Programowanie sieciowe
Wprowadzenie

Michał Kalewski

Instytut Informatyki
Politechnika Poznańska
ul. Piotrowo 2, 60–965 Poznań

mkalewski@cs.put.poznan.pl
http://www.cs.put.poznan.pl/mkalewski

Poznań · 26 luty 2019 r.

Plan wykładu

Informacje o przedmiocie

Programowanie sieciowe

Przedmiot obligatoryjny na pierwszym roku specjalności systemy rozproszone.

Przedmiot obejmuje 30 godz. wykładów oraz 30 godz. zajęć laboratoryjnych.

Nakład pracy w ramach przedmiotu przekłada się na 4 punkty ECTS.

Karta opisu modułu kształcenia dostępna jest na stronie internetowej Wydziału Informatyki: http://www.fc.put.poznan.pl/.

Zaliczenie wykładów, w formie kolokwium pisemnego, odbędzie się na ostatnim wykładzie w semestrze.

Materiały dydaktyczne do zajęć dostępne są w Internecie pod następującym adresem: http://www.cs.put.poznan.pl/mkalewski/.

Literatura przedmiotu

1. W. Richard Stevens, Bill Fenner, Andrew M. Rudoff. **Unix Network Programming, Volume 1: The Sockets Networking API**, 3rd Edition. *Addison-Wesley Professional*, 2003, http://www.unixnetworkprogramming.com/.
2. Douglas E. Comer, David L. Stevens. **Internetworking with TCP/IP, Volume III: Client-Server Programming and Applications, Linux/Posix Sockets Version**, 1st Edition. *Pearson*, 2000.
3. Christian Benvenuti. **Understanding Linux Network Internals**, 1st Edition. *O'Reilly Media*, 2006.
4. Kay Robbins, Steve Robbins. **Unix Systems Programming: Communication, Concurrency and Threads**, 2nd Edition. *Prentice Hall*, 2015.
5. Michael Kerrisk. **The Linux Programming Interface: A Linux and Unix System Programming Handbook**, 1st Edition. *No Starch Press*, 2010.

Literatura uzupełniająca podawana będzie na bieżąco w materiałach dydaktycznych.

Linux kernel source tree

Kody źródłowe jądra Linux dostępne są w serwisie GitHub pod następującym adresem: https://github.com/torvalds/linux.

POBRANIE:

```
$ git clone git@github.com:torvalds/linux.git

$ git clone https://github.com/torvalds/linux.git
```

Dokumentacja kodu źródłowego dostępna jest następującym adresem: https://www.kernel.org/doc/html/latest/.

Kody źródłowe systemu operacyjnego macOS (oraz iOS i OS X Server) dostępne są pod następującym adresem: https://opensource.apple.com.

Plan przedmiotu

Zagadnienia ogólne

- Protokoły i mechanizmy programistyczne na różnych warstwach sieciowego modelu odniesienia ISO/OSI (warstwy od drugiej do siódmej).
- Architektury i konstrukcje efektywnych procesów (serwerów) sieciowych.
- Programowanie sieciowe w systemach operacyjnych GNU/Linux, UNIX, Windows, Windows Phone, Android oraz iOS.
- Programowalne sieci komputerowe SDN (ang. *Software-Defined Networking*).

Tematyka wykładów (1/2)

