Oprogramowanie komunikacyjne dla Internetu Rzeczy

Wykład 2

Komunikacja za pomocą gniazd

Dr hab. inż. Jacek Rak

Katedra Teleinformatyki WETI PG
EA 144

Wprowadzenie

- Komunikacja za pomocą gniazd (ang. Sockets) nie jest symetryczna: rola klienta i serwera musi być jasno określona
- Komunikacja połączeniowa (przy wykorzystaniu TCP) jest podobna do systemu wejścia-wyjścia plików (tzn. komunikacja jest z tym samym procesem partnerskim serwera/klienta)
- W przypadku komunikacji bezpołączeniowej (po UDP), każda operacja może być związana z innym procesem w innej stacji

Wprowadzenie

- **Konfigurując serwer musimy określić czy jest on współbieżny czy iteracyjny**

	Serwer iteracyjny	Serwer współbieżny
Protokół połączeniowy	rzadko spotykany (Daytime)	typowy
Protokół bezpołączeniowy	typowy	rzadko spotykany (TFTP)

- **Dla użytkownika oprogramowania-klienta nie ma to większego znaczenia. Jednakże w implementacji występują istotne różnice.**

Adresy gniazd

- **Ogólna struktura adresów (zawarta w `<sys/socket.h>`)**

```
struct sockaddr {
  unsigned short sa_family;    //adres rodziny – wartość AF_xxx
  char    sa_data[14];         //maks. 14 bajtów adresu
                                 właściwego protokołu
}
```

- **Struktura adresu dla rodziny protokołów Internetu (zawarta w `<netinet/in.h>`)**

```
struct in_addr {
  unsigned long s_addr;  //32-bitowy identyfikator sieci/stacji
}

struct sockaddr_in {
  short sin_family;         //AF_INET
  unsigned short sin_port; //16-bitowy numer portu
  struct in_addr sin_addr; //32-bitowy identyfikator sieci/stacji
  char  sin_zero[8];        //nie jest używane
}
```

Adresy gniazd – sposób przekazywania

```
struct sockaddr_in serv_addr; //struktura adresu
                              //właściwa dla protokołu
                    .         //Internetu
                    .
                    .

connect(sockfd, (struct sockaddr*) &serv_addr,
        sizeof(serv_addr));
```

- drugi parametr wywołania funkcji `connect` to wskaźnik na strukturę adresu `sockaddr`
- trzeci parametr określa rozmiar struktury adresu
- Należy zwrócić uwagę na rzutowanie do ogólnego typu struktury adresu `sockaddr` umożliwiające wykorzystanie funkcji `connect` również w przypadku adresów innych typów (np. XNS, Unix)

Podstawowe funkcje

socket() Funkcja tworząca gniazdo (zwraca **deskryptor gniazda** `sockfd`)

```
#include <sys/types.h>
#include <sys/socket.h>
int socket(int family, int type, int protocol);
```

family:

- `AF_INET` – **protokoły Internetu**
- `AF_UNIX` – **protokoły Unixa**

type (rodzaj gniazda):

- `SOCK_STREAM` – **gniazdo strumieniowe**
- `SOCK_DGRAM` – **gniazdo datagramowe**
- `SOCK_RAW` – **gniazdo surowe**

Powiązanie wartości pola *protocol* z wartością pola *type*

type	protocol	protokół rzeczywisty
SOCK_DGRAM	IPPROTO_UDP	UDP
SOCK_STREAM	IPPROTO_TCP	TCP
SOCK_RAW	IPPROTO_ICMP	ICMP
SOCK_RAW	IPPROTO_RAW	IP

Podstawowe funkcje

bind() Funkcja łącząca gniazdo z portem i adresem IP

```
#include <sys/types.h>
#include <sys/socket.h>
int bind(int sockfd, struct sockaddr *myaddr, int adrlen);
```

- Pierwszy parametr (*sockfd*) określa deskryptor gniazda; drugi (*myaddr*) wskazuje strukturę zawierająca adres; trzeci (*adrlen*) określa wielkość struktury adresu
- Funkcja służy do odnotowania w systemie informacji o adresie serwera/klienta, oraz do upewnienia się, że jego adres jest niepowtarzalny
- łączy aplikację z określonym portem i adresem IP

Podstawowe funkcje

connect() Funkcja nawiązująca połączenie z serwerem

```
#include <sys/types.h>
#include <sys/socket.h>
int   connect(int   sockfd,   struct   sockaddr   *servaddr,
          int adrlen);
```

