Síťové programování

- Berkeley sockets
- Zdroje
  - Wikipedia
  - Google
  - Jan Kubr

Co víte o souborech a operacích s nimi

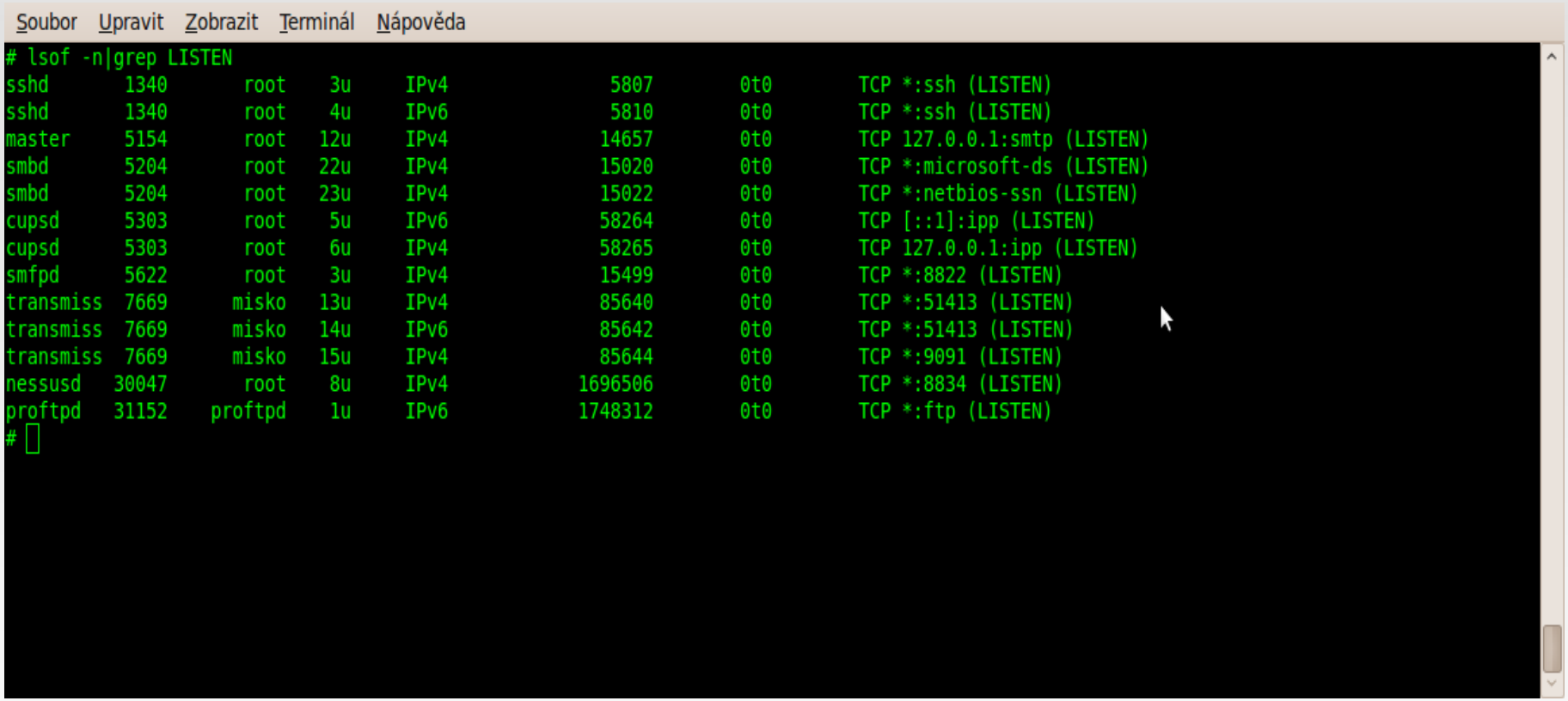
- Deskriptor (fd) – popisovač, číslo
- Zápis (write, fwrite, …) - deskriptor, data
- Čtení (read, fread, …) - deskriptor, data

Sockety

- Socket = koncový bod dvousměrné komunikace, koncept využívaný v řadě OS. V našem kontextu bereme jako koncový bod IP-enabled zařízení
- Síťový socket je reprezentován adresou soketu = kombinace IP(4/6) adresy a portu
  - Např. 147.32.83.180:12345
- Socket je operačním systémem mapován na konkrétní proces či thread