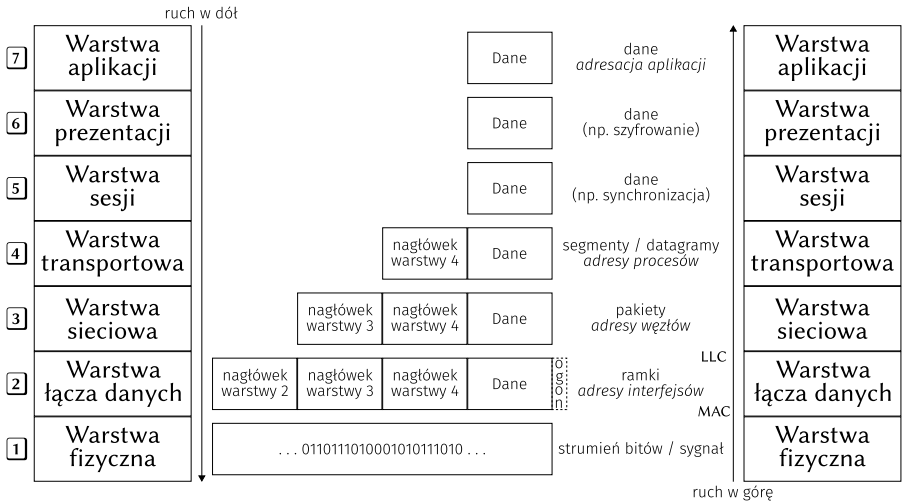
- Obsługa interfejsów sieciowych.
- Gniazda sieciowe PF_PACKET.
- Filtracja ramek nasłuchiwanych (biblioteka `libpcap`).
- Transmisja ramek z pakietami warstw wyższych (biblioteka `libnet`).
- Obsługa tablicy routingu i pamięci podręcznej protokołu ARP.
- Gniazda sieciowe PF_NETLINK.
- Nieprzetworzone gniazda sieciowe.
- Stream Control Transmission Protocol.
- Internet Protocol version 6.

Tematyka wykładów (2/2)

- Buforowanie w komunikacji sieciowej.
- Obsługa operacji wejścia/wyjścia komunikacji sieciowej.
- Architektury programów sieciowych.
- Filtracja i szyfrowanie komunikacji internetowej (projekty netfilter/iptables i OpenSSL).
- Mechanizmy obsługi komunikacji sieciowej w aplikacjach z graficznym interfejsem użytkownika.
- Komunikacja internetowa z zastosowaniem protokołów warstwy aplikacji (biblioteka `libcurl`).
- Wprowadzenie do programowalnych sieci komputerowych SDN.

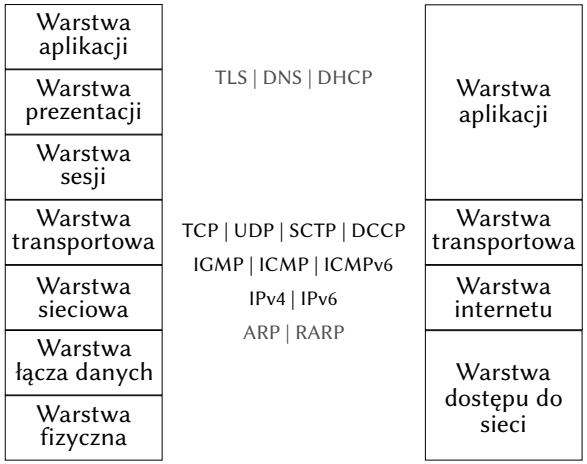
Repetytorium z sieci komputerowych

Model warstwowy ISO/OSI



Rysunek 1: Model warstwowy ISO/OSI (ang. *Open Systems Interconnection*).

Warstwowy model internetowy



Rysunek 2: Warstwowy model internetowy i jego porównanie z modelem ISO/OSI.

Zestaw protokołów internetowych (1/3)

Lista protokołów aplikacyjnych, nazywanych *usługami*, wraz z przypisanymi do nich numerami portów i protokołami warstwy transportowej utrzymywana jest przez *Internet Assigned Numbers Authority* (IANA) działającą przy *Internet Corporation for Assigned Names and Numbers* (ICANN).

Lista ta zawarta jest w dokumencie pt. „Service Name and Transport Protocol Port Number Registry” i udostępniana jest w Internecie pod następującym adresem: http://www.iana.org/assignments/service-names-port-numbers/service-names-port-numbers.xhtml.

IANA utrzymuje także listę numerów protokołów warstwy transportowej. Numery te oznaczają wartości pola `Protocol` w nagłów pakietu protokołu IPv4 lub pola `NextHeader` w nagłówku pakietu protokołu IPv6.

Lista ta zawarta jest w dokumencie pt. „Protocol Numbers” i udostępniana jest w Internecie pod następującym adresem: http://www.iana.org/assignments/protocol-numbers/protocol-numbers.xhtml.