- Pierwszy parametr (*sockfd*) określa deskryptor gniazda klienta; drugi (*servaddr*) wskazuje strukturę zawierająca adres serwera; trzeci (*adrlen*) określa wielkość struktury adresu;

- Funkcja może być również wykorzystana w przypadku komunikacji bezpołączeniowej. Wówczas można wykorzystać standardowe dla transmisji połączeniowej funkcje wysłania (write) i odbioru (read) informacji, bez konieczności każdorazowego podawania adresu docelowego serwera

Podstawowe funkcje

listen() Funkcja używana przez serwer połączeniowy. Przygotowuje gniazdo do odbierania połączeń.

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

- Pierwszy parametr (*sockfd*) określa deskryptor gniazda na którym przeprowadza się nasłuchiwanie; drugi (*backlog*) definiuje liczbę żądań połączeń, jakie serwer może umieścić w swojej kolejce

- Funkcja jest przeważnie wywoływana po funkcjach `socket` i `bind`, a przed funkcją `accept`
- Maksymalna dopuszczalna wartość parametru *backlog* wynosi 5

Podstawowe funkcje

accept() Funkcja akceptacji żądania ustanowienia połączenia czekającego w kolejce

```
#include <sys/types.h>
#include <sys/socket.h>
int accept(int sockfd, struct sockaddr *peer, int *addrlen);
```

- Funkcja bierze pierwsze żądanie z kolejki i tworzy drugie gniazdo o takich samych właściwościach jak gniazdo o identyfikatorze *sockfd*. Prametry *peer i addrlen* określają adres klienta, którego dotyczy żądanie.
- Parametr *addrlen*:
  - przed wywołaniem funkcji określa długość struktury adresu klienta
  - po wywołaniu może zawierać rezultat wywołania funkcji

Podstawowe funkcje

accept()

Serwer współbieżny - przykład

```
int sockfd, newsockfd;
if((sockfd = socket(...) < 0)
  err_sys("socket error");
if((bind(sockfd, ...) < 0)
   err_sys("bind error");
if((listen(sockfd, 5) < 0)
   err_sys("listen error");
for( ; ; ){
  newsockfd=accept(sockfd, ...); //blokowane wykonanie kolejnej funkcji
  if(newsockfd < 0)
    err_sys("accept error");

  if(fork() == 0){                    //wywoływane wewnątrz potomka
     close(sockfd);
     doit(newsockfd);                 //obsługa żądania
     exit(0);
  }
  close(newsockfd);                   //usunięcie przodka
}
```

Podstawowe funkcje

accept()

Serwer równoległy - przykład

```
int sockfd, newsockfd;
if((sockfd = socket(...) < 0)
  err_sys("socket error");
if((bind(sockfd, ...) < 0)
   err_sys("bind error");
if((listen(sockfd, 5) < 0)
   err_sys("listen error");
for( ; ; ){
  newsockfd=accept(sockfd, ...); //blokowane wykonanie kolejnej funkcji
  if(newsockfd < 0)
    err_sys("accept error");

  doit(newsockfd);                    //obsługa żądania
  close(newsockfd);
}
```

Podstawowe funkcje

```
#include <sys/types.h>
#include <sys/socket.h>

int send(int sockfd, char *buff, int nbytes, int flags);

int sendto(int sockfd, char *buff, int nbytes, int flags,
           struct sockaddr *to, int addrlen);

int recv(int sockfd, char *buff, int nbytes, int flags);

int recvfrom(int sockfd, char *buff, int nbytes, int flags,
             struct sockaddr *from, int* addrlen);
```

