

Programowanie sieciowe

dr Tomasz Tyrakowski

Dyżury: wtorki 12:00 – 13:00
czwartki 14:00 – 15:00
pokój B4-5

e-mail: ttomek@amu.edu.pl

materiały: <http://www.amu.edu.pl/~ttomek>

Wymagania

- podstawowa znajomość języka C
- podstawowa znajomość języka Java
- konta studenckie umożliwiające korzystanie ze stacji roboczych w systemach Linux i Windows XP
- Podstawowa znajomość systemów Linux i Windows XP (edycja kodu źródłowego, kompilacja i uruchamianie programów)

Warunki zaliczenia

- Czynne uczestnictwo w zajęciach laboratoryjnych.
- Test końcowy – pisemny.

Plan przedmiotu

- Programowanie z wykorzystaniem gniazd BSD (system Linux, język C)
- Programowanie z wykorzystaniem WinSock (system Windows XP, język C)
- Zdalne wywoływanie procedur – standard RPC (system Linux, język C)
- Programowanie gniazd w języku Java (system Linux/Windows, język Java)

Plan przedmiotu c.d.

- Zdalne wywoływanie metod w Javie (system Linux/Windows, język Java)
- Programowanie usług sieciowych z wykorzystaniem .NET Remoting (system Windows, język C#)

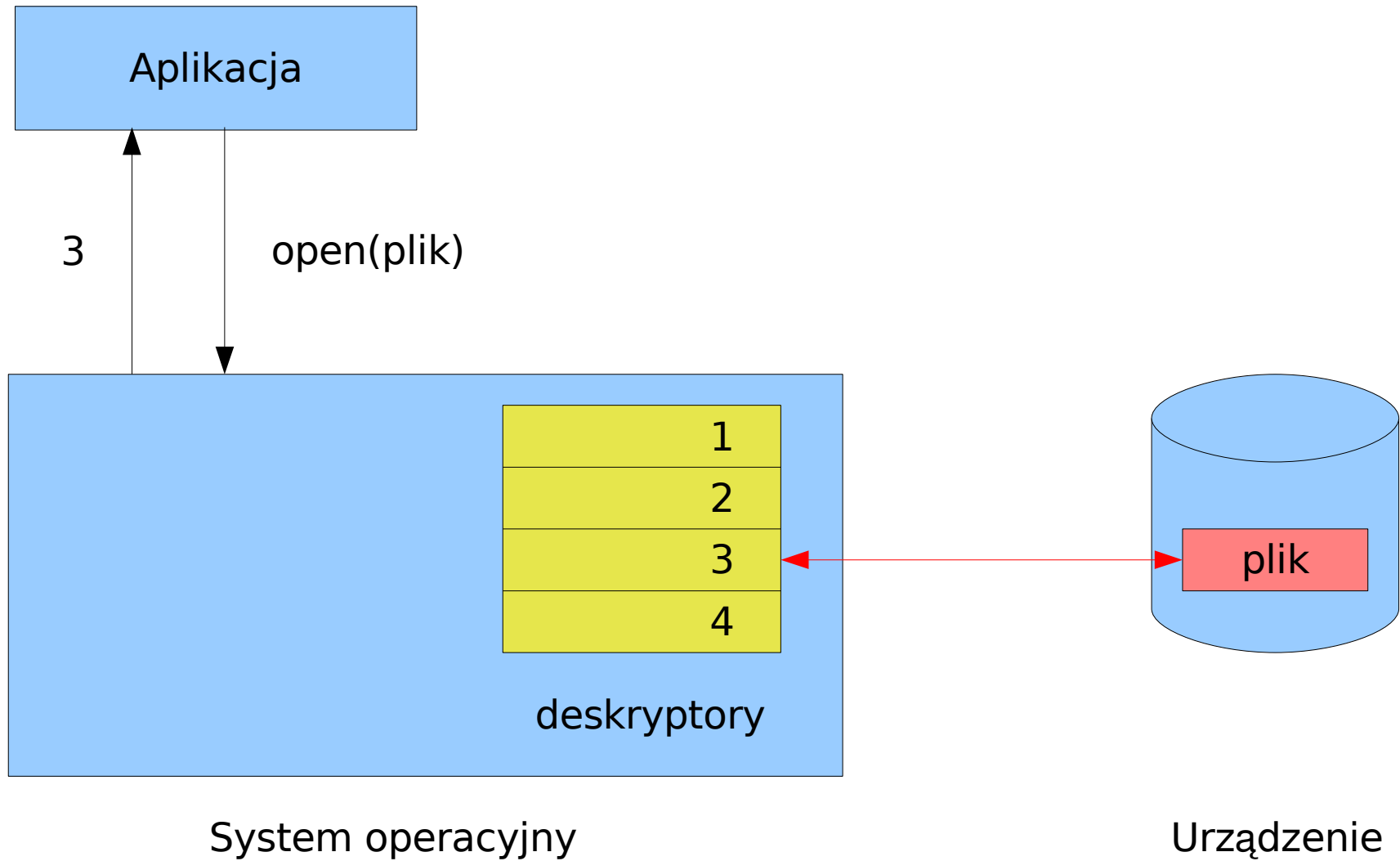
Literatura

- R. Stevens „Programowanie zastosowań sieciowych w systemie Unix”
- A. Jones, J. Ohlund „Programowanie sieciowe Microsoft Windows”
- M. Gabassi, B. Dupouy „Przetwarzanie rozproszone w systemie Unix”
- E. Harold „Java: programowanie sieciowe”
- S. McLean, J. Naftel, K. Williams „Microsoft .NET Remoting”

