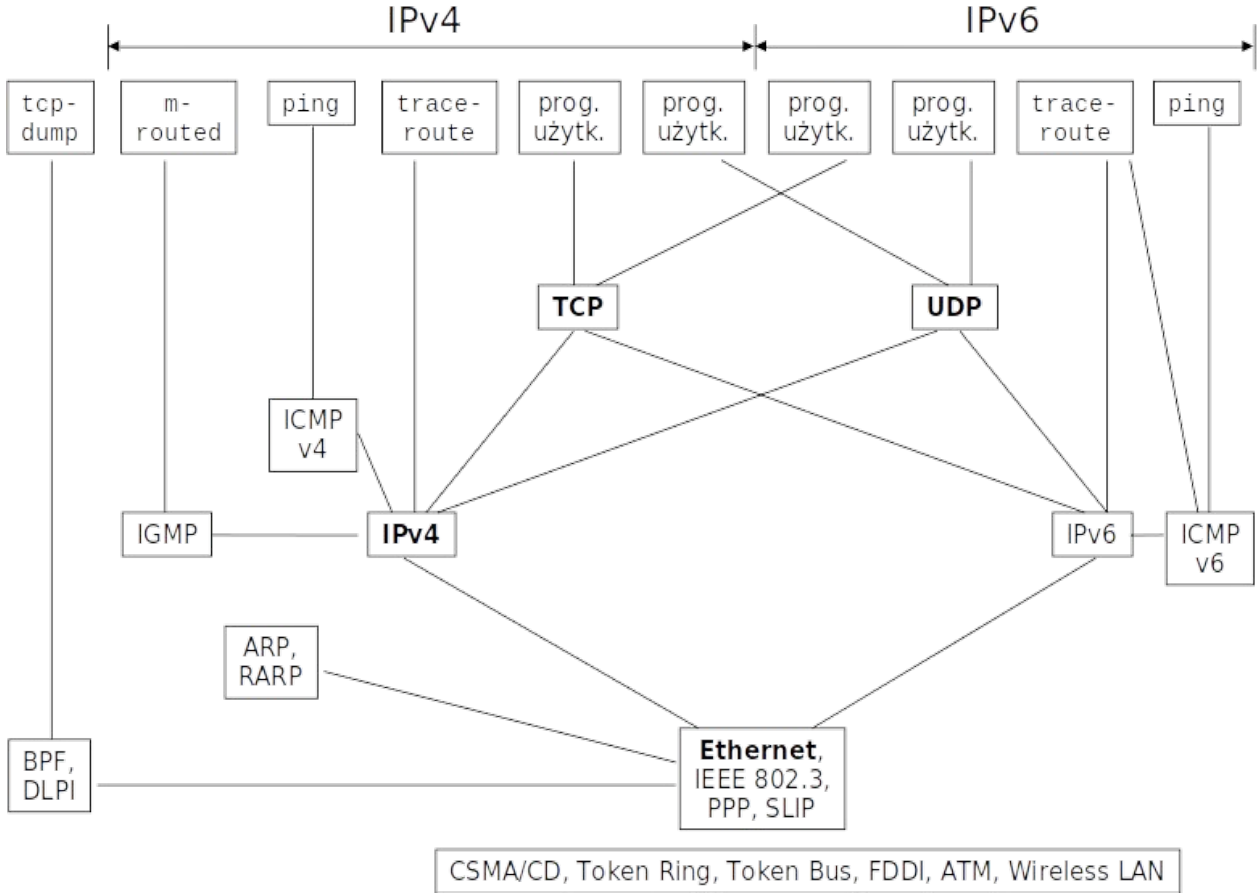
Pojęcie: cienki klient (ang. thin client)

Zasoby (ang. resources):

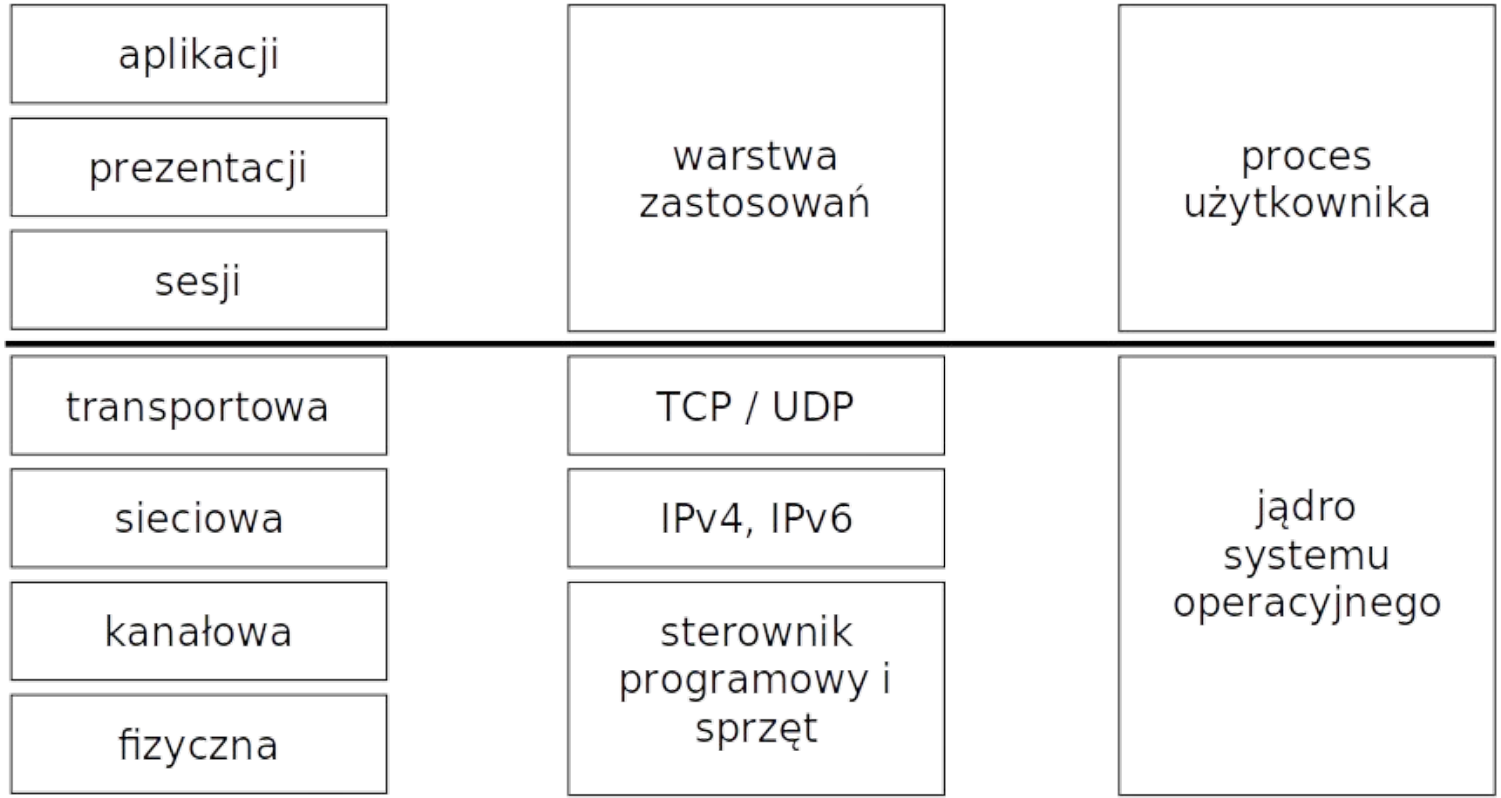
- przestrzeń dyskowa:
  - pobieranie danych,
  - przechowywanie danych,
- czas procesora,
- pamięć operacyjna,
- dostęp do innych urządzeń,
- łącze / brama (ang. gateway),