Parametr flags **typowo przyjmuje wartość 0**

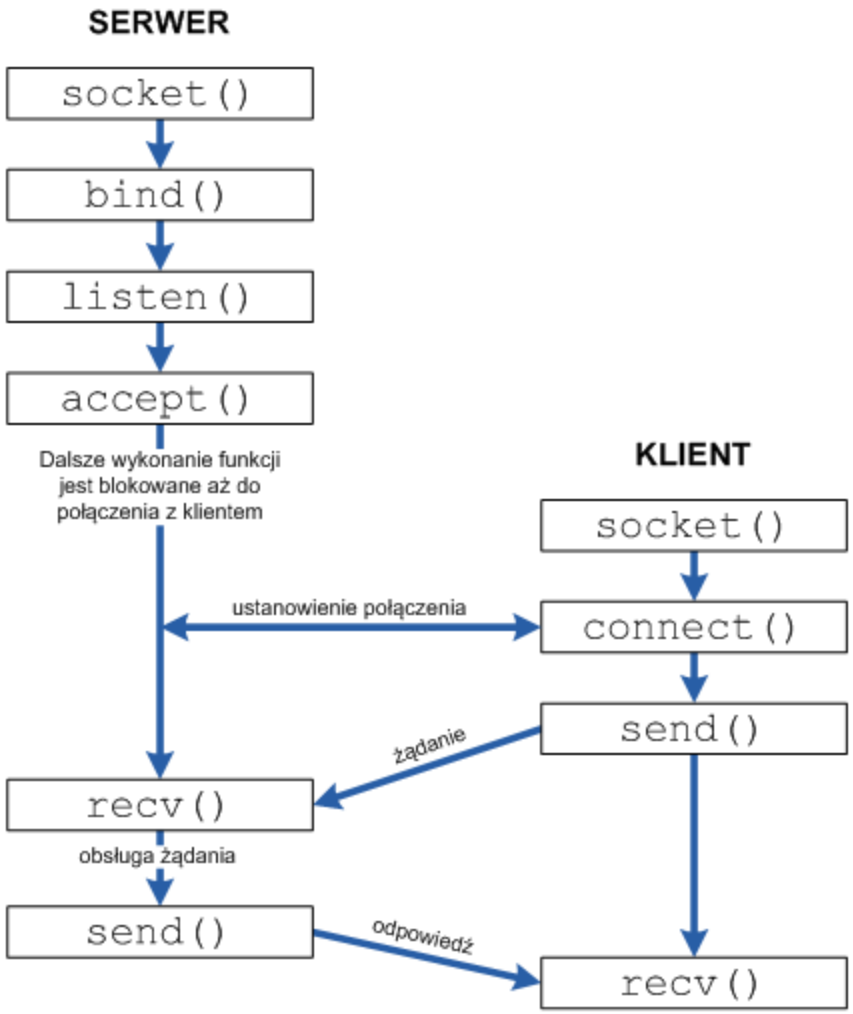
Podstawowe funkcje

close() Zamyka gniazdo

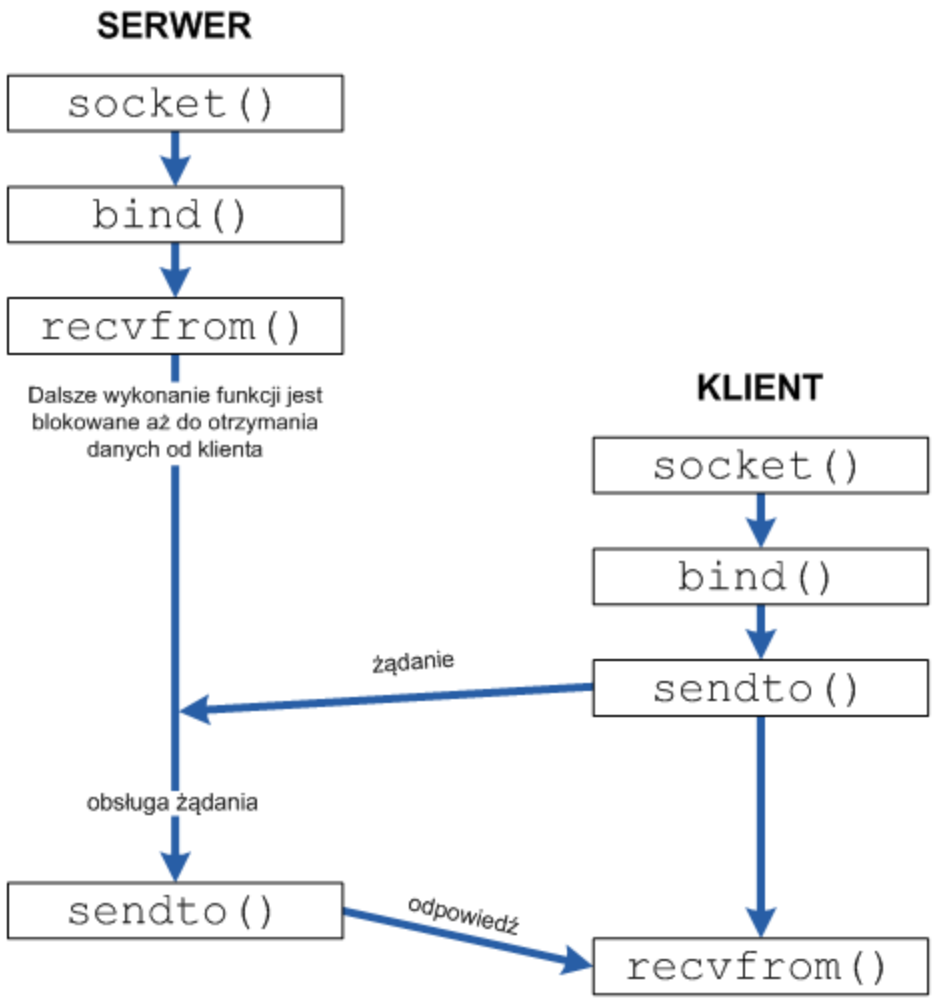
```
#include <sys/types.h>
#include <sys/socket.h>
int close(int sockfd);
```

