Iteracyjny serwer TCP i aplikacja UDP

Iteracyjny serwer TCP

Funkcje wywoływane przez serwer TCP

socket() - bind() - listen() - accept() - read() / write() - close()

socket()

Creates an endpoint for communication and returns a **file descriptor** that refers to that endpoint.

int socket(int domain, int type, int protocol);
domain e.g. AF_INET, AF_INET6, AF_UNIX
type e.g. SOCK_STREAM, SOCK_DGRAM
protocol e.g. 0 //wybierany automatycznie

Zwraca: deskryptor utworzonego gniazda lub -1 (błąd, ustawia errno).

bind()

assigns the address specified by addr to the socket referred to by the file descriptor sockfd

int bind(int sockfd, const struct sockaddr *addr, socklen_t addrlen);
sockfd - deskryptor gniazda
addr - adres, który zostanie przypisany gniazdu
addrlen - rozmiar struktury przekazanej jako addr

Zwraca: 0 (ok) lub -1 (błąd, ustawia errno).

bind()

W praktyce:

```
sa.sin_port = htons([port aplikacji]);
sa.sin_addr.s_addr = htonl(INADDR_ANY);

bind(sockfd, (struct sockaddr *) &sa, sizeof(sa));
```

listen()

marks the socket [...] as a passive socket, that is, as a socket that will be used to accept incoming connection requests using accept.

int listen(int sockfd, int backlog);

sockfd - deskryptor gniazda

backlog - rozmiar kolejki, w której buforowane są żądania połączeń. Jeżeli żądanie pojawi się gdy kolejka jest pełna, nie dotrze do aplikacji. Ten parametr jest jedynie wskazówką (ang. hint), implementacja może ją zignorować.

Zwraca: 0 (ok) lub -1 (błąd, ustawia errno)

(dla zainteresowanych) o działaniu backlog w systemie Linux: http://veithen.github.io/2014/01/01/how-tcp-backlog-works-in-linux.html

accept()

extracts the first connection request on the queue of pending connections for the listening socket, sockfd, creates a new connected socket, and returns a new file descriptor referring to that socket.

int accept(int sockfd, struct sockaddr *addr, socklen_t *addrlen);
sockfd - deskryptor gniazda serwera
addr - struktura, w której zostaną umieszczone dane klienta
addrlen - rozmiar struktury przekazanej jako addr

Zwraca: deskryptor połączonego klienta lub -1 (błąd, ustawia errno).

accept()

W praktyce:

```
struct sockaddr_in sa;

...

socklen_t sa_size = sizeof(sa);

accept(sockfd, (struct sockaddr *) &sa, &sa_size);
```

read()

attempts to read **up to** count bytes from file descriptor fd into the buffer starting at buf.

ssize_t read(int fd, void *buf, size_t count);
fd - deskryptor gniazda, które będzie oczekiwało na dane
buf - bufor, do którego zostaną przekazane odebrane dane
count - maksymalna ilość danych, która będzie przekazana do bufora

Zwraca: liczbę odebranych bajtów lub -1 (błąd, ustawia errno) lub 0 (gniazdo zamknięte).

write()

writes up to count bytes from the buffer pointed buf to the file referred to by the file descriptor fd

ssize_t write(int fd, const void *buf, size_t count);
fd - deskryptor gniazda, które będzie wysyłało na dane
buf - bufor, z którego zostaną odczytane dane do wysłania
count - maksymalna ilość danych, która będzie odczytana z bufora

Zwraca: liczbę wysłanych bajtów lub -1 (błąd, ustawia errno).

close()

closes a file descriptor, so that it no longer refers to any file and may be reused.

int close(int fd);
fd - deskryptor gniazda, które zostanie zamknięte

Zwraca: 0 (ok) lub -1 (błąd, ustawia errno).

setsockopt

int setsockopt(int socket, int level, int option_name, const void *option_value, socklen_t option_len);

socket - deskryptor gniazda
level - poziom protokołu np. SOL_SOCKET
option_name - nazwa ustawianej opcji np. SO_REUSEADDR
option_value - wartość opcji rzutowana na (void *) lub (char *)
option_len - rozmiar ustawianej wartości

Zwraca 0 (ok) lub -1 (błąd, ustawia errno na kod błędu).

setsockopt SO_REUSEADDR

```
int nFoo = 1;

setsockopt(nSocket, SOL_SOCKET, SO_REUSEADDR, (char*)&nFoo,
sizeof(nFoo)); //przed wywołaniem bind
```

Efekt: krótszy okres oczekiwania na możliwość ponownego dowiązania gniazda do tego samego adresu (IP:port).