Strona kliencka
Strona serwera
klient WWW
protokół HTTP
(PID procesu)
serwer WWW
TCP
protokół TCP
(numer portu)
TCP
IPv4
protokół IPv4
(adres IPv4)
IPv4
sterownik Ethernetu
protokół Ethernet
(adres MAC)
sterownik Ethernetu
„Kabel” Ethernet

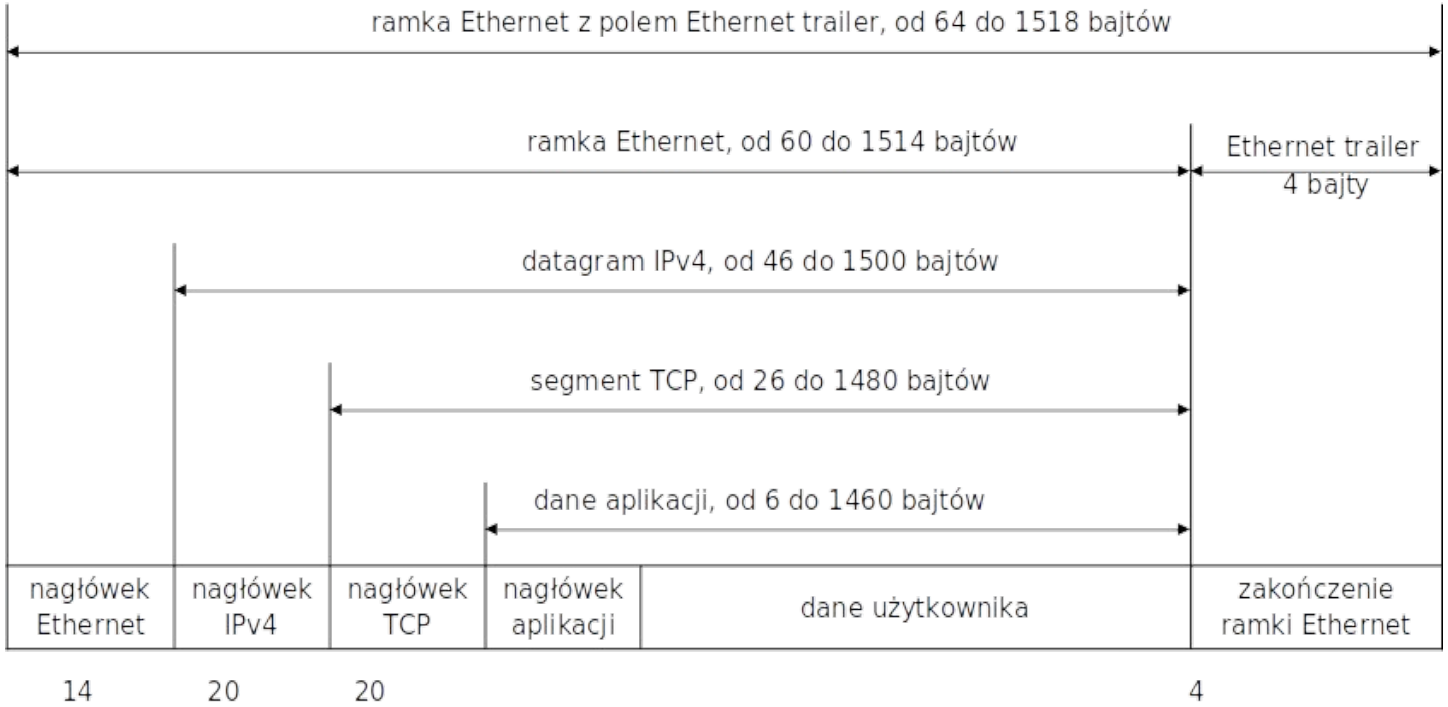
Przegląd protokołów



Model i rodzina



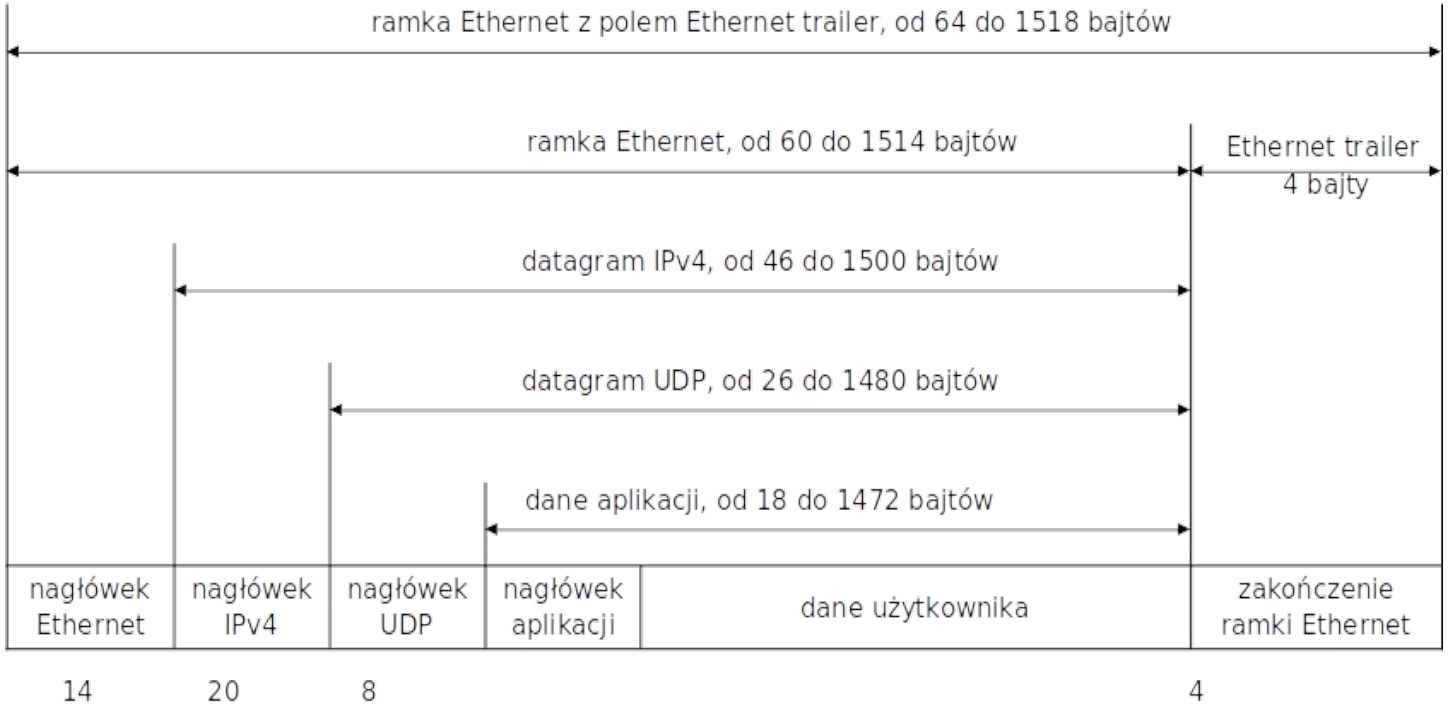
Enkapsulacja TCP/IPv4/Ethernet



Wprowadzone ograniczenia wywodzą się z MTU

TCP strumieniuje – nie ma ograniczeń

Enkapsulacja UDP/IPv4/Ethernet



Wprowadzone ograniczenia wywodzą się z MTU

UDP ogranicza host i pole określające wielkość

Internet Protocol i adres IPv4

Port

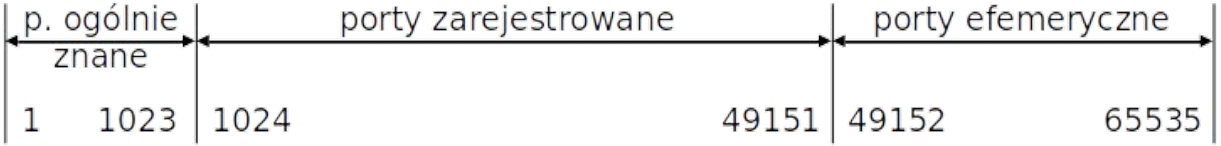
Port to 16-bitowa liczba całkowita z zakresu 1…65535.

Trzy grupy (IANA):

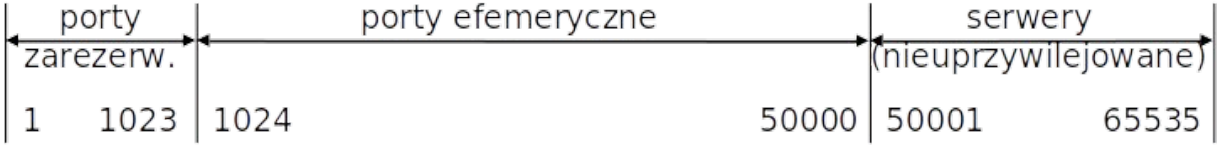
- strona serwera: porty ogólnie znane (ang. well-known port),
- strona serwera: porty zarezerwowane (ang. reserved port),
- strona klienta: porty efemeryczne (ang. ephemeral port).

Port

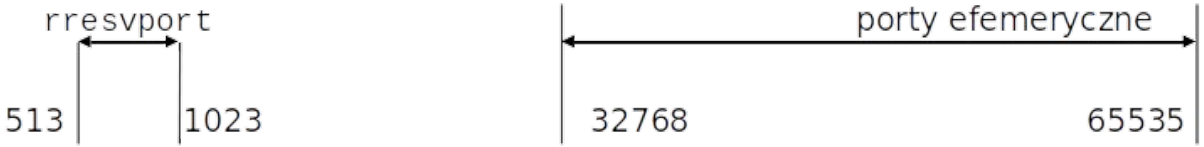
- Konwencje **przyjęte** przez IANA
  (ang. *Internet Assigned Numbers Authority*)



- Konwencje BSD



- Konwencje Solaris



Protokół użytkowy

- Protokół użytkowy to format komunikacji między procesami umieszczony najwyżej w hierarchii rodziny protokołów Internetu.
- Protokół użytkowy obejmuje warstwę sesji, prezentacji oraz zastosowań.
- Przykłady:
  - HTTP (ang. Hyper Text Transfer Protocol),
  - FTP (ang. File Transfer Protocol),
  - SSH (ang. Secure SHell).
- Skojarzenie port <-> protokół użytkowy nie musi być jednoznaczne!

Podsumowanie wprowadzenia

- Protokół Ethernet i adres MAC -> bezpośrednie przekazywanie danych.
- Protokół IP oraz adres IPv4 (IPv6) -> przekazywanie danych poza sieć lokalną (ang. routing).