Zestaw protokołów internetowych (2/3)

W systemach operacyjnych UNIX oraz GNU/Linux, lista usług przechowywana jest w pliku `/etc/services` o następującym formacie wierszy:

```
nazwa-usługi  numer-portu/protokół-transportowy  [alias]
```

zob. `services(5)`

PRZYKŁAD:

```
$ grep "^http" /etc/services
http      80/tcp    www       # WorldWideWeb HTTP
http      80/udp              # HyperText Transfer Protocol
https     443/tcp             # http protocol over TLS/SSL
https     443/udp
http-alt  8080/tcp  webcache  # WWW caching service
http-alt  8080/udp
```


Zestaw protokołów internetowych (3/3)

Lista protokołów transportowych przechowywana jest natomiast w pliku `/etc/protocols` o następującym formacie wierszy:

```
nazwa-protokołu  numer-protokołu  [alias]
```

zob. `protocols(5)`

PRZYKŁAD:

```
$ grep "^tcp\|^udp" /etc/protocols
tcp      6    TCP      # transmission control protocol
udp      17   UDP      # user datagram protocol
udplite  136  UDPLite  # UDP-Lite [RFC3828]
```

EtherType

EtherType to liczba dwu-bajtowa wykorzystywana w ramkach warstwy łącza danych do określenia transportowanego protokołu warstwy sieciowej.

Lista wartości EtherType utrzymywana jest przez *Institute of Electrical and Electronics Engineers* (IEEE) i udostępniana w Internecie pod następującym adresem: http://standards-oui.ieee.org/ethertype/eth.txt.

W systemach operacyjnych UNIX oraz GNU/Linux, lista wartości EtherType zapisana jest m.in. w pliku `if_ether.h`.

zob. `ethers(5)`

PRZYKŁAD:

```
$ cat ./if_ether.h | tr -s " " "\t" | cut -f 2,3 | grep "_IP"
ETH_P_IP    0x0800
ETH_P_IPX   0x8137
ETH_P_IPV6  0x86DD
```

https://github.com/torvalds/linux/blob/master/include/uapi/linux/if_ether.h

Podstawowe gniazda sieciowe

Korzystanie z mechanizmów komunikacji internetowej wymaga od programisty utworzenia obiektu zwanego *gniazdem sieciowym*, które traktowane jest przez system operacyjny jak plik specjalny.

Podstawowe gniazda sieciowe umożliwiają komunikację *strumieniową* (z użyciem protokołów TCP/IP) oraz *datagramową* (z użyciem protokołów UDP/IP).

Rysunek 3: Gniazda sieciowe w odniesieniu do warstwowego modelu internetowego.

Funkcja systemowa `socket(2)`

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

Parametry:

`domain` określa tzw. *domenę komunikacyjną*; domeny takie zadeklarowane są w pliku nagłówkowym `sys/socket.h`, a dla gniazd podstawowych protokołu IPv4 domena komunikacyjna określona jest stałą `PF_INET`.

`type` określa typ (semantykę) komunikacji we wskazanej domenie komunikacyjnej; dla gniazd podstawowych dostępne są dwa typy: *strumieniowy* (z użyciem protokołu TCP) określony stałą `SOCK_STREAM` oraz *datagramowy* (z użyciem protokołu UDP) określony stałą `SOCK_DGRAM`.

`protocol` określa numer protokołu dla wybranej domeny komunikacyjnej oraz wybranego typu komunikacji; dla gniazd podstawowych przyjmuje wartość `0`.

Wartość zwracana: numer *deskryptora gniazda* lub `-1` w przypadku błędu.

Deskryptor gniazda sieciowego (1/3)

W systemach operacyjnych UNIX oraz GNU/Linux, gniazdo sieciowe jest plikiem specjalnym, do którego programista uzyskuje deskryptor.

Lista deskryptorów plików procesu o identyfikatorze `[pid]` przechowywana jest w katalogu `/proc/[pid]/fd`.

Katalog ten zawiera dowiązania symboliczne o nazwach odpowiadających numerom deskryptorów plików procesu.

Każde takie dowiązanie symboliczne wskazuje na plik, z którym powiązany jest dany deskryptor.

W przypadku gniazd sieciowych, dowiązany plik zawiera etykietę typu (`socket`) oraz numer *i-węzła* danego gniazda sieciowego.

