

# Programowanie sieciowe

## Instrukcja do laboratorium LAB03

### Porządek bajtów w programach sieciowych

**Zadanie 1.** Przeanalizować i uruchomić program `byteorder.c` - program podaje jaka jest kolejność bajtów na komputerze PC. Wykorzystując ten program sprawdzić doświadczalnie jaka jest sieciowa kolejność bajtów - z użyciem funkcji `htons()` zmodyfikować program tak, aby dodatkowo sprawdzał i wyświetlał jaka jest sieciowa kolejność bajtów.

**Zadanie 2.** Kody źródłowe `tcpcli_1.c` i `tcpserv_1.c` implementują klienta i serwer usługi echa (**port 7**) (to co klient pobierze z klawiatury i wyśle, zostaje przez serwer odesłane do klienta i wyświetlone na ekranie), ale zawierają **jeden błąd** (tylko w jednym z programów), który uniemożliwia nawiązanie połączenia dla usługi echa. Proszę uruchomić programy, przeglądnąć kod, znaleźć ten błąd i go naprawić w kodzie źródłowym - do znalezienia błędu wykorzystać narzędzia `netstat/lsof/ss` i `tcpdump/wireshark`.

### Sygnały

**Zadanie 3. Oczekiwanie za zakończenie potomka.** Przeanalizować, skompilować i uruchomić program `test_signal.c` według poniższej kolejności:

```
$ ./test_signal&
Child PID is 32360
[1] 32359
$ kill -STOP 32360
stopped by signal 19
$ kill -CONT 32360
continued
$ kill -TERM 32360
killed by signal 15
[1]+  Done                  ./test_signal
$
```

Sprawdzić zachowanie programu w zależności od ustawień flag **WUNTRACED** i **WCONTINUED** w funkcji `waitpid()`.

**Zadanie 4. Sygnał SIGCHLD:** Uruchomić komunikację pomiędzy programami `echo_tcpservv6.c` i `echo_tcpcliv6.c` - przykład usługi ECHO (port 7), w której serwer odsyła do klienta wszystko to, co od niego otrzyma. **Połączyć się kilkakrotnie z serwerem** i sprawdzić ile procesów jest tworzonych przez serwer, i czy po zamknięciu klientów zostają procesy "zombie" (`sudo ps aux | grep 'nazwa programu serwera'`). Zaobserwować w programie `echo_tcpservv6.c`, jaka jest różnica, gdy w programie serwera współbieżnego z użyciem funkcji `fork()` zignorujemy sygnał **SIGCHLD** jawnie (wywołamy funkcję

signal() z parametrem SIG\_IGN - odkomentowana linia 108) i niejawnie (nie zrobimy nic - domyślną reakcją jest ignorowanie sygnału **SIGCHLD**), a co się stanie gdy odkomentujemy linię 91 przy zakomentowanych liniach 107 i 108.

**Sygnał SIGPIPE w programach używających gniazd:** jeśli w procesie funkcja pisząca odsyła dane do gniazda, które otrzymało segment RESET, to proces otrzymuje sygnał SIGPIPE. Domyślną reakcją na otrzymanie sygnału SIGPIPE jest zamknięcie procesu.

**Zadanie 5. Sygnał SIGPIPE i serwer iteracyjny:** Programy daytimetcpsrvv6\_04.c i daytimetcpcliv6\_04.c przesyłają dodatkowy strumień bajtów od serwera do klienta. Skompilować przykłady i uruchomić komunikację pomiędzy dwoma komputerami. Proszę zauważyć, że jeśli klient zostanie przerwany w trakcie działania (np. wciskając kombinację klawiszy CTRL-C) to serwer kończy działanie i nie przechodzi przez stan TIME\_WAIT. **Dlaczego występuje takie zjawisko? W prawidłowo zaprojektowanej aplikacji serwera nie można dopuścić do takiej sytuacji.** Zaobserwować w programie tcpdump/wireshark co dzieje się z połączeniem. Jaki segment TCP powoduje zerwanie połączenia? Dopisać odpowiedni kod w programie serwera, który będzie zabezpieczał przed niekontrolowanym zakończeniem serwera i wyświetlał komunikat o takim zdarzeniu.

**Rozwiązanie:** Obsłużyć sygnał **SIGPIPE** w serwerze tak, aby wypisywał komunikat na ekranie o powstałym problemie, zamykał gniazdo i przechodził do obsługi nowego połączenia.

**Zadanie 6. Sygnał SIGPIPE i serwer współbieżny:** Programy echo\_tcpsrvv6\_del.c i echo\_tcpcliv6.c implementują serwer i klienta usługi echa dla IPv6 (to co klient pobierze z klawiatury i wyśle do serwera, zostaje przez serwer odesłane do klienta z opóźnieniem dwóch sekund i wyświetlone na ekranie). Skompilować przykłady i uruchomić komunikację pomiędzy dwoma komputerami. Proszę zauważyć, że jeśli klient w trakcie działania, zaraz po wysłaniu danych, zostanie przerwany (np. kombinacją klawiszy **CTRL-C**), to proces obsługujący klienta (proces serwera, który obsługuje połączenie na gnieździe połączonym) kończy działanie. **Dlaczego występuje takie zjawisko?** Dopisać odpowiedni kod w programie serwera, który będzie zabezpieczał przed niekontrolowanym zakończeniem procesu obsługującego klienta i wyświetlał komunikat o takim zdarzeniu. **Dlaczego cały serwer nie kończy działania, jak w przypadku serwera iteracyjnego?**

**Rozwiązanie:** Obsłużyć sygnał **SIGPIPE** w serwerze tak, aby wypisywał komunikat na ekranie o powstałym problemie, zamykał gniazdo i zamykał poprawnie proces.

