

Programowanie sieciowe — skrypt do zajęć laboratoryjnych

MICHAŁ KALEWSKI

mkalewski@cs.put.poznan.pl

01 czerwca 2023 r.

Spis treści

Repozytorium GIT	1
Praca z kopią lokalną repozytorium	2
Repozytorium materiałów dydaktycznych do zajęć laboratoryjnych	2
1 Repetytorium z sieci komputerowych — podstawowe gniazda sieciowe	2
2 Obsługa interfejsów sieciowych funkcją systemową <code>ioctl(2)</code>	3
3 Gniazda sieciowe <code>PF_PACKET</code>	3
4 Filtracja ramek nasłuchiowanych (biblioteka <code>libpcap</code>)	4
5 Transmisja ramek z pakietami warstw wyższych (biblioteka <code>libnet</code>)	4
6 Obsługa tablicy routingu i pamięci podręcznej protokołu ARP	5
7 Gniazda sieciowe <code>PF_NETLINK</code>	6
8 Nieprzetworzone gniazda sieciowe	6
9 Internet Protocol version 6 (IPv6)	6
10 Stream Control Transmission Protocol (SCTP)	7
11 Buforowanie w komunikacji sieciowej	8
12 Obsługa operacji wejścia/wyjścia komunikacji sieciowej	8
13 Architektury serwerów sieciowych	9
PROJEKTY ZALICZENIOWE	9
Tematy projektów zaliczeniowych	9
Kryteria oceny projektów	9
Termin zaliczania projektów	9

Repozytorium GIT

Kody programów realizowanych na zajęciach laboratoryjnych programowania sieciowego należy przechowywać w repozytorium GIT na serwerze gitlab — środowisko [GitLab](https://gitlab.cs.put.poznan.pl) tego serwera dostępne jest pod następującym adresem: <https://gitlab.cs.put.poznan.pl>.

Posługując się środowiskiem [GitLab](https://gitlab.cs.put.poznan.pl), należy utworzyć nowy projekt o nazwie `programowanie-sieciowe`, dodać do niego plik o nazwie `README.md` (odnośnik „adding a README”), a następnie dodać do niego użytkownika (zakładka „Members”) o nazwie `mkalewski` z uprawnieniami *Developer*.

Kody programów z poszczególnych zajęć laboratoryjnych powinny być umieszczane w katalogach, których nazwy są datami tych zajęć w formacie `YYYY-MM-DD`. Nazwy plików z kodami programów powinny odpowiadać numerom zadań z poszczególnych zajęć laboratoryjnych zgodnie z następującym schematem: `zadanie_NN_dowolna_nazwa.c[pp]` (np.: `2023-03-02/zadanie_06_komunikator.c`).

Praca z kopią lokalną repozytorium

Poniżej przedstawiono przykład utworzenia kopii lokalnej repozytorium z serwera gitlab (`git clone`), dodania do niego nowego pliku (`touch`), zatwierdzenia w nim zmian (`git add` i `git commit`) oraz jego synchronizacji z repozytorium na serwerze gitlab (`git push`):

```
$ git clone https://gitlab.cs.put.poznan.pl/mkalewski/programowanie-sieciowe.git
$ cd programowanie-sieciowe/
$ touch test.txt
$ git add test.txt
$ git commit -m "Add test.txt"
$ git push -u origin master
```

UWAGA: dla serwera [GitLab](#) konieczne może okazać się wyłączenie weryfikacji certyfikatu SSL, np.:

```
$ git config --global http.sslVerify false
```

Repozytorium materiałów dydaktycznych do zajęć laboratoryjnych

Materiały dydaktyczne do zajęć laboratoryjnych programowania sieciowego (wraz ze skryptem w formacie PDF) znajdują się w repozytorium GIT na serwerze gitlab: <https://gitlab.cs.put.poznan.pl/mkalewski/ps-2023.git>:

```
$ git clone https://gitlab.cs.put.poznan.pl/mkalewski/ps-2023.git
```

