# Gniazda BSD

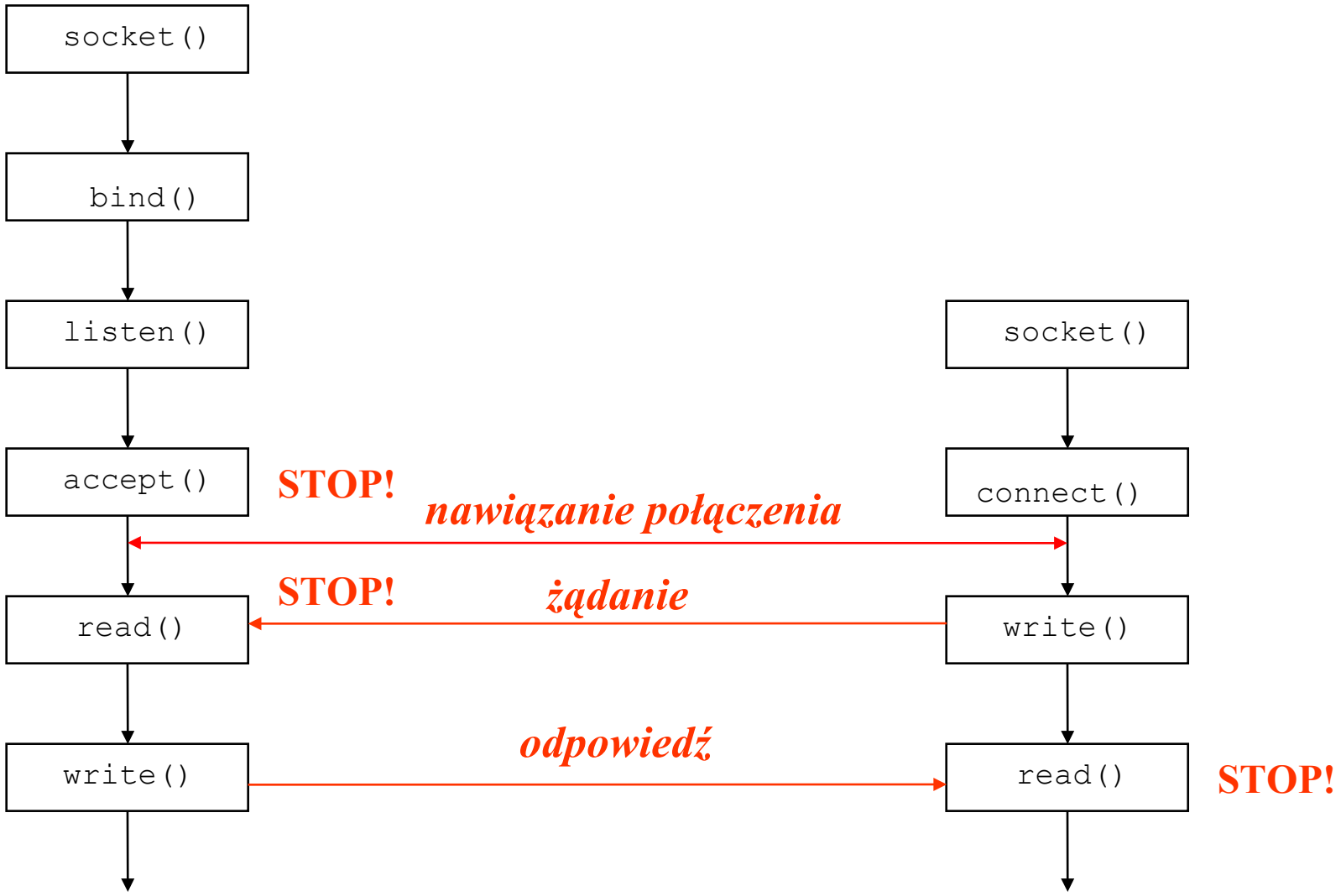
Literatura uzupełniająca:

**W. Richard Stevens,**

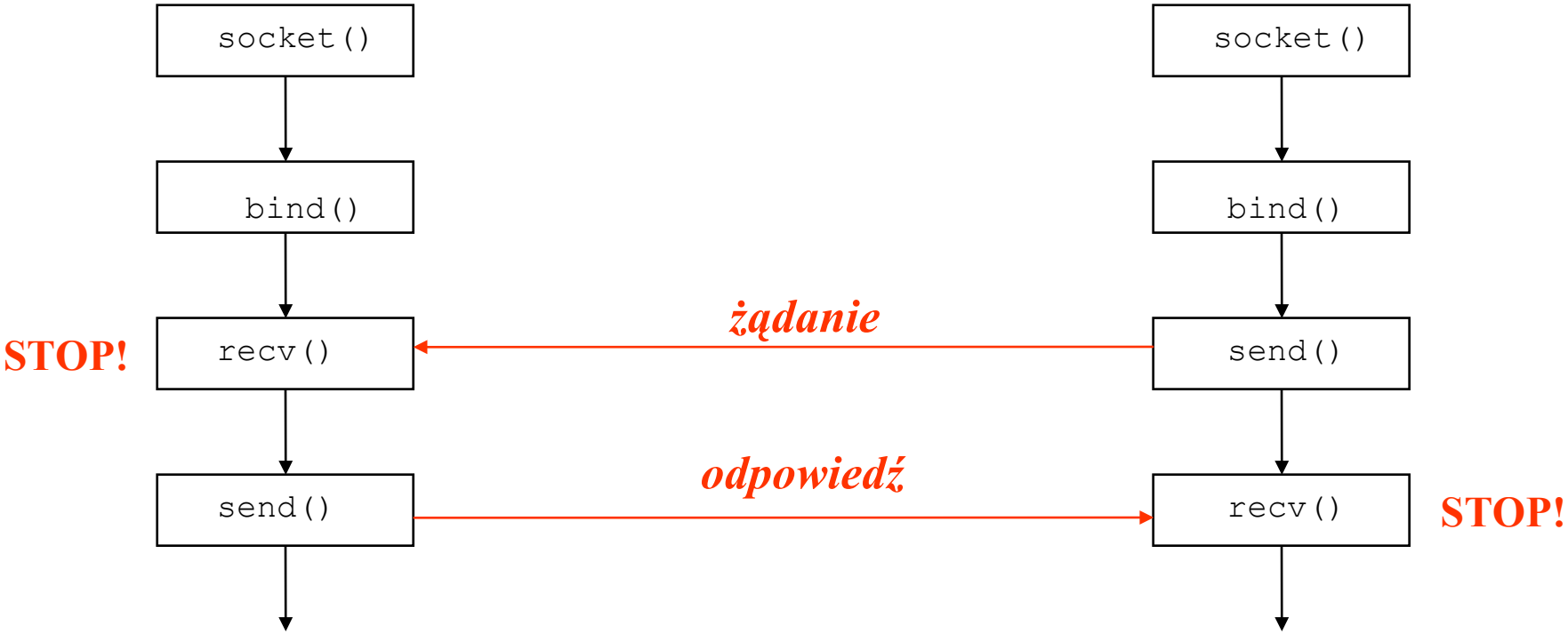
***Programowanie zastosowań sieciowych w systemie Unix***

WNT 1998

# Użycie gniazd w transmisji połączeniowej



# Użycie gniazd w transmisji bezpołączeniowej



# Funkcja systemowe dla gniazd

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

***gdzie***:

*domain*

- **AF_INET,** AF_UNIX, AF_INET6, ...

*type*

- **SOCK_STREAM**, SOCK_DGRAM, SOCK_RAW, SOCK_SEQPACKET, ...