I-węzły gniazd sieciowych przechowywane są w plikach w katalogu `/proc/net`.

zob. `proc(5)`

Deskryptor gniazda sieciowego (2/3)

Przykład:

```
$ ./server-tcp &
[1] 21627
$ ls -al /proc/21627/fd
total 0
dr-x------ 2 user users  0 Feb 17 11:02 .
dr-xr-xr-x 9 user users  0 Feb 17 11:02 ..
lrwx------ 1 user users 64 Feb 17 11:04 0 -> /dev/pts/0
lrwx------ 1 user users 64 Feb 17 11:04 1 -> /dev/pts/0
lrwx------ 1 user users 64 Feb 17 11:02 2 -> /dev/pts/0
lrwx------ 1 user users 64 Feb 17 11:04 3 -> socket:[69705]
$ readlink /proc/21627/fd/3
socket:[69705]
$ cat /proc/21627/fdinfo/3
pos:    0
flags:  02
```


Deskryptor gniazda sieciowego (3/3)

Przykład (ciąg dalszy):

```
$ head -n1 < /proc/net/tcp ; grep -a "69705" /proc/net/tcp
sl  local_address rem_address   st tx_queue rx_queue tr tm->when
retrnsmt   uid  timeout inode
1:  00000000:04D2 00000000:0000 0A 00000000:00000000 00:00000000
00000000  1000        0 69705 1 0000000000000000 100 0 0 10 0

$ netstat -atpn
Proto Recv-Q Send-Q Local Address Foreign Address  State
PID/Program name
tcp        0      0 0.0.0.0:1234  0.0.0.0:*        LISTEN
21627/server-tcp
```

zob. `netstat(8)`

</> Interfejs gniazd BSD — serwer TCP

```c
int sfd, cfd;
socklen_t sl;
struct sockaddr_in saddr, caddr;
memset(&saddr, 0, sizeof(saddr));
saddr.sin_family = AF_INET;
saddr.sin_addr.s_addr = INADDR_ANY;
saddr.sin_port = htons(1234);
sfd = socket(PF_INET, SOCK_STREAM, 0);
bind(sfd, (struct sockaddr*) &saddr, sizeof(saddr));
listen(sfd, 5);
while(1) {
  sl = sizeof(caddr);
  cfd = accept(sfd, (struct sockaddr*) &caddr, &sl);
  write(cfd, "Hello World!\n", 13);
  close(cfd);
}
```


</> Interfejs gniazd BSD — klient TCP

```c
int sfd;
char buf[128];
struct sockaddr_in saddr;
struct hostent* addrent;
addrent = gethostbyname(argv[1]);
memset(&saddr, 0, sizeof(saddr));
saddr.sin_family = AF_INET;
saddr.sin_port = htons(atoi(argv[2]));
memcpy(&saddr.sin_addr.s_addr, addrent->h_addr,
       addrent->h_length);
sfd = socket(PF_INET, SOCK_STREAM, 0);
connect(sfd, (struct sockaddr*) &saddr, sizeof(saddr));
read(sfd, buf, sizeof(buf));
close(sfd);
```

</> Interfejs gniazd BSD — serwer UDP

```c
int sfd;  socklen_t sl;  char buf[128];
struct sockaddr_in saddr, caddr;
memset(&saddr, 0, sizeof(saddr));
saddr.sin_family = AF_INET;
saddr.sin_addr.s_addr = INADDR_ANY;
saddr.sin_port = htons(1234);
sfd = socket(PF_INET, SOCK_DGRAM, 0);
bind(sfd, (struct sockaddr*) &saddr, sizeof(saddr));
while(1) {
  memset(&caddr, 0, sizeof(caddr));
  memset(&buf, 0, sizeof(buf));
  sl = sizeof(caddr);
  recvfrom(sfd, buf, 128, 0, (struct sockaddr*) &caddr, &sl);
  sendto(sfd, buf, strlen(buf)+1, 0,
         (struct sockaddr*) &caddr, sl);
}
```


</> Interfejs gniazd BSD — klient UDP (1/2)

```c
int sfd;
socklen_t sl;
char buf[128];
struct sockaddr_in saddr, caddr;
struct hostent* addrent;
addrent = gethostbyname(argv[1]);
memset(&caddr, 0, sizeof(caddr));
caddr.sin_family = AF_INET;
caddr.sin_addr.s_addr = INADDR_ANY;
caddr.sin_port = 0;
memset(&saddr, 0, sizeof(saddr));
saddr.sin_family = AF_INET;
memcpy(&saddr.sin_addr.s_addr, addrent->h_addr,
       addrent->h_length);
saddr.sin_port = htons(atoi(argv[2]));
```

cdn.

