Zagadnienia do trzeciego kolokwium

Typowy port dla usługi http/ftp

Port usługi HTTP: 80 Port usługi FTP: 21

Dodatkowe: Port usługi DNS: 53 Port usługi SSH: 22 Port usługi Telnet: 23 Port usługi DHCP (serwer): 67 Port usługi DHCP (klient): 68 Port usługi HTTPS: 443

Rodzaje komunikacji z użyciem gniazd

Komunikacja połączeniowa, oparta o protokół TCP, oraz komunikacja bezpołączeniowa, oparta na protokole UDP.

Czy do pisania/czytania do/z gniazd można używać standardowych funkcji uniksowych read/write

Tak, ponieważ gniazda to po prostu pliki identyfikowane w systemach unikowych poprzez ich deskryptor. Warunek jest taki, że musi to być komunikacja połączeniowa ponieważ żadna z tych metod nie ma możliwości podania adresu oraz portu obiektu docelowego.

Protokoły z warstwy 3 i 4 modeli OSI oraz TCP/IP

Model OSI Protokoły warstwy sieci (3): IP, RIP, EIGRP, OSPF Protokoły warstwy transportowej (4): TCP, UPD

Model TCP/IP Protokoły warstwy transportowej (3): TCP, UPD Protokoly warstwy aplikacji (4): HTTP, TELNET, SSH, FTP, POP3, SMTP, IMAP

Różnice między komunikacją z użyciem protokołów TCP i UDP

W TCP śledzone są pakiety, które wysłaliśmy, w UDP mamy to gdzieś, wysyłamy i zapominamy. TCP dba o to, żeby dane wysyłane do odbiorcy były kompletne, z UDP korzystamy raczej w przypadkach, kiedy nie zależy nam na tym czy wszystkie dane dotrą do odbiorcy. TCP jest protokołem połączeniowym, UDP nie. W TCP możliwa jest retransmisja danych.

Co to jest socket

Funkcja socket() tworzy nowe gniazdo, które może być wykorzystywane w komunikacji międzyprocesowej, w obrębie lokalnego systemu plików jak i za pomocą internetu.

Definicja funkcji:

```
int socket (int family, int type, int protocol);
```

Rezultat: deskryptor gniazda w przypadku, gdy gniazdo zostanie utworzone, -1 gdy wystąpi błąd.

Typy gniazd: SOCK_STREAM - gniazdo strumieniowe wykorzystujące protokół połączeniowy, np. TCP SOCK_DGRAM - gniazdo datagramowe wykorzystujące protokół bezpołączeniowy, np. UDP SOCK_RAW - gniazdo surowe wykorzystujące protokół warstwy sieciowej, np. IP

Enkapsulacja w kontekście komunikacji sieciowej

Kiedy dane przemieszczają się po poszczególnych szczeblach modelu OSI dodawane są do nich odpowiednie nagłówki i ogony, które zależą od protokołów zaimplemenotwanych dla poszczególnych warstw. W ten sposób powstają sumy kontrolne które wykorzystywane są w celu sorawdzenia, na urządzeniu odbiorcy, czy wszystkie dane dotarły, przez jakie protokoły mają być obsłużone itd.

Kolejność wywoływania funkcji komunikacji sieciowej połączeniowej przez proces będący w roli serwera

socket() -> bind() -> listen() -> accept() -> read() -> write()

Tworzenie gniazd: socket() Adresowanie: bind() Nawiązanie połączenia: listen(), accept() Komunikacja: read(), write()

Kolejność wywoływania funkcji komunikacji sieciowej połączeniowej przez proces będący w roli klienta

socket() -> connect() -> write() -> read()

Tworzenie gniazd: socket() Adresowanie: - Nawiązanie połączenia: connect() Komunikacja: read(), write()

Kolejność wywoływania funkcji komunikacji sieciowej bezpołączeniowej przez proces będący w roli serwera

socket() -> bind() -> recvfrom() -> sendto()

Tworzenie gniazd: socket() Adresowanie: bind() Komunikacja: recvfrom(), sendto()

Kolejność wywoływania funkcji komunikacji sieciowej bezpołączeniowej przez proces będący w roli klienta

socket() -> bind() -> sendto() -> recvfrom()

Tworzenie gniazd: socket() Adresowanie: bind() Komunikacja: recvfrom(), sendto()

Funkcje komunikacji sieciowej połączeniowej, które mogą zablokować proces

Są to funkcje: accept() oraz connect()

Rola funkcji bind()

Gniazda tworzone za pomoca funkcji socket() sa anonimowe. Aby do gniazda mogly dotrzec jakieś dane, wymagane jest przypisanie mu jednoznacznie identyfikujacej go nazwy (ang. binding), czyli adresu sieciowego komputera i numeru portu (w dziedzinie Internetu), bądź nazwy sciezkowej (dziedzina Unixa). Do tego celu służy funkcja bind():

```
int bind(int fd, struct sockaddr *my_addr, int addrlen)
```

Opis parametrów: **fd** - deskryptor gniazda **my_addr** - wsakźnik na strukturę adresu odpowiednią dla wybranej rodziny protokołów, do której należy gniazdo **addrlen** - rozmiar struktury my_addr

Wynik zwracany przez funkcję: 0 w przypadku sukcesu, -1, gdy błąd

Jawne przydzielenie adresu:

- serwer i klient w transmisji bezpołączeniowej
- serwer w transmisji połączeniowej

Rodzaje adresacji w komunikacji gniazdowej

Ogólna postać adresu gniazda:

```
struct sockaddr {
    u_short fa_family;  // rodzina adresów
    char sa_data[14];    // adres
}
```

