Programowanie sieciowe
Instrukcja do laboratorium LAB 07

Ćwiczenia wykonujemy w grupach dwuosobowych pomiędzy komputerami lub jeśli to niemożliwe za pomocą interfejsu loopback.

Zadanie 1. Różnice mechanizmu funkcji fork() i select() dla serwerów współbieżnych

1. W terminalu przejść do katalogu CW1:

2. Skompilować przykłady: tcpserv6_ws_echo_select.c i tcpcliv6_select_echo.c - przykład serwera i klienta wykorzystujący funkcję select(). **Uwaga**: przy kompilacji programu tcpcliv6_select_echo.c należy dołączyć bibliotekę math (-lm), np. :

```
gcc tcpcliv6_select_echo.c  -lm  -o cli
```

3. Otworzyć cztery terminale: dla programów: serwera, dwóch programów klienta i programu tcpdump

4. Uruchomić program tcpdump:

```
tcpdump -i lo -e -v  port 7
```

lub

```
tcpdump -i eth1 -e -v  port 7
```

w zależności czy badamy ruch pomiędzy klientem i serwerem uruchomionymi na tym samym komputerze, czy pomiędzy różnymi komputerami (niezależnie od adresów, które są używane).

5. W jednym z terminali uruchomić serwer, w drugim i trzecim połączyć się z serwerem za pomocą klienta, przesłać dane. Zaobserwować wymianę komunikatów w programie tcpdump.

6. W nowym terminalu uruchomić program netstat:

netstat --inet6 -tapn

Zidentyfikować gniazda należące do programu serwera i klienta.

7. Skompilować przykłady: tcpserv6_ws_echo_fork.c i tcpcliv6_echo.c. Powtórzyć punkty 4, 5 i 6 dla serwera z funkcją fork() .

Jakie różnice można zaobserwować porównując serwer współbieżny z funkcją fork() i select()?

Zadanie 2. Jedną z wad serwera z przykładu tcpserv6_ws_echo_select.c jest brak odporności na wymuszenie odmowy obsługi - typowy atak **DoS** (*Denial of Service*). Rozważmy, co się stanie wtedy, gdy jakiś złośliwy klient zechce się połączyć z naszym serwerem, wysłać dane nie zakończone znakiem nowego wiersza, po czym postanowił usunąć. Program serwera wywoła funkcję

`Readline()`, która odbierze od klienta wysłane bajty i zablokuje się po następnym wywołaniu funkcji systemowej `read`, oczekując na wysłanie przez klienta reszty danych ze znakiem końca linii. A zatem program serwera blokuje się i nie może obsługiwać pozostałych klientów, ani nowych połączeń, ponieważ jest zaimplementowany w jednym procesie za pomocą funkcji `select()`. Serwer zostanie odblokowany, jeśli klient odeśle znak końca linii lub zamknie połączenie.

a) przerobić klienta do ataku DoS w ten sposób, aby wysyłał dane, ale nie wysyłał znaku końca linii (nie wysyłać ostatniego znaku, który jest odczytywany z klawiatury) (serwer czyta całe linie funkcja `Readline()`, która będzie blokować serwer, dopóki nie otrzyma znaku końca linii lub klient zamknie połączenie, czyli funkcja `Readline()` zwróci 0).

b) Sprawdzić, czy po ataku DoS można się połączyć z serwerem za pomocą drugiego klienta.

c) uodpornić serwer na atak DoS - zastosować czas oczekiwania na czytanie dla gniazda np. opcją `SO_RCVTIMEO` i ew. zresetować połączenie dla klienta, który blokuje gniazdo.

(kod do ustawiania opcji `SO_RCVTIMEO` można skopiować z wcześniejszych przykładów)

Wniosek: Dla serwerów, które w jednym procesie obsługują wielu klientów, należy zwrócić uwagę na możliwość blokowania funkcji systemowych w nieoczekiwanym momencie.

Dodatkowo: odrzucać przez pewien czas połączenia od źle zachowujących się klientów

Zmienić katalog na CW2.

Zadanie 3: Porównanie mechanizmów POLL i EPOLL - nieobowiązkowe
Przykład echo_serv6_ws_poll.c jest implementacją serwera współbieżnego usługi echo z mechanizmem poll. Zapoznać się z kodem programu.

1. Skompilować przykład echo_serv6_ws_ poll.c. Ustawić stałą MAXEVENTS na 1000000, jeśli ma inną wartość.

gcc echo_serv6_ws_poll.c -o serv -lpthread

2. W innym terminalu przejść do katalogu CW2/client i skompilować program klienta komendą make.

3. Uruchomić program klienta bez parametrów i zapoznać się z parametrami wywołania programu. Następnie uruchomić klienta z następującymi parametrami:

```
./tcploadechoclient -D adres_serwera -p 7 -n 1000 -r 100000 -c 200 -s
```

4. Zaobserwować liczbę otwartych gniazd po stronie serwera i klienta, oraz czas obsługi zdarzeń wyświetlany przez program serwera. Zbadać, jak zmienia się czas obsługi w zależności od parametru -r programu `tcploadechoclient` i wartości stałej MAXEVENTS w kodzie programu echo_serv6_ws_poll.c

Zamknąć komunikację.

5. Przeanalizować przykład echo_serv6_ws_epoll.c, który jest implementacją serwera współbieżnego usługi echo z mechanizmem epoll i go skompilować.