</> Interfejs gniazd BSD — klient UDP (2/2)

ciąg dalszy

```c
sfd = socket(PF_INET, SOCK_DGRAM, 0);
bind(sfd, (struct sockaddr*) &caddr, sizeof(caddr));
strcpy(buf, "Hello server!\n");
sendto(sfd, buf, strlen(buf)+1, 0, (struct sockaddr*) &saddr,
       sizeof(saddr));
sl = sizeof(saddr);
recvfrom(sfd, buf, 128, 0, (struct sockaddr*) &saddr, &sl);
close(sfd);
```

Funkcja systemowa `gethostbyname(2)` we współczesnych implementacjach powinna być zastąpiona funkcją systemową `getaddrinfo(3)` (która zostanie omówiona na jednym z kolejnych wykładów.)

Funkcja systemowa `gethostbyname(2)` występuje w przykładach tylko na potrzeby zwięzłości implementacji.

32-bitowe oraz 64-bitowe typy danych

Tablica 1: Porównanie 32-bitowych oraz 64-bitowych typów danych.

Typ danych	Model ILP32	Model LP64
char	8	8
short	16	16
int	32	32
long	32	64
pointer	32	64

```c
typedef long unsigned int size_t;
```

zob. `stddef.h`

```c
typedef unsigned int socklen_t;
```

zob. `socket.h`

Stałe AF oraz PF

Prefiks AF odnosi się do *rodziny adresowej* (ang. *address family*), a prefiks PF odnosi się do *rodziny protokołu* (ang. *protocol family*) — domena komunikacyjna.
Podział taki wprowadzono na przypadek gdyby pojedynczy protokół wspierał wiele sposobów adresacji — stała PF niezbędna byłaby wówczas do utworzenia gniazda sieciowego, a stała AF używana byłaby w strukturze adresowej.
W rzeczywistości przypadek taki nigdy nie wystąpił i stałe AF mają te same wartości co stałe PF dla poszczególnych protokołów.

Przykład:

```
$ cat ./socket.h | tr -s " " "\t" | cut -f 2,3 | grep "_INET"
PF_INET   2
PF_INET6  10
AF_INET   PF_INET
AF_INET6  PF_INET6
```

https://github.com/torvalds/linux/blob/master/include/linux/socket.h

Podsumowanie

Podsumowanie

- Modele warstwowe sieci komputerowych: ISO/OSI oraz internetowy.
- Zestaw protokołów internetowych i EtherType.
- Podstawowe gniazda sieciowe:
  - funkcja systemowa `socket(2)`;
  - deskryptor gniazda sieciowego;
  - klient i serwer protokołu TCP;
  - klient i serwer protokołu UDP.
- 32-bitowe oraz 64-bitowe typy danych.
- Stałe AF i PF.

Pytania i zadania

Pytania i zadania — praca własna

1. Czy kapsułkowaniu zawsze musi towarzyszyć fragmentacja i na jakich warstwach modelu warstwowego ISO/OSI może ona wystąpić?
2. Jakie są różnice pomiędzy komunikacją strumieniową a komunikacją datagramową?
3. Sprawdź zawartość plików `/etc/protocols`, `/etc/services` oraz `if_ether.h`[1] w systemie operacyjnym UNIX lub GNU/Linux.
4. Sprawdź co oznacza wartość EtherType równa `0x8870`.
5. Czy implementując komunikację datagramową (bezpołączeniową) z zastosowaniem podstawowych gniazd sieciowych możliwe jest użycie funkcji `connect(2)`? Jeżeli tak, to w jakim celu?

[1] Plik nagłówkowy źródeł jądra systemu.

Dziękuję za uwagę!