- Protokół TCP/UDP -> kontrola przesyłanych danych między aplikacjami.
- Rola systemu operacyjnego to przekazywanie:
  - danych od aplikacji do wysłania,
  - dane otrzymanych dla aplikacji.
- System operacyjny utrzymuje skojarzenie: numer portu i PID procesu
- Protokół użytkowy: format komunikacji między aplikacjami.

Standardy systemu Unix

- IEEE (ang. Institute for Electrical and Electronics Engineers, Inc.), której standardy są normami ISO/IEC:
  - Posix – Przenośny Interfejs Systemu Operacyjnego (ang. Portable Operating System Interface) => Posix.1g,
  - ISO (ang. International Organization for Standarization),
  - IEC (ang. International Electrotechnical Commission),
- Open Group:
  - powstała z X/Open Company oraz OSF (ang. Open Software Foundation),
  - X/Open Portability Guide,
  - Single Unix Specification,
  - CDE – Common Desktop Environment,
- IETF (ang. Internet Engineering Task Force):
  - dokumenty RFC,
  - dokumenty Internet-Drafts.

Przykłady standardów RFC

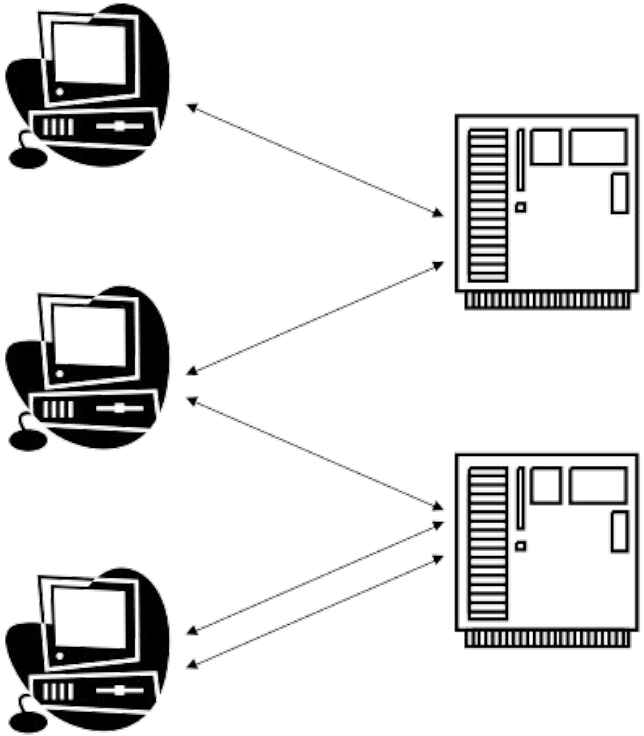
- Adres e-mail i format wiadomości:
  - RFC 822 Standard for the Format of ARPA Internet Text Messages, 1982r.
  - Rozszerzenia i aktualizacje: RFC 2822, RFC 4021, RFC 5280, RFC 5322, RFC 6854, etc.
- DNS:
  - RFC 882: DOMAIN NAMES - CONCEPTS and FACILITIES, 1983r.
  - RFC 883: DOMAIN NAMES - IMPLEMENTATION and SPECIFICATION, 1983r.
  - RFC 2929: Domain Name System (DNS) IANA Considerations, 2000r.
- IRC: RFC 1459: Internet Relay Chat Protocol, 1993r.
- Sieci lokalne: RFC 1918: Address Allocation for Private Internets
- VoIP: RFC 2543: SIP: Session Initiation Protocol
- HTTP: RFC 7231: Hypertext Transfer Protocol (HTTP/1.1): Semantics and Content
- Video: RFC 8216: HTTP Live Streaming