Operacje na plikach

```
int deskryptor;  
  
deskryptor = open(ścieżka, tryb);  
read(deskryptor, bufor1,  
      liczba bajtów);  
write(deskryptor, bufor2,  
       liczba bajtów);  
  
...  
close(deskryptor);
```

Operacje na plikach



Gniazda (sockets)

- Gniazdo identyfikuje połączenie sieciowe w aplikacji – deskryptor gniazda jest niezbędny w kontaktach aplikacji z systemem operacyjnym.
- Obsługa gniazd jest analogiczna do obsługi plików.
- Występują pewne różnice (inne funkcje, parametry itp.) - połączenia sieciowe to jednak nieco inne obiekty niż pliki.

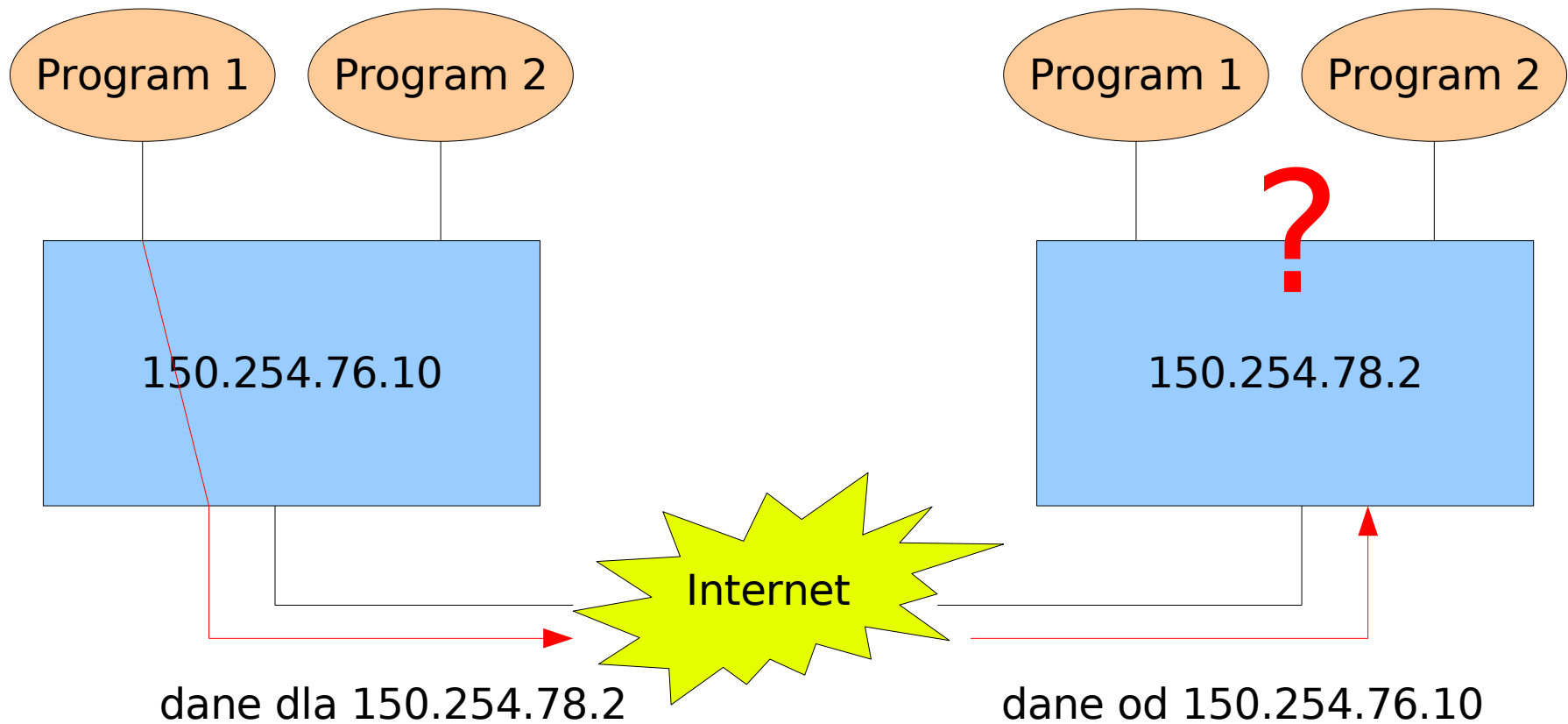
Operacje na gniazdach

```
int deskryptor;  
  
deskryptor = socket(typ gniazda);  
connect(deskryptor, adres);  
send(deskryptor, bufor, ile);  
recv(deskryptor, bufor, ile);  
...  
close(deskryptor);
```