1 Repetytorium z sieci komputerowych — podstawowe gniazda sieciowe

1. Korzystając z kodów w repozytorium GIT, uruchom serwer TCP (`tcp-server.c`) i sprawdź dane systemowe o utworzonym gnieździe sieciowym:

- katalog `/proc/[pid]/fd/`,
- plik `/proc/[pid]/fdinfo/[fd-number]`,
- plik `/proc/net/tcp`,
- polecenie `netstat`:

```
$ netstat -atpn
```

2. Korzystając z programu `nc(1)` zestaw połączenie TCP z serwerem TCP, a następnie sprawdź ponownie informacje o gniazdach sieciowych serwera tak, jak w zadaniu nr 1.

UWAGA: serwer TCP należy skompilować ponownie z wywołaniem funkcji systemowej `sleep(3)` — linia nr 35.

3. Korzystając z programu `wireshark`, podsłuchaj komunikację pomiędzy programem `nc(1)` a serwerem TCP i sprawdź wartości pól:

- *EtherType* w ramce Ethernet,
- Protokół w pakiecie IP,
- numer portu nadawcy i odbiorcy w segmencie TCP.



4. Korzystając z kodów w repozytorium GIT, uruchom serwer UDP (`udp-server.c`) i sprawdź dane systemowe o stworzonym gnieździe sieciowym:

- katalog `/proc/[pid]/fd/`,
- plik `/proc/[pid]/fdinfo/[fd-number]`,
- plik `/proc/net/udp`,
- polecenie `netstat`:

```
$ netstat -aupn
```

5. Sprawdź statystyki protokołów sieciowych używanych w systemie operacyjnym zawarte w pliku `/proc/net/protocols`:





```
$ cat /proc/net/protocols
```

- 2023-03-09  6. Napisz program serwera TCP, który działa zgodnie z następującym schematem:
- serwer, po akceptacji połączenia od nowego klienta, odbiera od niego ciąg bajtów i rozłącza się;
 - następnie, serwer nawiązuje połączenie z kolejnym serwerem — działającym pod adresem IP podanym jako pierwszy argument wywołania i numerem portu podanym jako drugi argument wywołania — oraz przesyła do niego ciąg bajtów odebrany od klienta i rozłącza się;
 - serwer powraca do akceptacji połączeń od kolejnych klientów.
- 2023-03-09  7. Napisz program serwera UDP, który działa zgodnie ze schematem przedstawionym w zadaniu nr 6.

2 Obsługa interfejsów sieciowych funkcją systemową ioctl(2)

1. Zbadaj wywołania systemowe komendy ifconfig(8):

```
$ strace ifconfig
$ strace -e open ifconfig
$ strace -e openat ifconfig
$ strace -e ioctl ifconfig
```



- 2023-03-16  2. Skompiluj i uruchom program o nazwie ifinfo.c, którego kod został podany w repozytorium GIT. Rozbuduj ten program tak, aby dodatkowo wyświetlane były adresy MAC interfejsów sieciowych (żądanie SIOCGIFHWADDR).
- 2023-03-16  3. Korzystając z kodów przedstawionych na wykładzie, napisz program, który włącza lub wyłącza interfejs sieciowy, którego nazwa podawana jest jako jego pierwszy argument wywołania. Drugim argumentem wywołania tego programu powinien być przełącznik włączania/wyłączania interfejsu.
- 2023-03-16  4. Korzystając z kodów przedstawionych na wykładzie, napisz program, który włącza lub wyłącza tryb nasłuchiwanie (ang. *promiscuous mode*) w interfejsie sieciowym, którego nazwa podawana jest jako jego pierwszy argument wywołania. Drugim argumentem wywołania tego programu powinien być przełącznik włączania/wyłączania trybu nasłuchiwanie.
- 2023-03-16  5. Skompiluj i uruchom program o nazwie ifsetup.c, którego kod został podany w repozytorium GIT. Rozbuduj ten program tak, aby dodatkowo konfigurował on maskę adresu IP (żądanie SIOCSIFNETMASK), która podawana będzie jako trzeci argument wywołania tego programu.
6. Skompiluj i uruchom program o nazwie ifaddrs.c, który demonstruje wykorzystanie funkcji systemowej getifaddrs(3). Kod tego programu znajduje się w repozytorium GIT.