*proto*

- dla domain=AF_INET → TCP, UDP, IP (dostarczone przez `getprotobyname()`

***wynik***:

uchwyt do nowo utworzonego gniazda albo -1 (→ errno)

# Funkcja systemowe dla gniazd

```
#include <sys/types.h>
#include <sys/socket.h>
int bind(
    int sockfd,
    const struct sockaddr *addr,
    socklen_t addrlen);
```

***gdzie***:

*sockfd*

- uchwyt skojarzony z gniazdem, dostarczony przez `socket()`

*sockaddr*

- wskaźnik na strukturę opisującą usługę, której istnienie serwer ogłasza w systemie (trójka: domena, port, adres IP)

*adrlen*

- długość struktury j.w.

***wynik***:

0 jeśli wykonanie poprawne, -1 w wypadku błędu (→ errno)

# Funkcja systemowe dla gniazd

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

struct sockaddr {
   sa_family_t     sa_family;
   char            sa_data[14]
}

struct sockaddr_in {
   short int       sin_family;
   unsigned short  int sin_port;
   struct in_addr  sin_addr;
   unsigned char   sin_zero[8];
}
```

# Funkcja systemowe dla gniazd

```
#include <sys/types.h>
#include <sys/socket.h>
int connect(int sockfd,
    const struct sockaddr *addr,
    socklen_t addrlen);
```

***gdzie***:

*sockfd*

- uchwyt skojarzony z gniazdem, dostarczony przez `socket()`

*sockaddr*

- wskaźnik na strukturę opisującą zdalną usługę, do której należy się dołączyć (jak poprzednio)

*sockaddr*

- długość struktury j.w.

***wynik***:

0 jeśli wykonanie poprawne, -1 w wypadku błędu (→ errno)

# Funkcja systemowe dla gniazd

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

***gdzie***:

*sockfd*

- uchwyt do gniazda, dostarczony przez `socket()`

*backlog*

- rozmiar kolejki, którą będzie utrzymywać system, aby przechowywać żądania kierowane do gniazda

***wynik***:

0 jeśli wykonanie poprawne, -1 w wypadku błędu (→ errno)

# Funkcja systemowe dla gniazd

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

***gdzie:***

*sockfd*

- uchwyt do gniazda, na którym spodziewamy się odebrania połączenia

*addr*

- wskaźnik na strukturę, która zostanie wypełniona adresem usługi, z której nadeszło połączenie

*addrlen*

- długość utworzonej struktury

***wynik:***

uchwyt do nowo utworzonego gniazda albo -1 w wypadku błędu (→ errno)

# Funkcja systemowe dla gniazd

```
#include   <sys/types.h>
#include   <sys/socket.h>
ssize_t send(
   int sockfd, const void *buf,
   size_t len, int flags);
```

***gdzie***:

*sockfd*

- uchwyt skojarzony z gniazdem,przez które zostanie wysłany komunikat, dostarczony przez `socket()`

*buf*

- wskaźnik na bufor z komunikatem

*len*

- długość wiadomości

*flags*

- 0, MSG_DONTROUTE, MSG_OOB, ...

***wynik***:

liczba wysłanych bajtów jeśli wykonanie poprawne albo -1 w wypadku błędu (→ errno)

# Funkcja systemowe dla gniazd

```
#include   <sys/types.h>
#include   <sys/socket.h>
ssize_t recv(
   int sockfd, const void *buf,
   size_t len, int flags);
```

***gdzie***:

*sockfd*

- uchwyt skojarzony z gniazdem, przez które zostanie odebrany komunikat, dostarczony przez `socket()`

*buf*

- wskaźnik na bufor z komunikatem

*len*

- długość bufora

*flags*

- 0, MSG_DONTROUTE, MSG_OOB, ...

***wynik***:

liczba odebranych bajtów jeśli wykonanie poprawne albo -1 w wypadku błędu (→ errno)

# Funkcja systemowe dla gniazd