Adres IP

- Liczba 32-bitowa (4 bajty) bez znaku.
- Jednoznacznie identyfikuje komputer w sieci Internet.
- W interakcji z użytkownikiem przedstawiany jako cztery liczby rozdzielone kropkami
- Np. 270544960 = 16.32.48.64
270544960 = 10203040h
10h=16, 20h=32, 30h=48, 40h=64

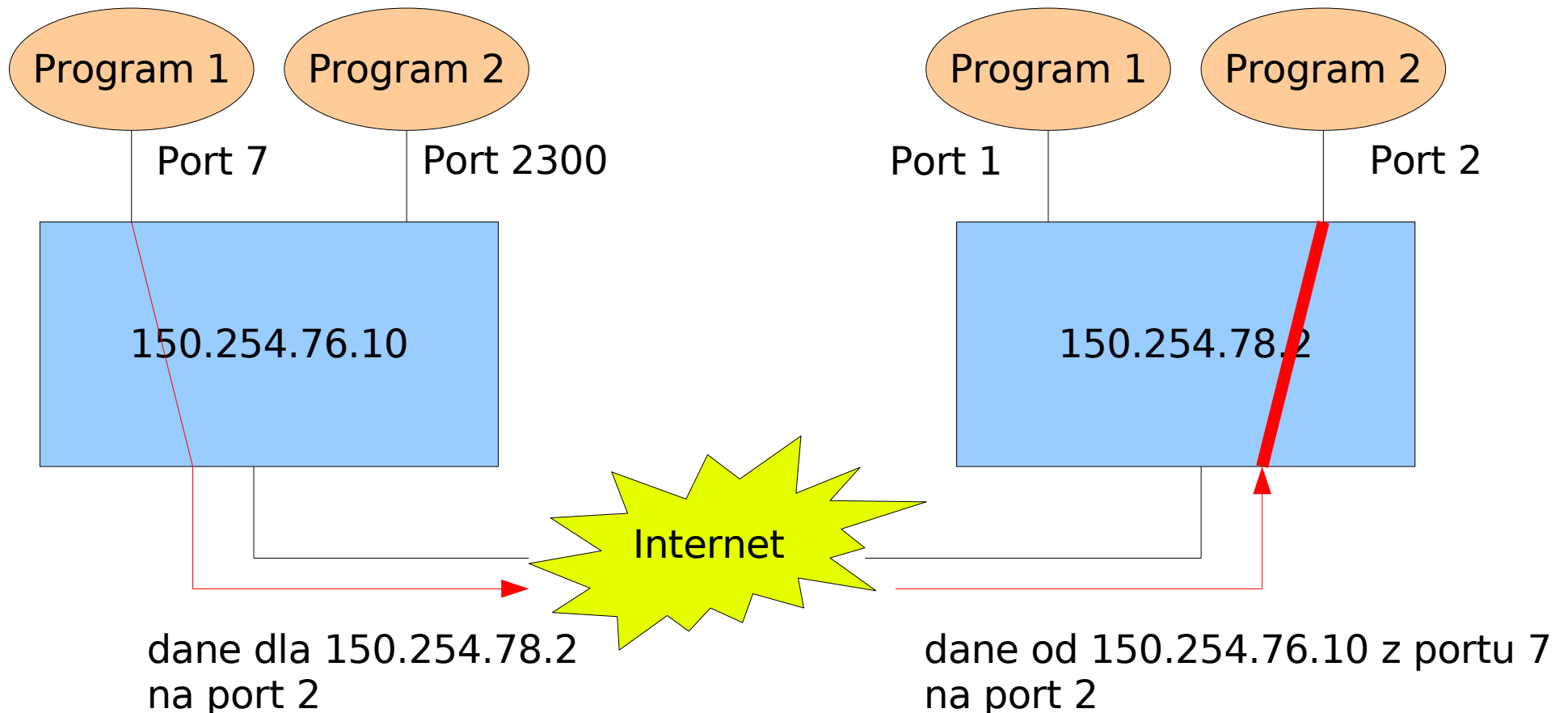
Adres IP c.d.



Adres IP c.d.

- Adres IP nadawcy informuje odbiorcę z którego komputera wysłano dane, jednak nic nie mówi o tym, który program na komputerze nadawcy je wysłał.
- Adres IP odbiorcy pozwala nadawcy określić komputer, który ma otrzymać dane ale nie konkretny program na komputerze odbiorcy, który ma je otrzymać.

Numer portu



Numer portu

- Liczba 16-bitowa (2 bajty), bez znaku.
- Jednoznacznie identyfikuje konkretne połączenie sieciowe w ramach jednego adresu IP.
- Para (adres IP, numer portu) jednoznacznie określa komputer w sieci Internet oraz konkretną aplikację na tym komputerze (zarówno w przypadku nadawcy, jak i odbiorcy).