3 Gniazda sieciowe PF_PACKET




1. Skompiluj programy znajdujące się w repozytorium GIT (ethrecv.c oraz ethsend.c). Na jednym komputerze uruchom program odbierający ramki sieci Ethernet II (ethrecv.c), a na drugim komputerze uruchom program transmitujący takie ramki (ethsend.c). Wysyłaj dane o różnym rozmiarze (mniejszym i większym niż 46B).
2. Zestaw połączenie bezprzewodowe w trybie *ad hoc* pomiędzy dwoma komputerami, np.:

```
# iwconfig wlan0 essid "test" mode Ad-Hoc
# ifconfig wlan0 10.0.0.1 netmask 255.0.0.0
```


Wykorzystując (domyślną) enkapsulację ethernetową na interfejsach bezprzewodowych, użyj programów z zadania nr 1 (ethrecv.c oraz ethsend.c) do komunikacji pomiędzy połączonymi komputerami (karta sieciowa wlan0). Wysyłaj dane o różnym rozmiarze (mniejszym i większym niż 46B) i porównaj odebrane ramki z ramkami odebranymi podczas realizacji zadania nr 1.

- 2023-03-23  3. Napisz program, który nasłuchuje ramki sieciowe ze wskazanego jako pierwszy argument wywołania interfejsu sieciowego i wyświetla następujące informacje o każdej odebranej ramce: jej rozmiar, adres MAC nadawcy, adres MAC odbiorcy, typ pakietu (wartość pola `sll_pkttype` struktury `sockaddr_ll`), wartość *EtherType* oraz transmitowane w niej dane.
- 2023-03-23  4. Napisz program, który działa zgodnie z następującą zasadą:
- program odbiera ramkę sieci Ethernet II z interfejsu sieciowego, którego nazwa podawana jest jako jego pierwszy argument wywołania;
 - program, odebrane z tej ramki dane, przesyła do kolejnego odbiorcy, korzystając z tego samego interfejsu sieciowego, pod adres MAC, który podawany jest jako jego drugi argument wywołania;
 - program powraca do odbierania kolejnej ramki.
- Jako wartości *EtherType* użyj liczby `0x8888`.

4 Filtracja ramek nasłuchiowanych (biblioteka `libpcap`)

1. Skompiluj i uruchom program filtrujący nasłuchiwane ramki sieciowe przy użyciu filtrów BPF, który znajduje się repozytorium GIT (`bpffilter.c`). Zmień używany w tym programie filtr `arp_filter` na filtr `icmp_filter` stałą `FILTER` i ponownie skompiluj i uruchom ten program.
2. Przy użyciu programu `tcpdump(1)` wygeneruj filtr BPF (w formacie kodu języka C — przełącznik `-dd`), który przechwytyje tylko zapytania transmitowane do systemu DNS.
- 2023-03-30  3. Zastosuj wygenerowany filtr w programie z zadania nr 3.3.
- 2023-03-30  4. Skompiluj i uruchom programy wykorzystujące bibliotekę `libpcap`, które znajdują się w repozytorium GIT (`pcapsniff.c` oraz `pcapfilter.c`). Program filtrujący nasłuchiwane ramki sieciowe (`pcapfilter.c`) uruchamiaj z różnymi filtrami zgodnie ze składnią opisaną w podręczniku `pcap-filter(7)`.
- 2023-03-30  5. Używając biblioteki `libpcap`, napisz program, który oblicza liczbę nasłuchiowanych ramek sieciowych z podziałem na następujące protokoły: ARP, IP, IP/UDP, IP/TCP oraz *inne*. Statystyki te powinny zostać wypisane przez realizowany program na standardowe wyjście po odebraniu sygnału `SIGINT`. Do analizy nagłówka ramki 802.3/Ethernet II wykorzystaj strukturę `ethhdr` zdefiniowaną w pliku nagłówkowym `if_ether.h`, a do analizy nagłówka protokołu IP wykorzystaj strukturę `iphdr` zdefiniowaną w pliku nagłówkowym `ip.h`.
 - 🔗 https://github.com/torvalds/linux/blob/master/include/uapi/linux/if_ether.h
 - 🔗 <https://github.com/torvalds/linux/blob/master/include/uapi/linux/ip.h>

