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 – obsługa sygnału SIGCHILD w procesie macierzystym. 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 SIGCHILD: 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 kilkukrotnie 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 **SIGCHILD** jawnie (wywołamy funkcję signal() z parametrem SIG_IGN - odkomentowana linia 108) i niejawnie (nie zrobimy nic - domyślną reakcją jest ignorowanie sygnału **SIGCHILD), 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_tcpservv6_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. Jeśli proces zakończy się prawidłowo to wypisze "Tutaj sprzatam po procesie", natomiast gdy skończy się w nieprzewidywalnym momencie, ten napis nie będzie się wypisywał. Prawidłowe kończenie klienta wykonujemy sekwencją klawiszy CTRL-D. 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. Prawidłowe wykonanie tego ćwiczenia polega na tym, żeby zawsze wypisywał się napis "Tutaj sprzatam po procesie", niezależnie jak zostanie zamknięty klient. 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 tą samą funkcjonalność za pomocą funkcji fcntl())

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

f) **Dla chętnych:** z ustawioną opcją gniazd **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 SIGCHILD?
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) Jakie kroki należy wykonać w procesie, aby w procesie można było odebrać i obsłużyć sygnał SIGURG?

j) W jaki sposób zakończyć natychmiast połączenie TCP (nie zresetować)?
k) Kiedy wywołanie funkcji close() na gnieździe połączonym spowoduje wysłanie segmentu FIN dla połączenia TCP, a kiedy nie?
l) Jak w serwerze UDP i TCP pobrać informacje o parametrach połączenia?
m) Kiedy dla gniazda blokującego TCP funkcja czytająca z gniazda zwraca wartość 0?
n) Kiedy dla gniazda blokującego TCP funkcja czytająca z gniazda zwraca wartość -1?
o) Kiedy dla gniazda blokującego TCP funkcja czytająca z gniazda zwraca wartość większą od zera?
p) Czym różnią się funkcja write() od send() i sendto?
q) Czym różni się funkcja read() od recv() i recvfrom?
r) Do czego służą i jakie parametry posiadają funkcje: socket(), connect(), bind(), listen(), accept(), close(), shutdown(), getsockname(), getpeername()?

Po wykładzie #4:

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