Przykłady standardów RFC

Badania:

- RFC 5106: The Extensible Authentication Protocol-Internet Key Exchange Protocol version 2 (EAP-IKEv2) Method
- RFC 5222: LoST: A Location-to-Service Translation Protocol

Klient – Serwer



Jedna lub wiele **aplikacji klienckich**

na jednej lub wielu **maszynach klienckich**

Jedna lub wiele **aplikacji serwerowych**

na jednej lub wielu **maszynach serwerowych**

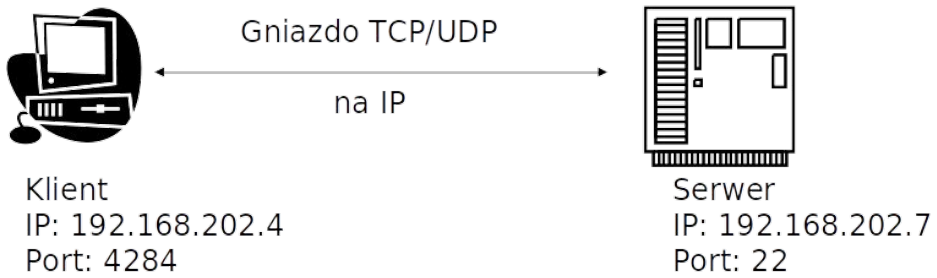
Klient
aplikacja, która pobiera zasoby.

Jak to możliwe ?

Serwer
aplikacja, która udostępnia zasoby.

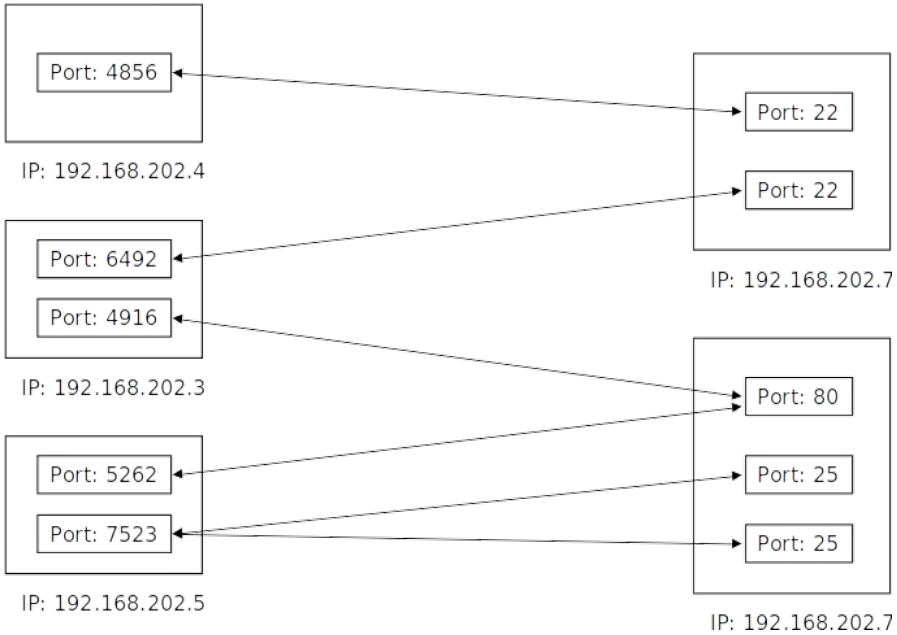
Gniazdo (ang. *socket*)

- **Gniazdo** to unikalna struktura dwóch par liczb (czyli czterech liczb):
  - adres IP i port klienta,
  - adres IP i port serwera.

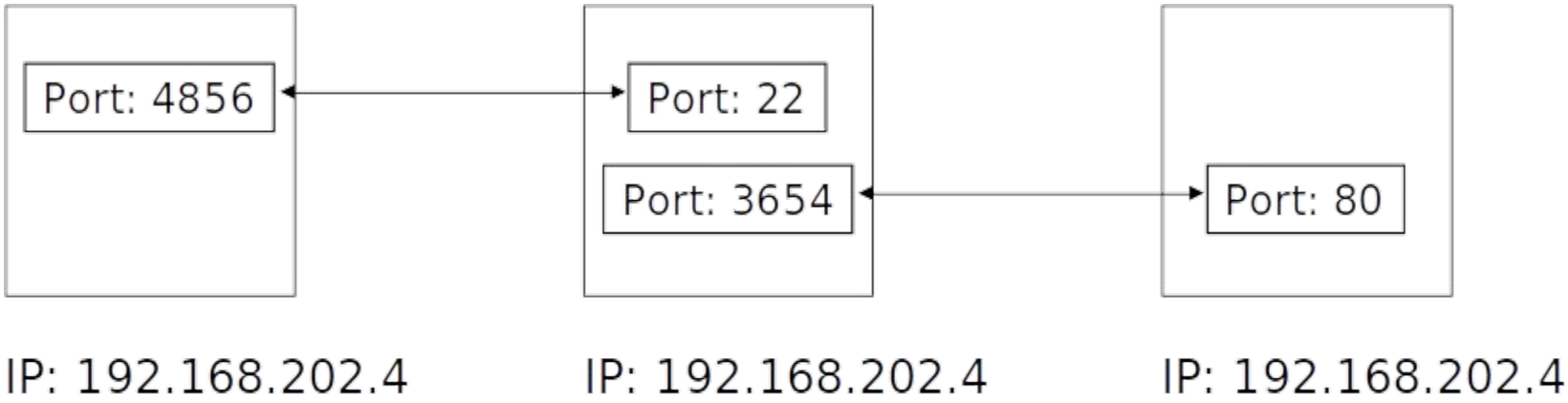


Klient / Serwer to aplikacje czy komputery ?

Architektura klient-serwer



Serwer to czy klient ?



?