5 Transmisja ramek z pakietami warstw wyższych (biblioteka `libnet`)

1. Skompiluj i uruchom program transmitujący żądania protokołu ARP, który znajduje się w repozytorium GIT (`arpreq.c`). Podczas działania programu, obserwuj transmitowane w sieci komputerowej ramki programem `Wireshark`. Zbadaj wpływ ustawień w plikach `/proc/sys/net/ipv4/conf/*/arp_accept` na sposób aktualizowania pamięci podręcznej przez protokół ARP.
2. Skompiluj i uruchom program transmitujący odpowiedzi protokołu ARP, który znajduje się w repozytorium GIT (`arprep.c`). Przejmij komunikację sieciową w sieci lokalnej wysyłając odpowiedź protokołu ARP z adresem IP bramy i własnym adresem MAC.
3. Zapoznaj się z działaniem programów `arping(8)` oraz `arp-scan(1)`.
- 2023-04-13  4. Korzystając z bibliotek `libpcap` oraz `libnet`, napisz program, który działa w sposób analogiczny do programu `arping(8)`. Do analizy nagłówka protokołu ARP wykorzystaj strukturę `arphdr` zdefiniowaną poniżej:

```

struct arphdr {
    u_int16_t ftype;
    u_int16_t ptype;
    u_int8_t  flen;
    u_int8_t  plen;
    u_int16_t opcode;
    u_int8_t  sender_mac_addr[6];
    u_int8_t  sender_ip_addr[4];
    u_int8_t  target_mac_addr[6];
    u_int8_t  target_ip_addr[4];
};

```

6 Obsługa tablicy routingu i pamięci podręcznej protokołu ARP

1. Zbadaj wywołania systemowe komend: arp(8), route(8) i ip(8):

```

$ strace -e open arp -n
$ strace -e openat arp -n
$ strace -e socket route -n
$ strace -e open route -n
$ strace -e openat route -n
$ strace -e socket ip route show
$ strace -e open ip route show
$ strace -e openat ip route show

```

UWAGA: polecenia systemowe arp(8) i route(8) powinny wykorzystywać funkcję systemową ioctl(2) i katalog /proc, a polecenie ip(8) powinno wykorzystywać gniazda sieciowe PF_NETLINK. W nowszych wersjach systemów operacyjnych GNU/Linux pakiet net-tools, zawierający implementacje arp(8) i route(8), może nie być zainstalowany, co oznacza, że polecenia te nie będą dostępne lub będą także korzystały z gniazd sieciowych PF_NETLINK. W razie potrzeby, proszę zatem o zainstalowanie pakietu net-tools.

2. Skompiluj i uruchom program odczytujący zawartość pamięci podręcznej protokołu ARP, który znajduje się w repozytorium GIT (arpget.c).
3. Skompiluj i uruchom program konfigurujący adres bramy domyślnej w tablicy routingu systemu, który znajduje się w repozytorium GIT (setgw.c).
4. Napisz program, który odbiera ramki sieci Ethernet II (z *EtherType* o wartości 0x8888) zawierające informacje o adresacji IP lub trasach routingu oraz konfiguruje, zgodnie z odebrany komunikatem, interfejs sieciowy lub tablicę routingu w systemie. Do konfiguracji interfejsu sieciowego wykorzystaj program ifsetup.c, który znajduje się w repozytorium GIT. Dane transmitowane ramkami sieci Ethernet II opisane są poniższą strukturą:

```

#define IRI_T_ADDRESS 0
#define IRI_T_ROUTE   1

struct ifrtinfo {
    int  iri_type;           /* msg type      */
    char iri_iname[16];      /* ifname        */
    struct sockaddr_in iri_iaddr; /* IP address    */
    struct sockaddr_in iri_rtdst; /* dst. IP address */
    struct sockaddr_in iri_rtmsk; /* dst. netmask   */
    struct sockaddr_in iri_rtgip; /* gateway IP     */
};