Alternatywa: SO_REUSEPORT

Zadanie

Napisz serwer, który przyjmie dwie liczby od klienta, obliczy sumę oraz odeśle wynik. Napisz klienta, który wczyta dwie liczby z klawiatury i prześle do serwera. Następnie wypisze wynik na ekranie. Przetestuj swoje programy z osobą siedzącą obok.

Wskazówka: sprintf(), strtol() / sscanf(), fgets() / argv

Aplikacja UDP

TCP vs UDP

TCP

- Strumieniowy
- Gwarancja dostarczenia
- Porządek FIFO
- Kontrola przepływu

UDP

- Pakietowy
- Brak gwarancji dostarczenia
- Brak porządku FIFO

Funkcje wywoływane przez węzeł UDP

socket() - bind() - sendto() / recvfrom() - close()

socket()

Creates an endpoint for communication and returns a **file descriptor** that refers to that endpoint.

int socket(int domain, int type, int protocol);
domain e.g. AF_INET, AF_INET6, AF_UNIX
type e.g. SOCK_STREAM, SOCK_DGRAM
protocol e.g. 0 //wybierany automatycznie

Zwraca: deskryptor utworzonego gniazda lub -1 (błąd, ustawia errno).

bind()

assigns the address specified by addr to the socket referred to by the file descriptor sockfd

int bind(int sockfd, const struct sockaddr *addr, socklen_t addrlen);
sockfd - deskryptor gniazda
addr - adres, który zostanie przypisany gniazdu
addrlen - rozmiar struktury przekazanej jako addr

Zwraca: 0 (ok) lub -1 (błąd, ustawia errno).

bind()

W praktyce:

```
sa.sin_port = htons([port aplikacji]);
sa.sin_addr.s_addr = htonl(INADDR_ANY);

bind(sockfd, (struct sockaddr *) &sa, sizeof(sa));
```

sendto()

ssize_t sendto(int socket, const void *message, size_t length, int flags, const struct sockaddr *dest_addr, socklen_t dest_len);

socket - deskryptor gniazda sieciowego
message - bufor z wiadomością do wysłania
length - rozmiar wiadomości
flags - flagi, zostawiamy 0
dest_addr - struktura, dla UDP: sockaddr_in z adresem węzła docelowego
dest_len - rozmiar przekazanej struktury adresowej

Zwraca: liczbę wysłanych bajtów lub -1 (błąd, ustawia errno na kod błędu).

Problemy

- MTU
- Za dużo danych przekazanych do sendto?
- Brak gwarancji dostarczenia

recvfrom()

size_t recvfrom(int socket, void *buffer, size_t length, int flags, struct sockaddr *address, socklen_t *address_len);

socket - deskryptor gniazda sieciowego
buffer - bufor, do którego będą zapisywane odebrane dane
length - rozmiar buforu
flags - flagi, zostawiamy 0
address - dla UDP: sockaddr_in, wypełniane przez recvfrom danymi adresowymi osoby przesyłającej dane. Można wpisać NULL.
address_len - rozmiar przekazanej struktury adresowej. Można wpisać NULL.

Zwraca: rozmiar odebranej wiadomości lub 0 (gniazdo zamknięte) lub -1 (błąd, ustwaia errno na kod błędu).

Korzystanie z recvfrom()

- Gdy nie potrzebujemy danych nadawcy pakietu jako ostatnie dwa argumenty podajemy NULL.
- W innym wypadku jako przedostatni argument podajemy adres miejsca w pamięci gdzie mają zostać umieszczone dane adresowe, a jako ostatni argument (przez referencję) długość przygotowanego miejsca w bajtach. Po wykonaniu funkcji pod adres podany w ostatnim argumencie zostanie wpisany rozmiar struktury adresowej stworzonej przez recvfrom().

Przykład użycia (zakładamy z góry, że posługujemy się IPv4):

```
struct sockaddr_in sa;
socklen_t sa_size = sizeof(sa);
recvfrom(....., &sa, &sa_size);
```

close()

closes a file descriptor, so that it no longer refers to any file and may be reused.

int close(int fd);
fd - deskryptor gniazda, które zostanie zamknięte

Zwraca: 0 (ok) lub -1 (błąd, ustawia errno).

Zadanie

Napisz (sam lub w zespole 2os.) serwer daytime UDP i klienta daytime UDP.

Serwer: oczekuje na wiadomości od klientów, na każdą “odpowiada” bieżącą datą i czasem.

Klient: wysyła wiadomość do serwera, odbiera bieżącą datę i czas, wyświetla na ekranie.