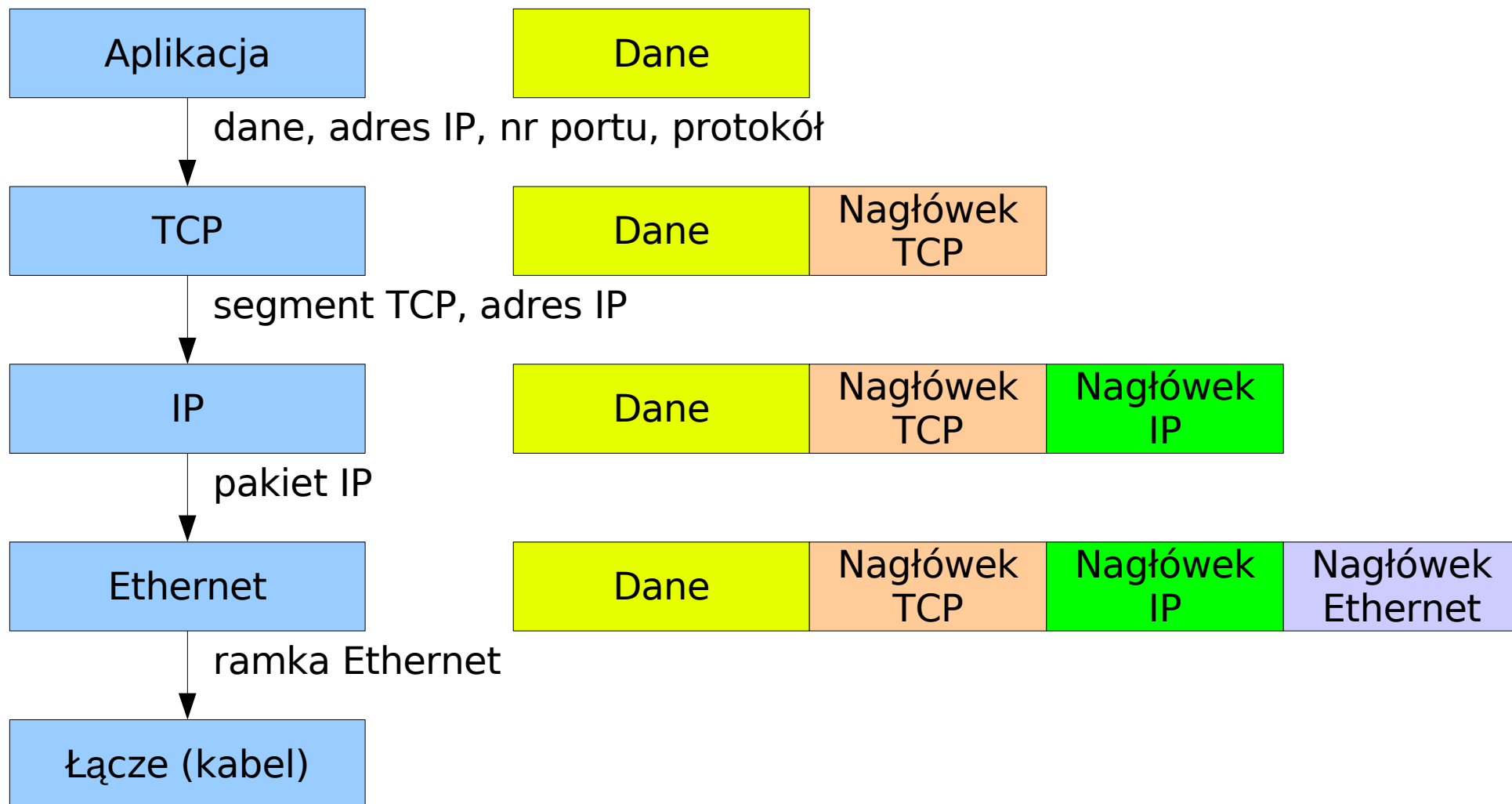
Protokoły transportowe

- Protokół UDP (user datagram protocol):
 - Brak kontroli dostarczania danych.
 - Mniejszy narzut komunikacyjny.
- Protokół TCP (transmission control protocol):
 - Zapewnia dostarczenie danych we właściwej kolejności i retransmisje.
 - Jest nieco bardziej kosztowny (potwierdzenia) niż UDP.

Protokoły transportowe

- TCP – połączenia na duże odległości, z wieloma hostami pośrednimi (routerami), zastosowania, w których wymagany jest wysoki stopień niezawodności kosztem pewnej utraty wydajności
- UDP – sieci lokalne (prawdopodobieństwo utraty danych „w kablu” jest niewielkie), zastosowania, w których najważniejsza jest wydajność (np. sieciowe systemy plików)

Stos protokołów (przykład)



Funkcje systemowe

- Konwersje między formatem lokalnym i sieciowym.
- Tworzenie i niszczenie gniazd.
- Nasłuch gniazda na sieci.
- Wykonanie połączenia.
- Wysyłanie i odbieranie danych.
- Obsługa nazw domenowych, manipulacja adresami.