```
#include   <unistd.h>
#include   <sys/socket.h>
int close(int fd);
```

***gdzie***:
*fd*

- uchwyt skojarzony z gniazdem, dostarczony przez `socket()`

***wynik:***
0 jeśli wykonanie poprawne albo -1 w wypadku błędu (→ errno)

# Funkcja systemowe dla gniazd

```
#include <sys/types.h>
#include <sys/socket.h>
int inet_aton(
   const char *cp, struct in_addr *inp);
```

***gdzie***:

*cp*

- wskaźnik na adres IP hosta w postaci znakowej (a.b.c.d)

*inp*

- wskaźnik na strukturę, w której zostanie umieszczony adres w postaci „binarnej”

***wynik:***

0 jeśli wykonanie poprawne albo -1 w wypadku błędu (→ errno)

# Funkcja systemowe dla gniazd

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

typedef uint32_t in_addr_t;

struct in_addr {
          in_addr_t s_addr;
};
```

# Funkcja systemowe dla gniazd

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

in_addr_t inet_addr(const char *cp);
```

***gdzie***:

*cp*

- wskaźnik na adres hosta (domenowy) w postaci znakowej

***wynik:***

spakowany adres IP albo -1 w wypadku błędu (→ errno) (co rodzi pewien problem...)

# Funkcja systemowe dla gniazd

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

char *inet_ntoa(struct in_addr in);
```

***gdzie***:

*cp*

- wskaźnik na adres IP hosta w spakowanej postaci „binarnej”

***wynik:***

adres IP w postaci znakowej (a.b.c.d) umieszczony w statycznym buforze (nadpisywanych przy każdym kolejnym wywołaniu)

# Funkcja systemowe dla gniazd

```
#include <netdb.h>
struct protoent *getprotobyname(
            const char *name);
```

***gdzie:***

*name*

- łańcuch zawierający nazwę protokołu (”tcp”, ”udp”, ”ip”)

***wynik:***

wskaźnik na statyczny utworzoną strukturę (nadpisywaną przy kolejnych wywołaniach) opisującą protokół lub NULL w przypadku błędu

# Funkcja systemowe dla gniazd

```
#include <netdb.h>

 struct protoent {
  char  *p_name;      /* official protocol name */
  char **p_aliases;   /* alias list */
  int    p_proto;     /* protocol number */
}
```

# Funkcja systemowe dla gniazd

```
#include <netdb.h>
struct protoent
*getprotobynumber(int proto);
```

***gdzie:***

*proto*

- numer protokołu

***wynik:***

wskaźnik na statyczny utworzoną strukturę (nadpisywaną przy kolejnych wywołaniach) opisującą protokół lub NULL w przypadku błędu

# Funkcja systemowe dla gniazd

```
#include <netdb.h>
struct servent *getservbyname
(const char *name, const char *proto);
```

***gdzie:***

*name*

- łańcuch zawierający nazwę usługi (np. ”pop3”)

*proto*

- łańcuch zawierający nazwę protokołu (”tcp”, ”udp”, ”ip”)

***wynik:***

wskaźnik na statyczny utworzoną strukturę (nadpisywaną przy kolejnych wywołaniach) opisującą daną usługę lub NULL w przypadku błędu

# Funkcja systemowe dla gniazd

```
#include <netdb.h>

struct servent {
        char  *s_name;          /* official service name */
        char **s_aliases;       /* alias list */
        int    s_port;          /* port number */
        char  *s_proto;         /* protocol to use */
}
```

# Funkcja systemowe dla gniazd

```
#include <netdb.h>
struct servent *getservbyport
(int port, const char *proto);
```

***gdzie:***

*name*

- numer portu przypisany do usługi (np. 110)

*proto*

- łańcuch zawierający nazwę protokołu (”tcp”, ”udp”, ”ip”)

***wynik:***

wskaźnik na statyczny utworzoną strukturę (nadpisywaną przy kolejnych wywołaniach) opisującą daną usługę lub NULL w przypadku błędu