- W przypadku niezawodnego dostarczania danych (tryb połączeniowy poprzez TCP), wszystkie dane pozostające do wysłania (wewnątrz jądra) muszą być przesłane/potwierdzone.
- Powrót z wykonania funkcji `close` ma miejsce przeważnie natychmiast, lecz dla wszystkich danych będących w kolejce, jądro próbuje przeprowadzić operację wysłania

Przykład: protokół połączeniowy



Przykład: protokół bezpołączeniowy



Komunikacja w systemie Windows - WinSock

Po stronie serwera:

- inicjalizacja Winsock
- Create
- Bind
- Listen
- Accept (connection)
- Receive/Send (data)
- Disconnect

Po stronie klienta:

- inicjalizacja Winsock
- Create
- Connect
- Send/Receive (data)
- Disconnect

Komunikacja w systemie Windows - WinSock

```
#include <winsock2.h>
#include <ws2tcpip.h>
#include <stdio.h>

#pragma comment(lib, "Ws2_32.lib")

int main() {
  return 0;
}
```

Komunikacja w systemie Windows - WinSock

Inicjalizacja użycia biblioteki WinSock2:

(musi być przeprowadzona przed wywołaniem funkcji w niej zawartych)

1) Stworzenie obiektu wsaData

```
WSAData wsaData;
```

2) Wywołanie funkcji WSAStartup w celu zainicjowania użycia **WS2_32.dll** oraz sprawdzenie ewentualnych błędów

```
int iResult;

iResult = WSAStartup(MAKEWORD(2,2), &wsaData);
if (iResult != 0) {
    printf("WSAStartup failed: %d\n", iResult);
    return 1;
}
```

Polecenie `MAKEWORD(2,2)` wymusza wersję 2.2 Winsock

Komunikacja w systemie Windows - WinSock

Stworzenie Socketu typu STREAM

```
SOCKET mysock = socket(AF_INET, SOCK_STREAM, 0);
if (mysock == INVALID_SOCKET) {
  cout << "Socket Creation Failed!" << endl;
  cout << "Error Code:  " << WSAGetLastError() <<
 endl;
  system("pause >nul");
  WSACleanup();
  return 0;
}
```

Komunikacja w systemie Windows - WinSock

Połączenie z serwerem (po stronie klienta)

```
sockaddr_in sin;
sin.sin_port = htons(2000);
sin.sin_addr.s_addr = inet_addr("193.107.33.44");
sin.sin_family = AF_INET;

if (connect(mysock,(sockaddr*)&sin, sizeof(sin)) == INVALID_SOCKET) {
  cout << "Socket Connection Failed" << endl;
  cout << "Error Code:  " << WSAGetLastError() << endl;
  system("pause >nul");
  closesocket(mysock);
  WSACleanup();
  return -1;
}
```

Komunikacja w systemie Windows - WinSock

Zakończenie komunikacji (po stronie klienta)

```
WSACleanup();
closesocket(mysock);
```

Funkcja WSACleanup kończy korzystanie z biblioteki WS2_32 DLL

Komunikacja w systemie Windows - WinSock

Stworzenie Socketu typu DGRAM

```
SOCKET mysock = socket(AF_INET, SOCK_DGRAM, IPPROTO_UDP);
if (mysock == INVALID_SOCKET) {
  cout << "Socket Creation Failed!" << endl;
  cout << "Error Code:  " << WSAGetLastError() << endl;
  system("pause >nul");
  WSACleanup();
  return 0;
}
```