Sygnał **SIGURG** przesyłany jest do odbiorcy danych pozapasmowych. Domyślną reakcją procesu na otrzymanie sygnału SIGURG jest jego **zignorowanie**. Aby otrzymać sygnał SIGURG w procesie należy **ustanowić procedurę obsługi sygnału i ustawić właściciela gniazda**. Gniazda połączone nie dziedziczą właściciela gniazda z gniazda nasłuchującego.

**Zadanie 7. Sygnał SIGURG:** Skompilować przykłady tcprecv01.c i tcpsend01.c. Uruchomić program tcprecv01 na dowolnym porcie.

a) Połączyć się z serwerem za pomocą programu tcpsend01 podglądając jednocześnie wymianę komunikatów programem tcpdump/wireshark.

Procedurę z punktu a) powtórzyć dla:

b) zakomentowanych lini 17, 25, 29 z funkcją sleep() w programie tcpsend01.c

c) dodatkowo odkomentowanych linii 19 i 20

d) dodatkowo zakomentowanymi liniami nr 30, 31 i 32 w programie tcprecv01.c, które odpowiadają za ustawienie właściciela gniazda (linie 28 i 29 realizują tę samą funkcjonalność za pomocą funkcji fcntl())

e) sprawdzić jaka jest domyślna reakcja procesu na odebranie sygnału SIGURG

f) **Dla chętnych:** z ustawioną opcją gniazda **SO\_OOBINLINE** w programie tcprecv01.c

#### **Odpowiedzieć na następujące pytania:**

- Dlaczego kolejność otrzymywanych danych różni się dla punktu a) i b)
- Dlaczego dla punktu d) nie odbieramy danych pozapasmowych, pomimo, że są wysyłane przez program tcpsend01
- Ile bajtów danych pozapasmowych można przesłać w jednym wywołaniu funkcji send()?
- Dlaczego w podpunkcie c) pojawia się błąd, jeśli wyślemy bez opóźnień dwa razy po sobie dane pozapasmowe?

#### **Do przygotowania na następne zajęcia (LAB04):**

1. Wiadomości z LAB01, LAB02, LAB03.

2. Wiadomości z wykładów 2, 3, 4.

Odpowiedzieć na pytania:

- a) Jaka jest sieciowa kolejność bajtów?
- b) Kiedy do procesu jest wysyłany sygnał SIGCHLD?
- c) Kiedy do procesu jest wysyłany sygnał SIGPIPE?
- d) W jaki sposób wysłać segment TCP RST?
- e) Do czego służy funkcja waitpid()?
- f) Do czego służy funkcja sigaction()?
- g) Do czego służy flaga SA\_RESTART?
- h) Kiedy do procesu jest wysyłany sygnał SIGURG?
- i) W jaki sposób zakończyć natychmiast połączenie TCP (nie zresetować)?
- j) Kiedy wywołanie funkcji close() na gnieździe połączonym spowoduje wysłanie segmentu FIN dla połączenia TCP, a kiedy nie?
- k) Jak w serwerze UDP i TCP pobrać informacje o parametrach połączenia?
- l) Kiedy dla gniazda blokującego TCP funkcja czytająca z gniazda zwraca wartość 0?
- m) Kiedy dla gniazda blokującego TCP funkcja czytająca z gniazda zwraca wartość -1?

- n) Kiedy dla gniazda blokującego TCP funkcja czytająca z gniazda zwraca wartość większą od zera?
- o) Czym różni się funkcja write() od send() i sendto?
- p) Czym różni się funkcja read() od recv() i recvfrom?
- q) Do czego służą i jakie parametry posiadają funkcje: socket(), connect(), bind(), listen(), accept(), close(), shutdown(), getsockname(), getpeername()?

Po wykładzie #4:

- r) Kiedy dla gniazda blokującego UDP funkcja czytająca z gniazda zwraca wartość 0?
- s) Kiedy dla gniazda blokującego UDP funkcja czytająca z gniazda zwraca wartość -1?
- t) Kiedy dla gniazda blokującego UDP funkcja czytająca z gniazda zwraca wartość większą od zera?
- u) Kiedy dla gniazda nieblokującego funkcja czytająca z gniazda zwraca wartość -1?
- v) Co to są funkcje reentrant?
- w) Czym wyróżnia się funkcja rcvfrom()?
- x) Jakie flagi obsługują funkcje recv() i recvfrom()? Czym różni się od funkcji read()?
- y) Jakie flagi obsługują funkcje send() i sendto()? Czym różni się od funkcji write()?
- z) Do czego używa się funkcji connect() dla protokołu UDP?
- aa) Kiedy na gnieździe UDP możemy odbierać błędy asynchroniczne protokołu ICMP?