Ogólna gniazdowa struktura adresowa

```
<sys/socket.h>                          // plik nagłówkowy
```

```
struct sockaddr {                       // uogólniona gniazdowa SA
 uint8_t        sin_len;                // długość struktury, 16 bajtów (?)
 sa_family_t    sin_family;             // AF_XXX
 char           sa_data[14];            // adres właściwy dla protokołu
};
```

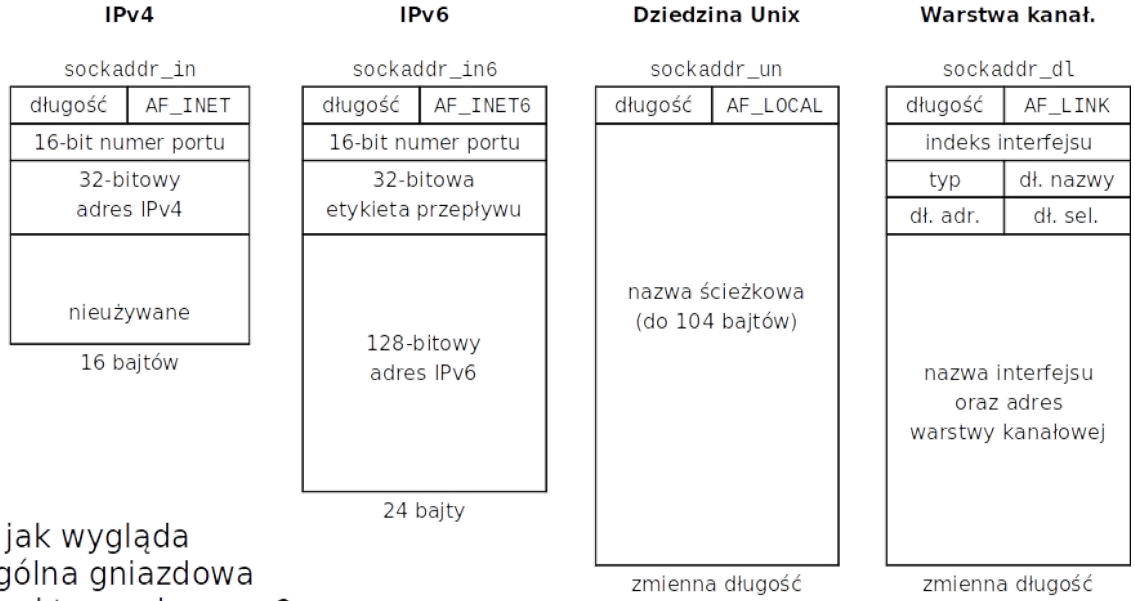
Gniazdowa SA dla IPv4

```
<netinet/in.h>                  // plik nagłówkowy

struct in_addr {                // struktura adresu IPv4
 in_addr_t s_addr;              // 32-bitowy adres IPv4 w sieciowej
};                              // kolejności bajtów

struct sockaddr_in {            // gniazdowa struktura adresowa IPv4
 uint8_t        sin_len;        // długość struktury, 16 bajtów
 sa_family_t    sin_family;     // AF_INET
 in_port_t      sin_port;       // 16 bitowy numer portu TCP/UDP
 struct in_addr sin_addr;       // w sieciowej kolejności bajtów
 char           sin_zero[0];    // 32-bitowy adres IPv4 (j.w.)
};                              // nieużywany
```

Gniazdowe struktury adresowe



A jak wygląda ogólna gniazdowa struktura adresowa?

Operacje na SA

- Przesyłane parametry to co najmniej:
  - deskryptor gniazda (`int`),
  - dowolna gniazdowa struktura adresowa zrzutowana na ogólną gniazdową strukturę adresową (`struct sockaddr *`),
  - długość struktury (`int` lub `int *`).

- Sposób przekazywania parametrów z procesu do jądra:
  `int function ( int, struct sockaddr *, int );`
  Przykład: `bind, connect, sendto.`

- Sposób pobierania parametrów z jądra do procesu(*):
  `int function ( int, struct sockaddr *, int * );`
  Przykład: `accept, recvfrom, getsockname, getpeername`

(*) często dana funkcja przesyła parametry w obie strony

Kolejność bajtów

Jeśli zapiszemy daną liczbę w jednym modelu, a odczytamy tak, jakby to był drugi z modeli, wówczas otrzymamy liczbę o przestawionej kolejności bajtów (nie bitów!):

3232287249 <-> 298494144

192.168.202.17 <-> 17.202.168.192

5432 <-> 14357

Kolejność bajtów

- Systemowa kolejność bajtów
  (ang. host byte order) to kolejność przechowywania bajtów w pamięci komputera:
  - Linux: little-endian,
  - HP-UX: big-endian.
- Sieciowa kolejność bajtów
  (ang. network byte order) to kolejność przechowywania bajtów w protokołach sieciowych – tylko big-endian.

Funkcje konwertujące

Oznaczenia:

‘n’ = network oraz ‘h’ = host

‘s’ = short oraz ‘l’ = long

Funkcje host -> network

```
uint16_t htons (uint16_t host16bitvalue);
uint32_t htonl (uint32_t host32bitvalue);
```

Funkcje network -> host

```
uint16_t ntohs (uint16_t net16bitvalue);
uint32_t ntohl (uint32_t net32bitvalue);
```

presentation <-> numeric

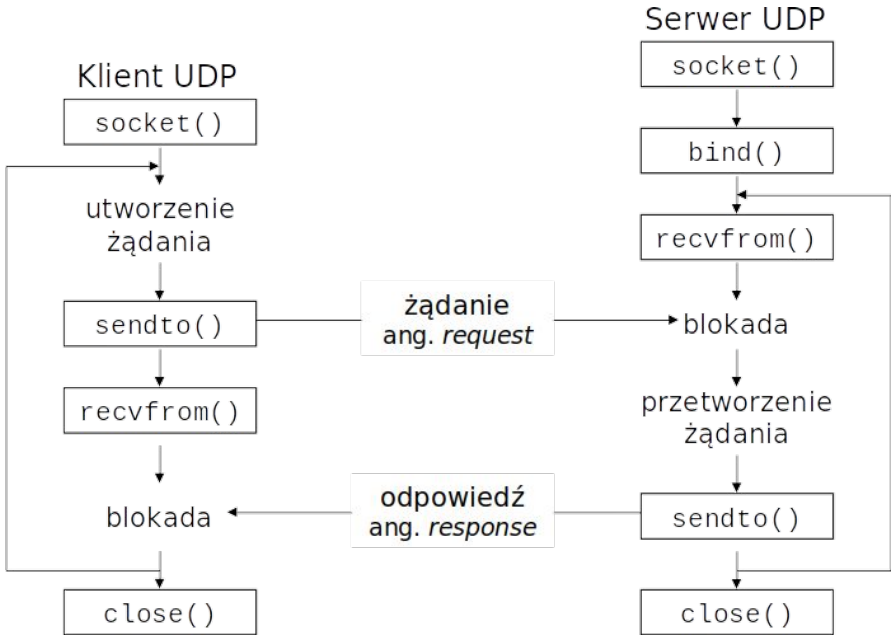
presentation (network) -> numeric (host)

```
int inet_aton (const char *strptr, struct in_addr *addrptr);
in_addr_t inet_addr (const char *strptr);
int inet_pton (int family, const char *strptr, void *addrptr);
```

numeric (host) -> presentation (network)

```
char *inet_ntoa (struct in_addr inaddr);
const char *inet_ntop (int family, const void *addrptr, char *strptr, size_t len);
```