Konwersje

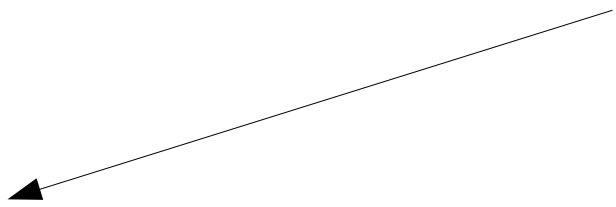
Intel: little endian

$$4128 = 1020h = 20h \mid 10h = 32 \mid 16$$

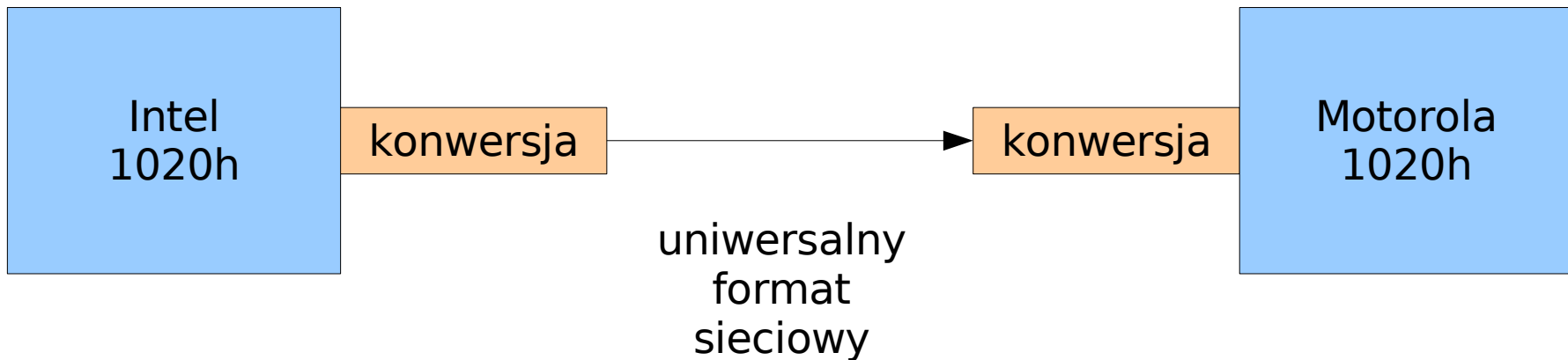
Motorola: big endian

$$4128 = 1020h = 10h \mid 20h = 16 \mid 32$$

$$4128 = 20h10h(\text{Intel})$$


$$(\text{Motorola}) 20h10h = 8208$$

Konwersje c.d.



format sieciowy = big endian

Konwersje c.d.

Z formatu hosta na format sieciowy:

- `long htonl(long)`
- `short htons(short)`

Z formatu sieciowego na format hosta:

- `long ntohl(long)`
- `short ntohs(short)`

Konwersje - przykład

```
#include <stdio.h>
#include <netinet/in.h>

int main(void) {
    short h, n;
    h = 0x1020; /* 4128 */
    n = htons(h);
    printf("%x\n", n);
    return 0;
}
```

wynik: 2010 = 8208

Tworzenie gniazd

```
int socket(int domain,  
           int type,  
           int protocol)
```

domain: **PF_INET**, PF_IPX, PF_APPLETALK

type: **SOCK_STREAM**, **SOCK_DGRAM**

protocol: **0**

Niszczzenie gniazd

```
int close(int socket)
```

Zarówno `socket`, jak i `close` zwracają **-1** w przypadku błędu.

Nasłuch na sieci - schemat

Gniazdo A.

Utworzenie gniazda, przypisanie mu numeru portu, na którym będzie nasłuchiwać, rozpoczęcie oczekiwania na połączenia.

Po nawiązaniu połączenia tworzone jest automatycznie **nowe gniazdo B**, które służy do wymiany danych z jednym konkretnym nadawcą, po czym jest niszczone.

Po zakończeniu komunikacji **gniazdo A** ponownie przechodzi w stan oczekiwania.

Nasłuch na sieci

```
int bind(int socket,  
         struct sockaddr *adres,  
         socklen_t addrlen)
```

W strukturze `sockaddr` powinien znaleźć się lokalny adres IP oraz numer portu, na którym dane gniazdo ma nasłuchiwać.

sockaddr_in

```
struct sockaddr_in {  
    sa_family_t sin_family; /*AF_INET*/  
    u_int16_t    port;  
    struct in_addr sin_addr;  
};
```

```
struct in_addr {  
    u_int32_t s_addr;  
};
```

Nasłuch na sieci c.d.

Jeśli w polu `sin_addr` podamy stałą `INADDR_ANY`, to nasłuch będzie prowadzony na wszystkich adresach, jakie posiada dany host.

Port powinien mieć numer ≥ 1024 (porty o niższych numerach są zarezerwowane dla usług systemowych).

bind - przykład

```
#include <netinet/in.h>
#include <sys/socket.h>
[... ]
int s = socket(AF_INET, SOCK_STREAM,
               0);
struct sockaddr_in adr;
adr.sin_family = AF_INET;
adr.sin_port = 1025;
adr.sin_addr.s_addr = INADDR_ANY;
bind(s, (struct sockaddr*) &adr,
      sizeof(adr));
```

Nasłuch na sieci c.d.

```
int listen(int socket, int  
queue_len)
```

Gniazdo `socket` rozpoczyna nasłuch. Drugi parametr określa ile maksymalnie połączeń może oczekiwać w kolejce na obsługę przez aplikację.

Nasłuch na sieci c.d.

```
int accept(int socket,  
           struct sockaddr *adres,  
           socklen_t  
*dlugosc_adresu)
```

Czeka na nadchodzące połączenie, po czym zwraca deskryptor nowo utworzonego gniazda, a w adres podaje dane (host, port) nawiązującego połączenie.