```

Podpowiedź: proszę zapoznać się z programem `irsend.c`, który znajduje się w repozytorium GIT. Program ten wysyła ramki, które mają być odbierane przez programy zrealizowane w ramach tego zadania.

7 Gniazda sieciowe PF_NETLINK

1. Skompiluj i uruchom program odczytujący zawartość tablicy routingu, który znajduje się w repozytorium GIT (`rtget.c`).

Zaobserwuj wywołania systemowe tego programu uruchamiając go z użyciem narzędzia `strace`:

```
$ strace ./rtget
```

Zmodyfikuj kod źródłowy tego programu tak, aby wyświetlał on (dowolne) komunikaty w konsoli dla wszystkich przetwarzanych przez niego struktur w odebranej wiadomości od jądra systemu operacyjnego.

Zmodyfikuj kod źródłowy tego programu tak, aby wyświetlał on zawartość wszystkich tablic routingu dostępnych w systemie operacyjnym.

2. Skompiluj i uruchom program dodający trasę do tablicy routing, który znajduje się w repozytorium GIT (`rtset.c`).

UWAGA: w kodzie źródłowym tego programu należy ustawić wartość stałej `DEV_NUMBER` na numer indeksu wyjściowego interfejsu sieciowego (numery te, dla kart sieciowych, można uzyskać np. poleceniem systemowym `ip(8)`).

3. Skompiluj i uruchom program monitorujący zmiany w zawartości tablicy routingu, który znajduje się w repozytorium GIT (`rtmon.c`). Po uruchomieniu tego programu, obserwuj wyświetlane przez niego komunikaty podczas dodawania i usuwania wpisów do tablicy routingu.



4. Napisz program, który, korzystając z gniazd sieciowych PF_NETLINK, usunie wskazany wpis z tablicy routingu.

2023-04-27

8 Nieprzetworzone gniazda sieciowe

1. Skompiluj programy znajdujące się w repozytorium GIT (`iprecv.c` oraz `ipsend.c`). Na jednym komputerze uruchom program odbierający pakiety IP (`iprecv.c`), a na drugim komputerze uruchom program transmitujący takie pakiety (`ipsend.c`). Wysyłaj dane o różnym rozmiarze.
2. Skompiluj i uruchom program transmitujący żądania protokołu ICMP, który znajduje się w repozytorium GIT (`ipping.c`).



3. Zmodyfikuj kod źródłowy programu z zadania nr 8.2 tak, aby dodatkowo weryfikować pole `type` w nagłówkach odbieranych pakietów protokołu ICMP (wartość tego pola w pakietach odbieranych powinna być równa stałej `ICMP_ECHOREPLY`).

2023-05-04



4. Napisz program, który działa zgodnie z następującą zasadą:

2023-05-04

- program odbiera pakiet IP;
- program, odebrane z tego pakietu dane, przesyła do kolejnego odbiorcy pod adres IP, który podawany jest jako jego pierwszy argument wywołania;
- program powraca do odbierania kolejnego pakietu.

Jako wartości `IPPROTO` użyj liczby 222.

9 Internet Protocol version 6 (IPv6)

1. Sprawdź aktualną konfigurację protokołu IPv6 na swoim komputerze:

```
$ ip -6 addr show
$ ip -6 route show
$ ip -6 neigh show
```

Porównaj wynik ostatniego polecenia z zawartością pamięci podręcznej protokołu ARP.

2. Sprawdź komunikację internetową przy zastosowaniu protokołu IPv6:

```
$ ping6 google.com
$ traceroute6 google.com
$ mtr -6 google.com
```

3. Skompiluj i uruchom programy serwera (ipv6-tcp-server.c) oraz klienta (ipv6-tcp-client.c) wykorzystujące protokół IPv6, które znajdują się w repozytorium GIT.