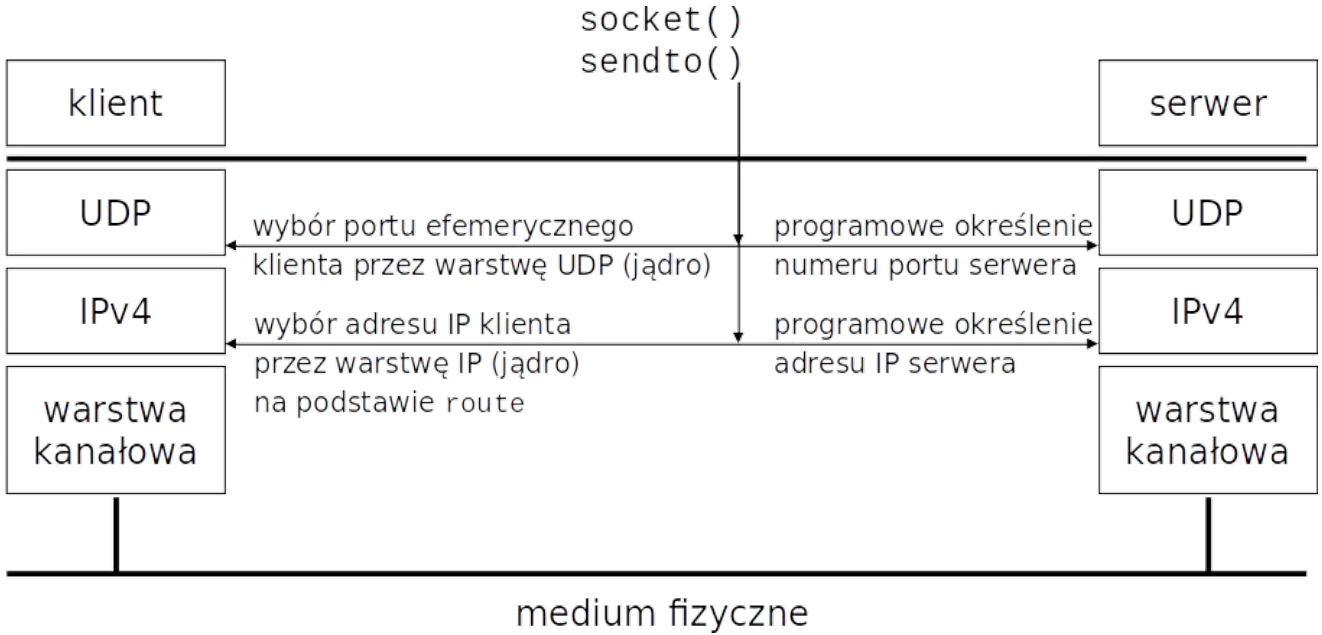
Algorytm iteracyjny dla UDP



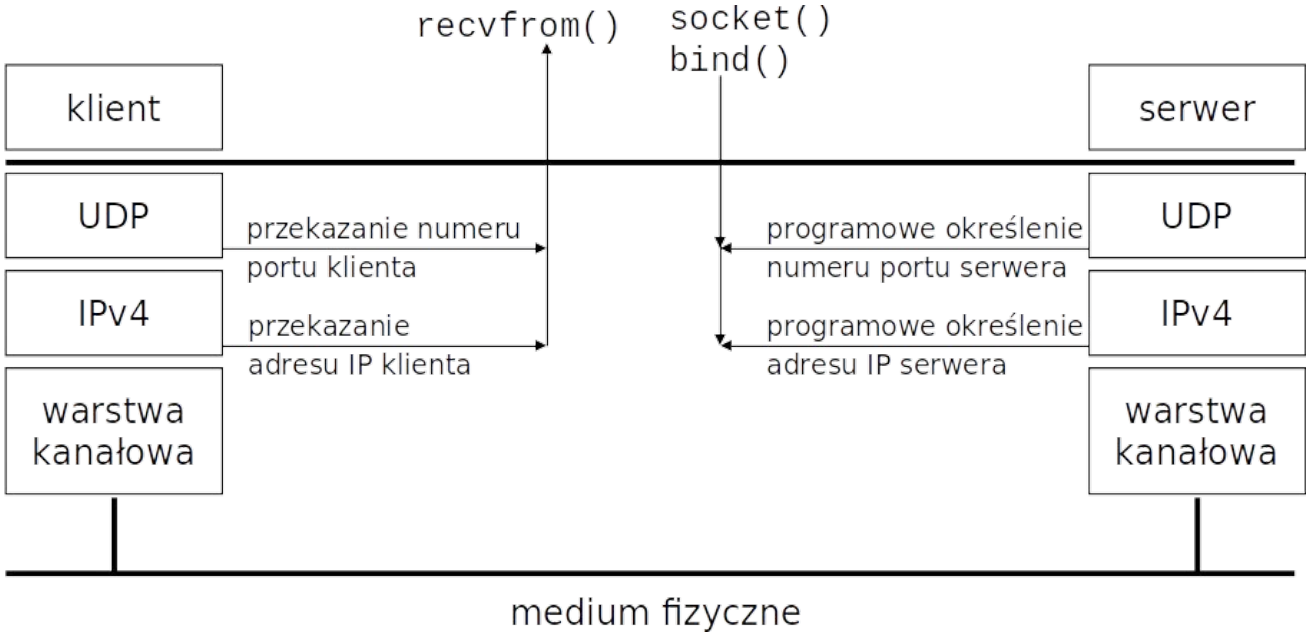
Algorytm iteracyjny dla UDP

- **Nie ma fazy akceptacji połączenia** – kto nadeśle datagram, ten będzie przyjęty (nie oznacza to, że każdy klient musi być obsłużony).
- Serwer pracuje w **nieskończonej pętli**, gdyż nie wiadomo kiedy klient zakończy swoje działanie (brak fazy zakończenia połączenia) => w jednej pętli może występować tylko jedna wymiana komunikatów, która musi zapewnić pełne obsłużenie klienta.
- Serwer działa w trybie **iteracyjnym** – serwer współbieżny, ze względu na wcześniejszy powód, mógłby nigdy się nie zakończyć => nieskończone mnożenie się procesów.
- W skończonym buforze obowiązuje kolejka **FIFO**.
- Funkcja **close()** odnosi się do socket'u, a nie do połączenia.