Blokuje aplikację dopóki ktoś nie spróbuje nawiązać połączenia.

Nasłuch TCP - schemat

- `s = socket(...)`
- `bind(s, ...)`
- `listen(s, ...)`
- `nowy_s = accept(s, ...)` ←
 - obsługa połączenia za pomocą `nowy_s`
 - `close(nowy_s)`
- _____

Wykonanie połączenia TCP

```
int connect(int socket,  
            struct sockaddr *adres,  
            socklen_t dlugosc_adr)
```

Jako adres podajemy strukturę `sockaddr_in`, odpowiednio rzutując wskaźniki aby uniknąć błędów kompilacji.

Wysyłanie danych - TCP

```
ssize_t send(int socket,  
             void *bufor,  
             size_t liczba_bajtow,  
             int flagi)
```

Wysyła `liczba_bajtow` bajtów danych przez łącze identyfikowane przez `socket` (już połączone przez `connect`) spod adresu wskazywanego przez `bufor`. Flagi mają zwykle wartość 0.

Wysyłanie danych - UDP

```
ssize_t sendto(int socket,  
               void *bufor,  
               size_t liczba_bajtow,  
               int flagi,  
               struct sockaddr *odb,  
               socklen_t dl_odb)
```

Odbieranie danych - TCP

```
ssize_t recv(int socket,  
             void *bufor,  
             size_t max_dlugosc,  
             int flagi)
```

Odbiera maksymalnie max_dlugosc bajtów, zwraca faktyczną liczbę przeczytanych bajtów. Dane umieszcza w buforze.

Odbieranie danych - UDP

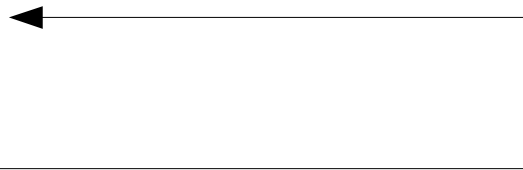
```
ssize_t recvfrom(int socket,  
                 void *bufor,  
                 size_t max_dlugosc,  
                 int flagi,  
                 struct sockaddr *odkogo,  
                 socklen_t *dl_odkogo)
```