Zmodyfikuj kod serwera w taki sposób, aby wyświetlał on adres IP klienta, który nawiązał połączenie (użyj funkcji systemowej `inet_ntop(3)`).

Nawiąż połączenie do uruchomionego procesu serwera korzystając z programu telnet, który wykorzystuje protokół IPv4.



4. Korzystając z poniższej funkcji, napisz program klienta protokołu TCP, który jest niezależny od protokołu warstwy sieciowej.

```
int _connect(const char *host, const char *service) {
    int sfd;
    struct addrinfo hints, *res, *ressave;
    memset(&hints, 0, sizeof(hints));
    hints.ai_family = AF_UNSPEC;
    hints.ai_socktype = SOCK_STREAM;
    getaddrinfo(host, service, &hints, &res);
    ressave = res;
    do {
        sfd = socket(res->ai_family, res->ai_socktype,
                    res->ai_protocol);
        if (connect(sfd, res->ai_addr, res->ai_addrlen) == 0)
            break;
        close(sfd);
        sfd = -1;
    } while((res = res->ai_next) != NULL);
    freeaddrinfo(ressave);
    return sfd;
}
```

2023-05-11

10 Stream Control Transmission Protocol (SCTP)

1. Wykorzystując programy `sctp_test(1)` oraz `sctp_darn(1)` zestaw połączenie pomiędzy dwoma procesami.

```
$ sctp_darn -H 0 -P 2500 -l
...
$ sctp_darn -H 0 -P 2600 -h 127.0.0.1 -p 2500 -s
...
```

```
$ sctp_test -H 0 -P 2500 -l
...
$ sctp_test -H 0 -P 2600 -h 127.0.0.1 -p 2500 -s
...
```

2. Skompiluj i uruchom program wykorzystujący połączenia wieloma łączami z zastosowaniem funkcji `sctp_bindx(3)` (*demonstracja selekcji łącz*), który znajduje się w repozytorium GIT (`sctpmh.c`).

3. Skompiluj i uruchom programy serwera (`sctpms-server.c`) oraz klienta (`sctpms-client.c`) protokołu SCTP wykorzystujące połączenia wieloma strumieniami, które znajdują się w repozytorium GIT.

4. Skompiluj i uruchom program serwera (`sctpmh-server.c`) oraz klienta (`sctpmh-client.c`) protokołu SCTP wykorzystujące połączenia wieloma łączami, które znajdują się w repozytorium GIT.

W trakcie połączenia pomiędzy procesami klienta i serwera zasymuluj awarię jednego z interfejsów sieciowych.



5. Zmodyfikuj programy z zadania nr 10.4 tak, aby dodatkowo wykorzystywały połączenie wieloma strumieniami analogicznie jak w zadaniu nr 10.3.

Obserwuj komunikację pomiędzy procesami klienta i serwera z użyciem programu wireshark.

2023-05-18

11 Buforowanie w komunikacji sieciowej

1. Zweryfikuj rozmiary buforów nadawczych i odbiorczych gniazd sieciowych protokołów TCP i UDP zapisane w plikach poniższych:

- `/proc/sys/net/ipv4/tcp_wmem,`
- `/proc/sys/net/ipv4/tcp_rmem,`
- `/proc/sys/net/ipv4/udp_mem,`
- `/proc/sys/net/ipv4/udp_wmem_min,`
- `/proc/sys/net/ipv4/udp_rmem_min.`

2. Skompiluj i uruchom program (`socket-buffers.c`) pobierający rozmiary buforów gniazd sieciowych protokołów TCP i UDP, który znajduje się w repozytorium GIT.

Porównaj otrzymane wyniki z zawartością plików z zadania nr 1.