Analiza z punktu widzenia klienta



Analiza z punktu widzenia serwera



Implementacja: serwer UDP

```
struct sockaddr_in servaddr, cliaddr;       // struktury adresowe serwer/klient
memset (&servaddr, 0, sizeof(servaddr));// wyzerowanie str.adr. serwera
servaddr.sin_family     = AF_INET;
servaddr.sin_addr.s_addr          = htonl (INADDR_ANY);
servaddr.sin_port                 = htons (9100);

// utworzenie socket'u i przypisanie adresu/portu lokalnego:
int sockfd = socket (AF_INET, SOCK_DGRAM, 0);
int bindresult = bind (sockfd, (struct sockaddr *) &servaddr, sizeof (servaddr));

int msglen;
const size_t MAXLINE = 100;
char msg[MAXLINE];

while ( 1 )
{
    // odbierz datagram
    socklen_t cliaddrlen = sizeof(cliaddr);
    msglen = recvfrom (sockfd, msg, MAXLINE, 0,
                       (struct sockaddr *) &cliaddr, &cliaddrlen);
    printf ("msg: %s msglen: %d\n", msg, msglen);
    // wyślij datagram
    sendto (sockfd, msg, msglen, 0, (struct sockaddr *) &cliaddr, cliaddrlen);
}
```

Implementacja: klient UDP

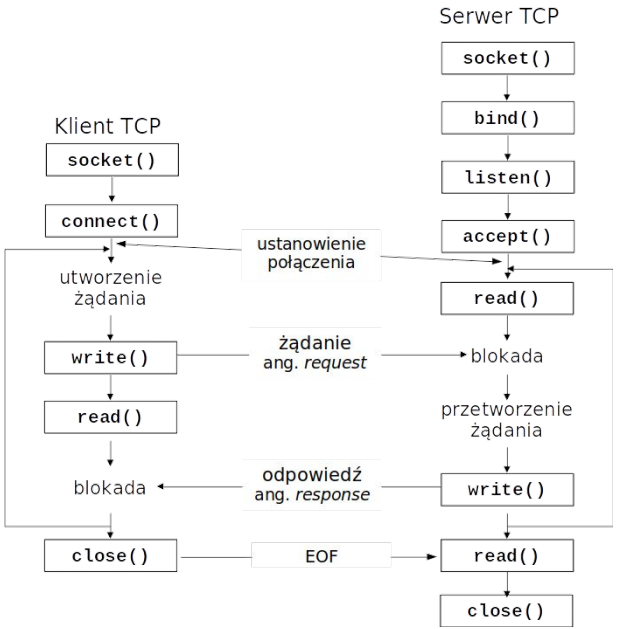
```
struct sockaddr_in servaddr;                // struktura adresowa serwera
memset (&servaddr, 0, sizeof(servaddr));    // wyzerowanie str.adr. serwera
servaddr.sin_family     = AF_INET;
inet_pton (AF_INET, "127.0.0.1", &servaddr.sin_addr);
servaddr.sin_port               = htons (9100);

// utworzenie socket'u
int sockfd = socket (AF_INET, SOCK_DGRAM, 0);

int msglen;
const size_t MAXLINE = 100;
char msg[MAXLINE];

while ( 1 )
{
    // wyślij datagram
    socklen_t servaddrlen = sizeof(servaddr);
    fgets (msg, MAXLINE-1, stdin);
    sendto (sockfd, msg, strlen(msg), 0,
                    (struct sockaddr *) &servaddr, servaddrlen);
    // odbierz datagram
    msglen = recvfrom (sockfd, msg, MAXLINE, 0, NULL, NULL);
    printf ("msg: %s msglen: %d\n", msg, msglen);
}
```

Algorytm iteracyjny dla TCP



Uzgodnienie trójfazowe (ang. *three-way handshake*)

Otwarcie bierne (ang. passive open) – przygotowanie serwera na przyjęcie połączenia:

- serwer C/C++: `socket(), bind(), listen(), accept()` (blokada)

Otwarcie aktywne (ang. active open) – klient nawiązuje połączenie wysyłając pakiet synchronizujący (ang.synchronize):

- klient C/C++: `connect()` (blokada)
- tcpdump: SYN

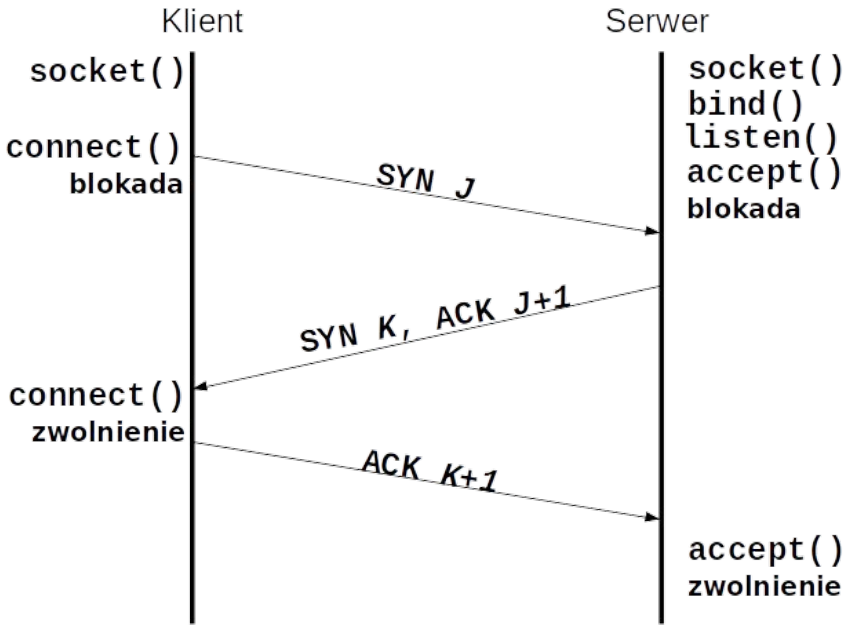
Potwierdzenie (ang. acknowledgment) ze strony serwera:

- tcpdump: SYN, ACK
- klient C/C++: `connect()` (zwolnienie)

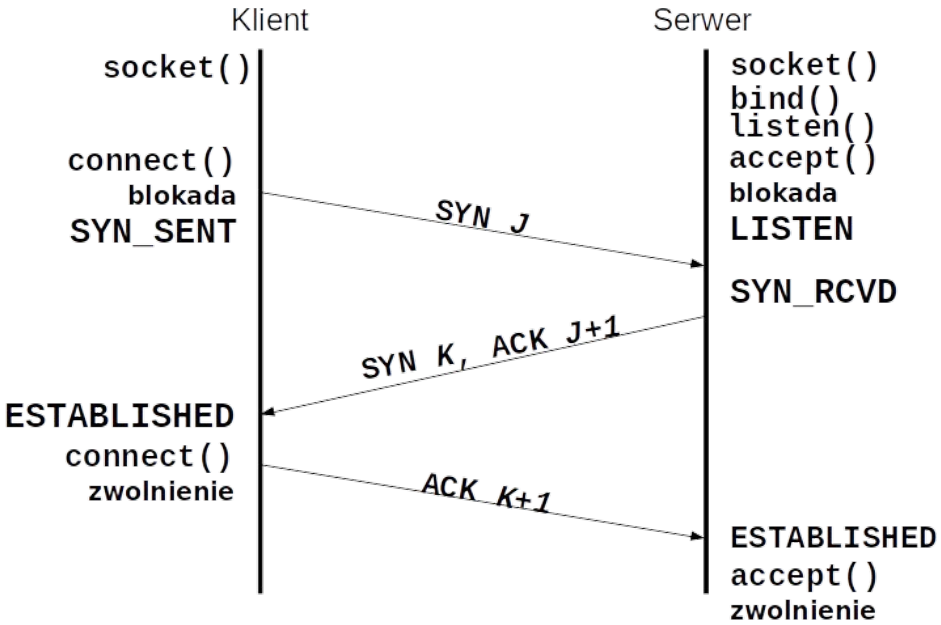
Potwierdzenie ze strony klienta:

- tcpdump: ACK
- serwer C/C++: `accept()` (zwolnienie)

Uzgodnienie trójfazowe



Stany TCP - nawiązanie połączenia



Przesyłanie danych

Serwer oczekuje na przyjęcie danych:

- serwer C/C++: read() (blokada)

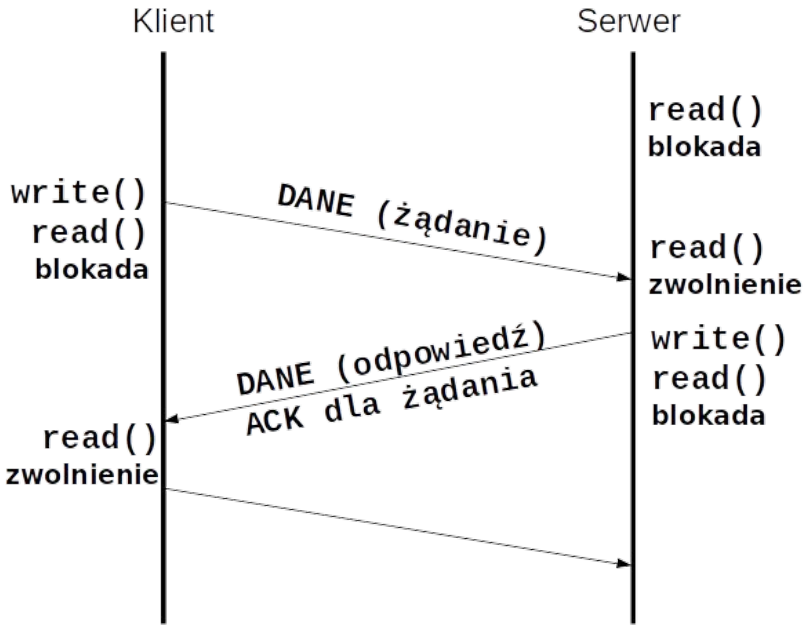
Wysłanie żądania, oczekiwanie na odpowiedź:

- klient C/C++: write(), read() (blokada)
- serwer C/C++: read() (zwolnienie)

Przetworzenie żądania, odpowiedź:

- serwer C/C++: write(), read() (blokada)
- klient C/C++: read() (zwolnienie)
- tcpdump: ACK dla żądania, ACK dla odpowiedzi.

Przesyłanie danych



Zakończenie połączenia TCP

Zamknięcie aktywne (ang. active close):

- C/C++: close()
- tcpdump: FIN

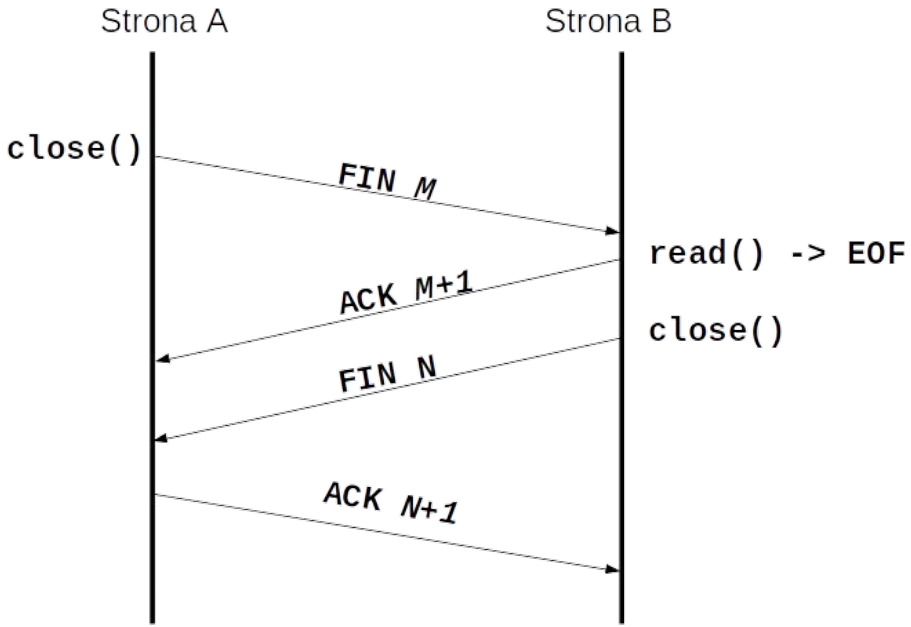
Zamknięcie bierne (ang. passive close):

- tcpdump: ACK
- C/C++: read() -> EOF
- ... odstęp czasu ...
- C/C++: close()
- tcpdump: FIN

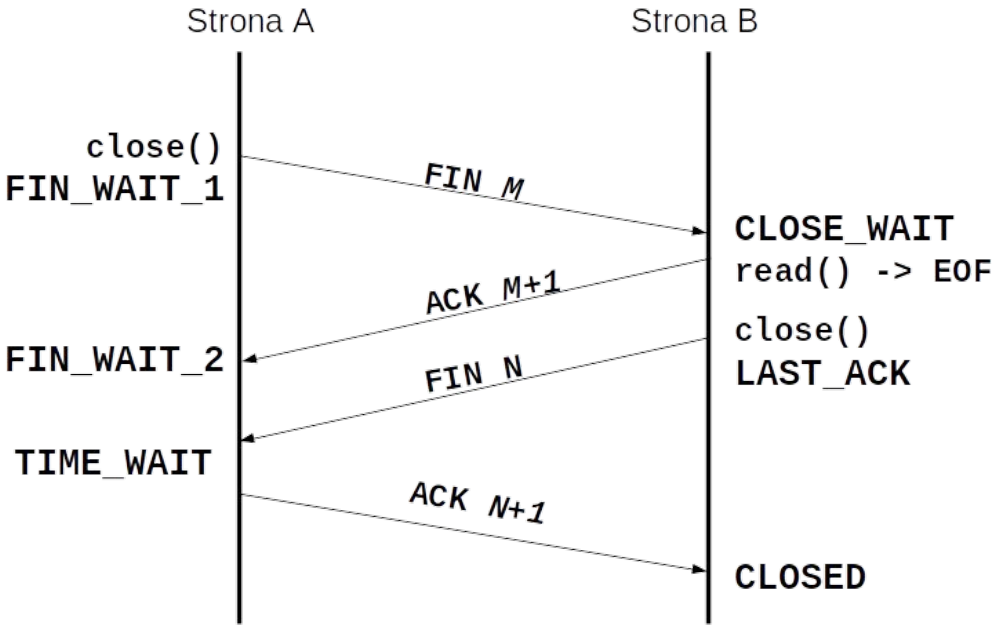
Potwierdzenie zamknięcia:

- tcpdump: ACK

Zakończenie połączenia



Zakończenie połączenia



Dziękuję za uwagę ;)


Tatry