Nasłuchiwanie UDP - schemat

- `s = socket(...)`

- `bind(s, ...)`

- `recvfrom(s, ...)`



- _____

- `close(s)`

Adresy

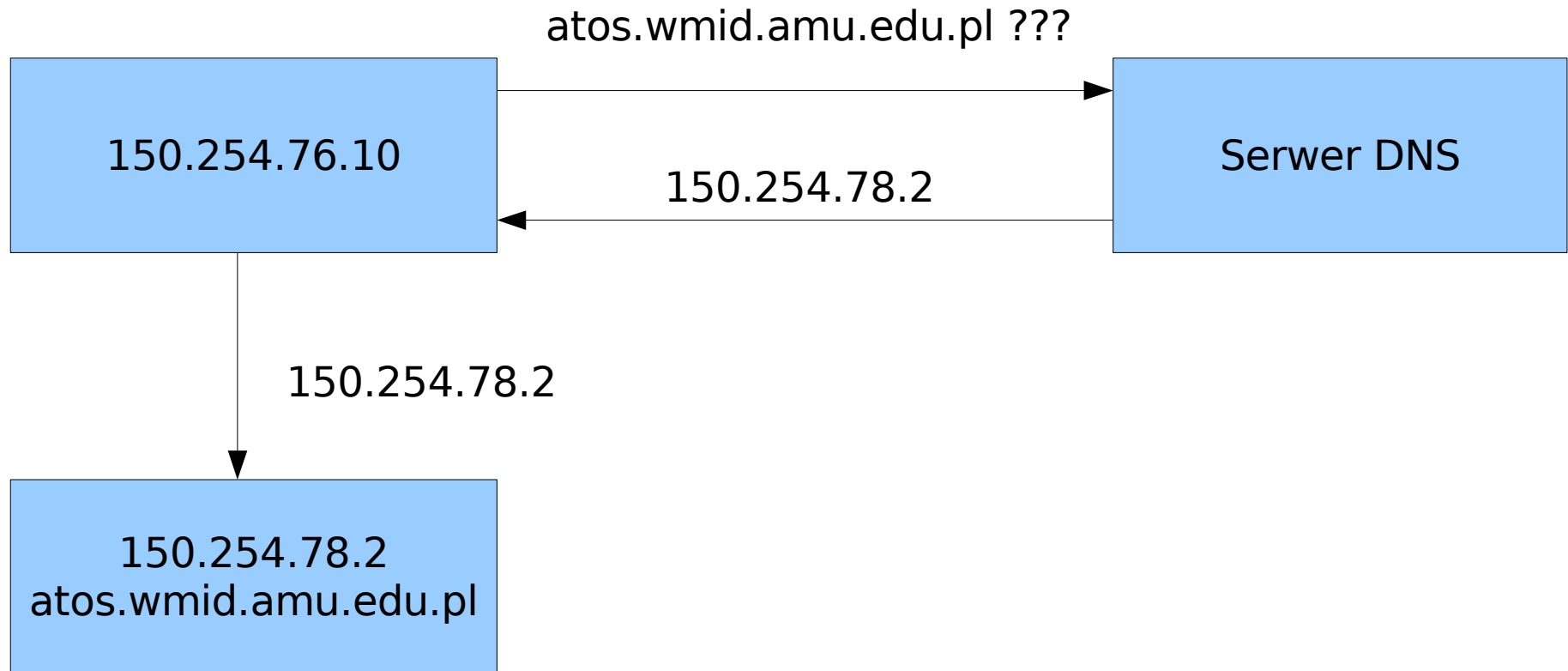
```
in_addr_t inet_addr(char *adr)
```

Zamiana a.b.c.d na adres w postaci 32-bitowej.

```
char* inet_ntoa(struct in_addr adr)
```

Zamiana adresu 32-bitowego na łańcuch postaci a.b.c.d.

Nazwy domenowe



DNS – Domain Name Service

Nazwy domenowe

```
#include <netdb.h>

struct hostent* gethostbyname(
                                char *nazwa)

struct hostent {
    char* h_name,
    ...
    char **h_addr_list;
}

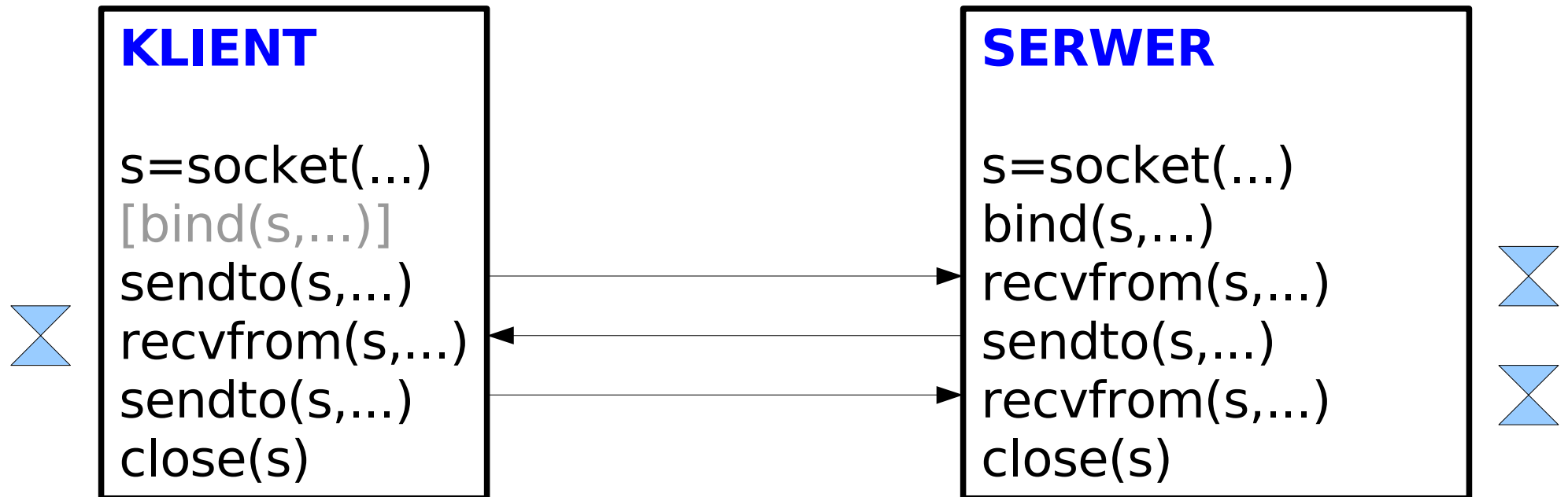
Makro h_addr to skrót h_addr_list[0]
```

Nazwy domenowe c.d.