# Funkcja systemowe dla gniazd

```
#include <netdb.h>
struct hostent
*gethostbyname(const char *name);
```

*gdzie:*

name

- nazwa hosta w postaci łańcucha (symboliczna lub adres IP)

*wynik:*
wskaźnik na statyczny utworzoną strukturę (nadpisywaną przy kolejnych wywołaniach) opisującą danego hosta lub NULL w przypadku błędu

# Funkcja systemowe dla gniazd

```
#include <netdb.h>

struct hostent {
    char  *h_name;        /* official name of host */
    char **h_aliases;     /* alias list */
    int    h_addrtype;    /* host address type */
    int    h_length;      /* length of address */
    char **h_addr_list;   /* list of addresses */
}
```

# Funkcja systemowe dla gniazd

```
#include <netdb.h>
struct hostent *gethostbyaddr(
const void *addr, socklen_t len, int type);
```

***gdzie***:

*addr*

- wskaźnik na strukturę opisującą adres IP hosta (np. struct in_addr uzyskany z inet_addr())

*len*

- długość (rozmiar) struktury

*type*

- typ adresu (w naszym przypadku AF_INET)

***wynik***:

wskaźnik na statyczny utworzoną strukturę (nadpisywaną przy kolejnych wywołaniach) opisującą danego hosta lub NULL w przypadku błędu

# Funkcja systemowe dla gniazd

```
#include <sys/types.h>
#include <sys/socket.h>
int setsockopt(int sockfd, int level,
int optname, const void *optval, socklen_t optlen);
```

***gdzie***:

*sockfd*
- uchwyt do gniazda, którego opcje będą modyfikowane

*level*
- poziom modyfikacji (SOL_SOCKET)

*optname*
- stała symboliczna reprezentująca opcję (SO_REUSEADDR)

*optval*
- wskaźnik na daną reprezentującą nową wartość opcji (najczęściej int)

optlen
- długość danej reprezentującej opcję

***wynik***:

0 jeśli wykonanie poprawne albo -1 w wypadku błędu (→ errno)

# Prosty klient TCP 1/3

```
#include <stdio.h>
#include <stdlib.h>
#include <errno.h>
#include <netdb.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <string.h>

void err(char *where) {
    fprintf(stderr, "error in %s: %d\n", where, errno);
    exit(1);
}

int main(int argc, char *argv[]) {
    char *remote = "www.wi.zut.edu.pl";
    struct servent *sent;
    struct protoent *pent;
    int  port;
    int  sock;
    int  result;
    in_addr_t ipadr;
    struct sockaddr_in addr;
    struct hostent *hent;
    char buf[2048];
```

# Prosty klient TCP 2/3

```
sent = getservbyname("http", "tcp");
if(sent == NULL)
    err("getservbyname");
port = sent->s_port;
pent = getprotobyname("tcp");
if(pent == NULL)
    err("getprotobyname");
hent = gethostbyname(remote);
printf("Host: %s\n", hent->h_name);
printf("IP: %s\n", inet_ntoa(*((struct in_addr *)hent->h_addr)));
addr.sin_family = AF_INET;
addr.sin_port = port;
addr.sin_addr = *((struct in_addr *)hent->h_addr);
memset(addr.sin_zero, '\0', 8);
sock = socket(AF_INET, SOCK_STREAM, pent->p_proto);
if(sock < 0)
    err("socket");
result = connect(sock, (struct sockaddr *)&addr, sizeof(struct sockaddr));
if(result < 0)
    err("connect");
```

# Prosty klient TCP 3/3

```
    strcpy(buf, "GET / \r\n\r\n");
    write(sock, buf, strlen(buf));
    while((result = read(sock, buf, sizeof(buf))) > 0) {
        buf[result] = '\0';
        puts(buf);
    }
    close(sock);
    return 0;
}
```