Uruchomić serwer z uprawnieniami użytkownika root.

6. Powtórzyć punkty 3 i 4 dla serwera z mechanizmem EPOLL. Czy czasy obsługi różnią się dla POLL i EPOLL - dlaczego?

Zadanie 4: Obsługa dużej liczby klientów w serwerze

1. Spróbować uruchomić komunikację jak w zadaniu 3 dla serwerów: echo_serv6_ws_epoll.c, echo_serv6_ws_poll.c i tcpserv6_ws_echo_fork.c dla liczby połączeń równej wartości komendy 'ulimit -n'. Ile udaje się nawiązać połączeń? Dlaczego?

2. Korzystając z konfiguracji jądra w systemie plików /proc i polecenia ulimit skonfigurować system w taki sposób, aby w serwerze dało się obsłużyć równolegle 100 000 połączeń klientów. Wykonać testy obciążające poszczególne serwery przez 100000 równoległych połączeń (do obciążania użyć kilku klientów na różnych komputerach z Zadania 3). Ile udaje się nawiązać połączeń dla poszczególnych serwerów?

Uwzględnić następujące parametry systemu operacyjnego Linux:

1. Maksymalną liczbę otwartych plików w systemie: /proc/sys/fs/file-max
   Podgląd poleceniem sysctl:
   ```
   sysctl fs.file-max
   ```
   Zmiana poleceniem sysctl, np. na wartość 120000:
   ```
   sysctl -w fs.file-max=120000
   ```
2. Maksymalny rozmiar tablicy deskryptorów dla mechanizmu epoll:
`/proc/sys/fs/epoll/max_user_watches`
3. Maksymalną liczbę otwartych plików w pojedynczym procesie:
   Podgląd:
   ```
   ulimit -n
   ```
   Zmiana, np. na 100005:
   ```
   ulimit –n 100005
   ```
4. Szybkość tworzenia nowych połączeń TCP:

$$szybkość = \frac{(net.ipv4.ip\_local\_port\_range)}{(net.ipv4.tcp\_fin\_timeout)}$$

5. Liczbę portów dostępną dla klienta: parametr net.ipv4.ip_local_port_range (/proc/sys/net/ipv4/ip_local_port_range)

6. Parametr nf_conntrack_max (find /proc -name '*conntrack_max*')

Zadanie 5: Tryby 'level-triggered' i 'edge-trigered'

Zmienić katalog na CW3.

1. Skompilować przykłady echo_serv6_ws_epoll.c i tcpcliv6_select_echo.c. Uruchomić komunikację pomiędzy serwerem i klientem. Przesłać ciągi znaków o różnej długości - poniżej i powyżej 10 znaków (bufor odbiorczy został ograniczony do 10 znaków).

2. Przykład echo_serv6_ws_epoll.c implementuje tryb 'level-triggered'. Zmienić na tryb 'edge-trigered'. W tym celu dodać flagę EPOLLET dla gniazda połączonego w strukturze ev przekazanej do funkcji epoll_ctl() (zmiana w linii 236). Skompilować przykład.

3. Powtórzyć punkt pierwszy dla zmodyfikowanego przykładu z punktu 2. Jaka jest różnica?

4. Zmodyfikować przykład zmieniony w punkcie 2 tak, aby działał poprawnie, tzn. opróżniał bufor odbiorczy dla każdego powiadomienia o możliwości czytania gniazda. W tym celu:

- gniazdo połączone zamienić na nieblokujące - użyć funkcji accept4() z flagą SOCK_NONBLOCK
- powtarzać procedurę czytania i wysyłania z gniazda np. w pętli while(), dopóki bufor odczytujący nie zostanie opróżniony, czyli dopóki nie pojawi się błąd EAGAIN lub EWOULDBLOCK. Dla gniazda nieblokującego, błędy EAGAIN i EWOULDBLOCK przy odczycie oznaczają, że bufor z którego czytamy jest pusty.

5. Sprawdzić poprawność implementacji mechanizmów z punktu 4.

UWAGA: dla mechanizmu EPOLL/POLL należy używać gniazd nieblokujących (także nasłuchujących), tak, aby blokowanie następowało tylko dla funkcji epoll_wait().

Pytania sprawdzające:

1. Jakie modele obsługi można stosować dla gniazd sieciowych?

2. Kiedy gniazdo jest gotowe do czytania dla funkcji z rodziny select()/poll()?

3. Kiedy gniazdo jest gotowe do pisania dla funkcji z rodziny select()/poll()?

4. Co zwraca funkcja select()?

6. Co zwraca funkcja poll()?

7. Co zwraca funkcja epoll_wait()?

8. Jakie różnice można zaobserwować w działaniu serwera współbieżnego z użyciem funkcji z rodziny select() i serwera współbieżnego z użyciem funkcji fork(), przy użyciu poleceń netstat (ss) i tcpdump (wireshark)?

9. Czym różni się funkcja select() od pselect()?

10. Czym różni się funkcja poll() od ppoll()?

11. Czym różni się tryb 'level-triggered' i 'edge-trigered' dla mechanizmu epoll?

12. Dlaczego należy stosować gniazda nieblokujące dla funkcji typu select()/poll() i mechanizmu epoll?

13. Dlaczego musimy stosować gniazda nieblokujące dla trybu 'edge-trigered' dla mechanizmu epoll?

14. Jak działa mechanizm epoll - czym różni się od użycia funkcji poll()?