3. Manipulując zawartością plików z zadania nr 1 oraz korzystając z programów klienta i serwera protokołu TCP z zajęć nr 1, doprowadź do następujących dwóch sytuacji:

- a) pojedyncze wywołanie funkcji systemowej `read(2)` nie odbierze wszystkich wysłanych danych;
- b) pojedyncze wywołanie funkcji systemowej `read(2)` odbierze dane wysłane dwoma wywołaniami funkcji systemowej `write(2)`.



4. Wykorzystaj przedstawione na wykładzie funkcje `_read()` i `_write()` tak, aby poprawić działanie programów klienta i serwera protokołu TCP z zajęć nr 1 w odniesieniu do rozmiarów systemowych buforów nadawczych i odbiorczych gniazd sieciowych TCP. *Zastosuj podejście ze stałym rozmiarem danych.*

2023-06-01



5. Korzystając z kodów przedstawionych na wykładzie zmień rozmiar buforów nadawczych i odbiorczych gniazd sieciowych protokołu TCP przy użyciu funkcji `setsockopt(2)`.

2023-06-01

12 Obsługa operacji wejścia/wyjścia komunikacji sieciowej

Uruchom program serwera protokołu TCP (`server.c`), który znajduje się w repozytorium GIT, a następnie:

1. Skompiluj i uruchom programy nakładające własne limity czasowe na operację odczytu danych: `01-read-alarm.c`, `02-read-select.c`, `03-read-rcvtimeo.c`.
2. Skompiluj i uruchom programy ilustrujące modele obsługi operacji wejścia/wyjścia komunikacji sieciowej: `04-read-nonblock.c`, `05-read-sigio.c`, `06-read-aio.c`.
3. Zmodyfikuj program asynchronicznej obsługi operacji odczytu danych (`06-read-aio.c`) tak, aby po zakończeniu tej operacji wykonywana była wskazana funkcja programu (`SIGEV_THREAD`).

13 Architektury serwerów sieciowych

1. Skompiluj i uruchom programy ilustrujące podstawowe architektury serwerów sieciowych: `tcp-server-fork.c`, `tcp-server-thread.c` oraz `tcp-server-select.c`, które znajdują się w repozytorium GIT.
2. Zmodyfikuj kod serwera `tcp-server-select.c` tak, aby wykorzystywał on mechanizm `epoll(7)` w miejsce funkcji systemowej `select(2)`.

PROJEKTY ZALICZENIOWE

Kody źródłowe realizowanych projektów zaliczeniowych należy przechowywać we własnym repozytorium GIT, w katalogu o nazwie `projekt-zaliczeniowy`. Do każdego projektu należy dodać plik o nazwie `README.md`, w którym należy umieścić następujące informacje:

- nazwa i opis projektu (nie więcej niż kilka zdań);
- zawartość plików źródłowych i zasadę ich kompilacji;
- sposób uruchomienia programu/programów projektu.

Tematy projektów zaliczeniowych

1. Wersja rozproszona narzędzia `arp-scan(1)`.
2. Wersja rozproszona narzędzia `arpwatch(8)`.
3. Analizator komunikacji sieci bezprzewodowych IEEE 802.11/Wi-Fi.
4. Narzędzie sterujące mocą sygnału karty bezprzewodowej.
5. Gra dla mobilnych bezprzewodowych sieci ad-hoc.
6. System monitorowania parametrów sieci komputerowej w systemie operacyjnym GNU/Linux z użyciem gniazd sieciowych `PF_NETLINK`.
7. System ataku typu *DNS spoofing*.
8. Serwer usługi DHCP.
9. System kopiowania wiadomości e-mail pomiędzy serwerami poczty elektronicznej z użyciem protokołu IMAP.
10. Protokół routingu dynamicznego (np. RIP lub rozwiązanie własne).

Kryteria oceny projektów

- Poprawność implementacji oraz czytelność kodu źródłowego.
- Zgodność funkcjonalności projektu z uzgodnionymi wymaganiami.
- Inicjatywa i pomysłowość w realizacji projektu.

Termin zaliczania projektów

Ostatecznym terminem zaliczania projektów są ostatnie zajęcia laboratoryjne w semestrze.