Rodziny adresów: AF_UNIX (PF_UNIX) - wewnętrzne protokoły UNIX AF_INET (PF_INET) - protokoły Internetu TCP/IP

Jak określa się adres (jaki jest format adresu) w przypadku komunikacji sieciowej bazującej na protokole IPv4(AF_INET)

Adresy IP mają ogólnie formę 32-bitowych (lub 128-bitowych w wersji IPv6) liczb i mogą być zapisywane jako sekwencja czterech(ośmiu) liczb rodzielonych kropkami, np. 192.33.71.12 lub wykorzystując DNS jako nazwa_hosta.nazwa_domeny, np. google.com

Warstwy modelu sieciowego OSI

1. Warstwa fizyczna
2. Warstwa łącza danych
3. Warstwa sieci
4. Warstwa transportowa
5. Warstwa sesji
6. Warstwa prezentacji
7. Warstwa aplikacji

Funkcje accept(), sendto(), listen(), connect(), recvfrom()

Funkcja accept():

```
int accept(int s, struct sockaddr *addr, socklen_t *addrlen);
```

Pobiera pierwsze połączenie z kolejki oczekujacych, tworzy nowe gniazdo o tych samych właściwościach co stare gniazdo i zwraca jego deskryptor. Blokuje proces do momentu nawiązania nowego połączenia jeśli kolejka jest pusta.

Funkcja sendto():

```
int sendto(int s, const void *msg, size_t len, int flags, const struct
sockaddr *to, socklen_t tolen);
```

Wykorzystywana w komuniakcji bezpołączeniowej ponieważ można podać adres oraz port docelowy.

Funkcja listen():

```
int listen(int s, int backlog);
```

Zgłasza w systemie gotowość do przyjmowania połączeń oraz ustala maksymalną liczbę połączeń oczekujących na obsłużenie.

Funkcja connect():

```
int connect(int sockfd, const struct sockaddr *serv_addr, socklen_t
addrlen);
```

Konieczne dla gniazda strumieniowego. Nawiązuje połączenie z gniazdem strumieniowym serwera, blokuje proces klienta do momentu ustanowienia połączenia, tylko jedno pomyślne połączenie. Możliwe dla gniazda datagramowego. Zapamiętuje adres partnera komunikacji, można powtarzać wielokrotnie.

Funkcja recvfrom():

```
int recvfrom(int s, void *buf, size_t len, int flags, struct sockaddr
*from, socklen_t *fromlen);
```

Wykorzystywana w komuniakcji bezpołączeniowej ponieważ można podać adres oraz port, z którego chcemy uzyskać odpowiedź.

Jaką opcję należy ustawić funkcji socket(), aby możliwe było wysyłanie za jej pomocą wiadomości typu broadcast

Trzeba ustawić typ gniazda na SOCK_DGRAM:

```
socket(AF_INET, SOCK_DGRAM, 0)
```

Do czego służy funkcja inet_pton()

Konwertuje adresy IPv4 i IPv6 z postaci tekstowej na binarną.

```
int inet_pton(int af, const char *src, void *dst);
```

Do czego służy funkcja inet_ntoa()

Konwertuje adres internetowy do postaci stringowej składającej się z cyfr i kropek.

Do czego służy (co identyfikuje) adres IP a co port

IP to liczba przypisana do interfejsu sieciowego w maszynie, port to liczba przypisana do jakiejś usługi, z którą powiązany jest dany adres IP.

Co to jest broadcast

Broadcast – rozsiewczy (rozgłoszeniowy) tryb transmisji danych polegający na wysyłaniu przez jeden port (kanał informacyjny) pakietów, które powinny być odebrane przez wszystkie pozostałe porty przyłączone do danej sieci (domeny rozgłoszeniowej).

Używając logicznych wyrażeń bitowych operację tę możemy zapisać jako: `broadcast = adresIP | (! maska)`

Czy w komunikacji sieciowej można używać standardowych funkcji unixowych read() i write()? Czym różnią się od nich funkcje send() i recv()

Korzystając z recv() and send() mamy możliwość zdefiniowania kilku dodatkowych flag, które wpływają na specjalne zachowanie gniazda. Funkcje read() oraz write() działają dokładnie tak samo jak recv() i send() ale bez żadnych flag. Dodatkowo, w przypadku przesyłania datagramów lepiej sprawdzają się funkcje sendto() oraz recvfrom().

Z jakich źródeł korzysta funkcja gethostbyname

Korzysta ona z konfiguracji zawartej w pliku /etc/host.conf

Czy podane dwa hosty o adresach IP 192.168.1.218 oraz 192.168.1.186 należą do tej samej podsieci o masce 26-bitowej? Odpowiedź uzasadnij

IP1: 192.168.1.218 -> 11000000.10101000.00000001.11011010 IP2: 192.168.1.186 -> 11000000.10101000.00000001.10111010

Maska 26-bitowa (czyli 6 miejsce na hosta): 11111111.11111111.11111111.11000000 IP1 AND Maska: 11000000.10101000.00000001.11000000 IP2 AND Maska: 11000000.10101000.00000001.10000000

Adresy nie są identycze więc nie należą do tej samej podsieci.

Podaj IP broadcoast: Maska 255.255.255.240, przykładowy adres w tej sieci: 192.168.1.10

IP: 192.168.1.10 -> 11000000.10101000.00000001.00001010 Maska: 255.255.255.240 -> 11111111.11111111.11111111.11110000 !Maska: 00000000.00000000.00000000.00001111

Broadcast: 11000000.10101000.00000001.00001111

W jakim pliku na Linuxie są zapisywane mapowania port-usługa(np. HTTP - 80)

W pliku /etc/services

Funkcja, która konwertuje nazwę hosta na adres IP, szukając lokalnie oraz odpytując serwer DNS

gethostbyname()