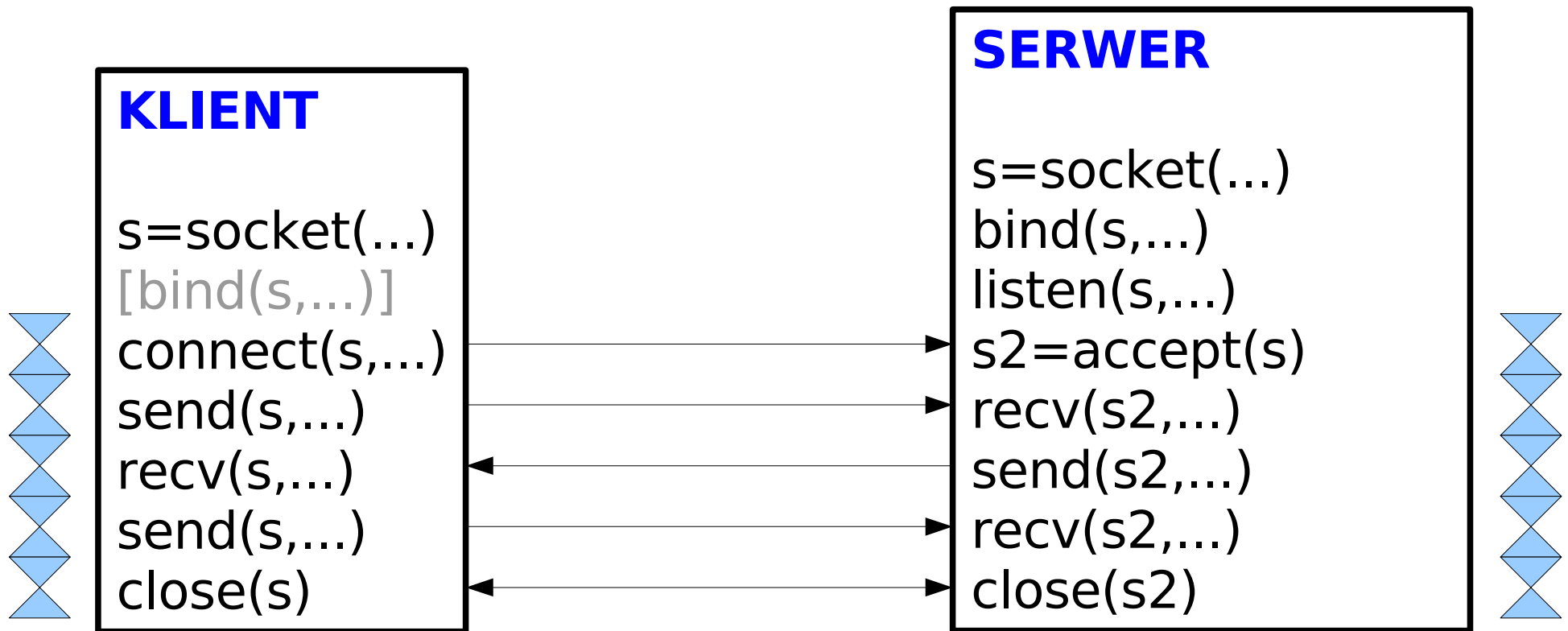
```
struct sockaddr_in adr;  
struct hostent *he;  
adr.sin_family = AF_INET;  
adr.sin_port = htons(2000);  
he = gethostbyname(  
    "atos.wmid.amu.edu.pl"  
);  
adr.sin_addr.s_addr =  
*((u_int32_t*) he->h_addr);
```

Schemat dla UDP

 = możliwe zablokowanie aplikacji



Schemat dla TCP



Funkcja select

Problem:

Funkcje `send`, `recv`, `sendto`, `recvfrom` i `accept` są blokujące. Nie jest możliwa implementacja programu, który nasłuchuje jednocześnie na TCP i UDP.

Rozwiązanie:

Funkcja `select`.

Funkcja select c.d.

```
#include <sys/select.h>
```

```
lub
```

```
#include <unistd.h>
```

```
#include <sys/time.h>
```

```
int select(int n,  
           fd_set *readfds,  
           fd_set *writefds,  
           fd_set *exceptfds,  
           struct timeval *timeout)
```

Funkcja select c.d.

`FD_CLR(int fd, fd_set *set)`
Usuń deskryptor fd ze zbioru set.

`FD_ISSET(int fd, fd_set *set)`
Czy fd znajduje się w zbiorze set?

`FD_SET(int fd, fd_set *set)`
Dodaj fd do zbioru set.

`FD_ZERO(fd_set *set)`
Wyczyść cały zbiór set.

Funkcja select c.d.

```
struct timeval {  
    long tv_sec;    /* sekundy */  
    long tv_usec;   /* mikrosekundy */  
}
```

Funkcja select c.d.

- Funkcja select zwraca liczbę deskryptorów, na których zaszły zdarzenia.
- Dany zbiór deskryptorów może być ignorowany (podajemy NULL).
- Podanie NULL jako czasu oczekiwania powoduje natychmiastowy powrót.
- Podanie zerowego czasu oczekiwania oznacza czekanie do skutku.

Funkcja select c.d.

- Zwracana wartość to liczba deskryptorów, na których wystąpiło zdarzenie. 0 oznacza przeterminowanie (minął czas a nic się nie wydarzyło), a -1 to błąd (np. nieprawidłowe deskryptory w zbiorach).
- Dzięki `select` możliwy jest nasłuch na wielu gniazdach jednocześnie.

Funkcja select - przykład

```
[...]
int dtcp, dudp, maxd, ret;
struct timeval tv;
fd_set zbior;
dtcp = socket(AF_INET, SOCK_STREAM,
              0);
dudp = socket(AF_INET, SOCK_DGRAM,
              0);
[ bind, listen na dtcp
  ale NIE accept ]
```

Funkcja select c.d.

```
maxd = (dtcp > dudp ? dtcp : dudp);  
tv.tv_sec = tv.tv_usec = 0;  
FD_ZERO(&zbior);  
FD_SET(dtcp, &zbior);  
FD_SET(dudp, &zbior);  
ret = select(maxd, &zbior, NULL,  
             NULL, &tv);
```

Funkcja select c.d.

```
if (ret > 0) {  
    if (FD_ISSET(dtcp, &zbior)) {  
        [accept, send, recv, ...]  
    }  
    if (FD_ISSET(dudp, &zbior)) {  
        [recvfrom, sendto, ...]  
    }  
}
```