Komunikacja w systemie Windows - WinSock

Przesłanie wiadomości od klienta do serwera

```
sockaddr_in sin;
sin.sin_port = htons(2000);
sin.sin_addr.s_addr = inet_addr("193.107.33.44");
sin.sin_family = AF_INET;

int buflen = 200;
char buf[buflen];

strcpy(buf, "To ja twój klient");

int iResult = sendto(mysock,
                  buf, buflen, 0, (SOCKADDR *) & sin, sizeof (sin));
if (iResult == SOCKET_ERROR) {
   cout <<"sendto failed!" <<endl;
   cout << "Error Code:  " << WSAGetLastError() << endl;
   system("pause >nul");
   closesocket(mysock);
   WSACleanup();
   return 1;
}
```

Komunikacja w systemie Windows - WinSock

Obsługa żądań po stronie serwera (tryb STREAM)

```
sockaddr_in sin;
sin.sin_port = htons(2000);
sin.sin_addr.s_addr = inet_addr("193.107.33.44");
sin.sin_family = AF_INET;

if (bind(mysock,(sockaddr*)&sin, sizeof(sin)) == SOCKET_ERROR){
  cout << "Socket failed to bind!" << endl;
  cout << "Error Code: " << WSAGetLastError() << endl;
  system("pause >nul");
  WSACleanup();
  return 0;
}
cout << "Socket Binded Successfuly!" << endl;
system("pause >nul");

//nasłuchiwanie rozpoczęte
while (listen(mysock, SOMAXCONN) == SOCKET_ERROR);
```

Komunikacja w systemie Windows - WinSock

Obsługa żądań po stronie serwera (tryb STREAM)

```
SOCKET client;
int lin = sizeof(sin);
client = accept(mysock,(sockaddr*) &sin, &lin);
cout << "Connection Established!" << endl;
char buf[200] = "Hello\n";
send(client, buf, sizeof(buf), 0);
closesocket(mysock);
closesocket(client);
WSACleanup();
```

Komunikacja w systemie Windows - WinSock

Obsługa żądań po stronie serwera (tryb DGRAM)

```
SOCKET mysock = socket(AF_INET, SOCK_DGRAM, IPPROTO_UDP);
if (mysock == INVALID_SOCKET) {
  cout << "Socket Creation Failed!" << endl;
  cout << "Error Code: " << WSAGetLastError() << endl;
  system("pause >nul");
  WSACleanup();
  return 0;
}
cout << "Socket Creation Successful!" << endl;
system("pause >nul");

sockaddr_in sin;
sin.sin_port = htons(2000);
sin.sin_addr.s_addr = inet_addr("193.107.33.44");
sin.sin_family = AF_INET;
```

Komunikacja w systemie Windows - WinSock

Obsługa żądań po stronie serwera (tryb DGRAM)

```
if (bind(mysock,(sockaddr*)&sin, sizeof(sin)) == SOCKET_ERROR){
  cout << "Socket failed to bind!" << endl;
  cout << "Error Code: " << WSAGetLastError() << endl;
  system("pause >nul");
  WSACleanup();
  return 1;
}
cout << "Socket Binded Successfuly!" << endl;
system("pause >nul");

cout << "Process of receiving datagrams has been started!" << endl;

sockaddr_in SenderAddr;
int SenderAddrSize = sizeof (SenderAddr);

int iResult;
int iBufLen =1000;
char mybuffer[iBufLen];
```

Komunikacja w systemie Windows - WinSock

Obsługa żądań po stronie serwera (tryb DGRAM)

```
iResult = recvfrom(mysock, mybuffer, iBufLen, 0, (SOCKADDR *) & SenderAddr,
                   &SenderAddrSize);
if (iResult == SOCKET_ERROR) {
 cout << "recfrom failed!" << endl;
 cout << "Error Code: " << WSAGetLastError() << endl;
 system("pause >nul");
}
//wypisanie odebranego tekstu
  cout << mybuffer <<endl;
system("pause >nul");
cout << "Finished receiving. Closing socket.";
iResult = closesocket(mysock);
if (iResult == SOCKET_ERROR) {
  cout << "closing failed!" << endl;
  cout << "Error Code: " << WSAGetLastError() << endl;
  system("pause >nul");
  return 1;
}
WSACleanup();
```

Komunikacja w systemie Windows - WinSock

Dodatkowe materiały:

http://msdn.microsoft.com/en-us/library/windows/desktop/ms738545(v=vs.85).aspx

książka pod adresem http://www.jimprice.com/winsock/winsock2api-withtoc.PDF