Ukázka socketů z Linuxu

- Příkaz lsof



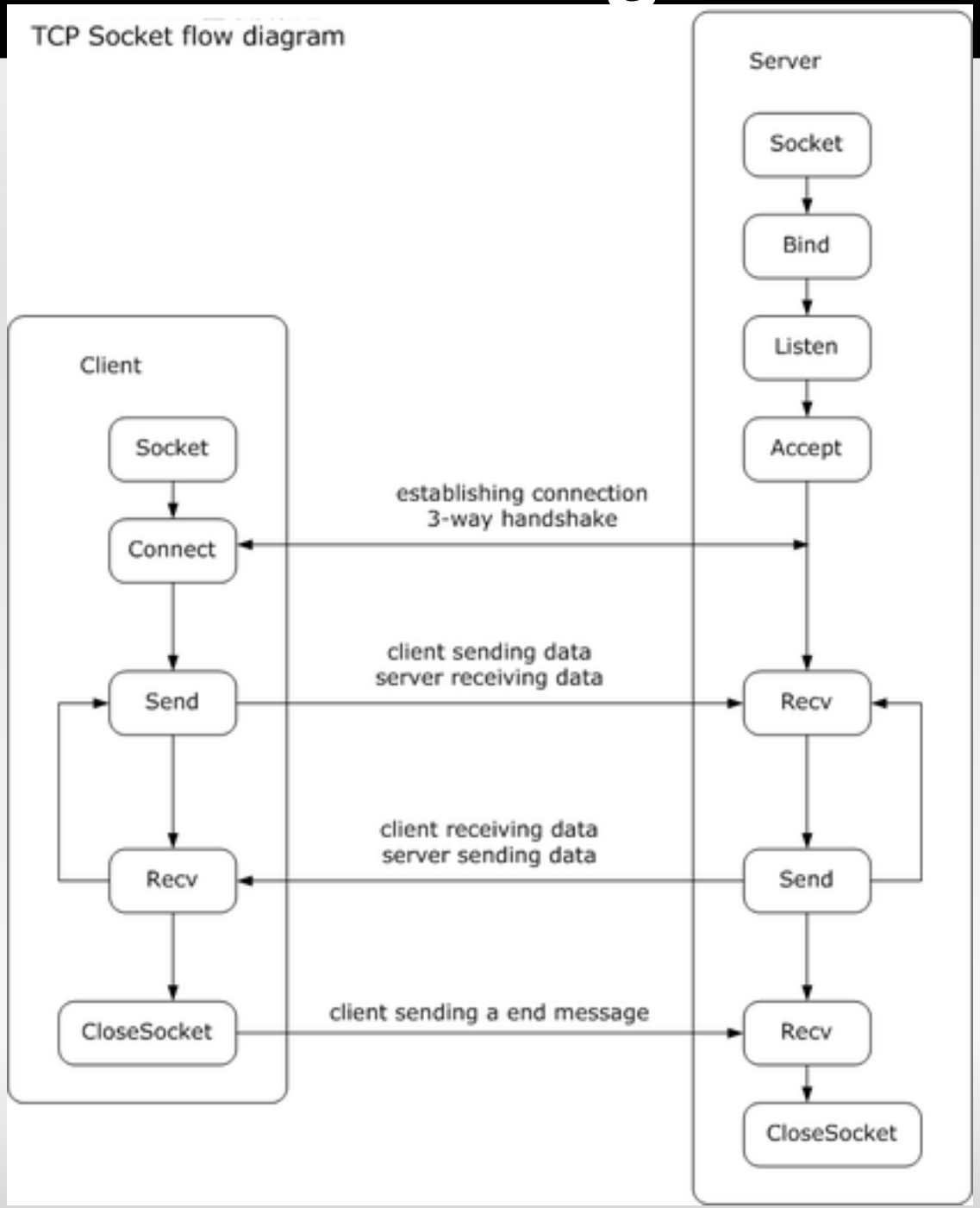
Typy soketů

- Datagram socket
  - Connectionless, pro UDP
- Stream socket
  - Connection-oriented, TCP
- Raw socket
  - Transportní vrstva se vynechá
  - Typické aplikace – ICMP, IGMP, OSPF, ...

Implementace socketů

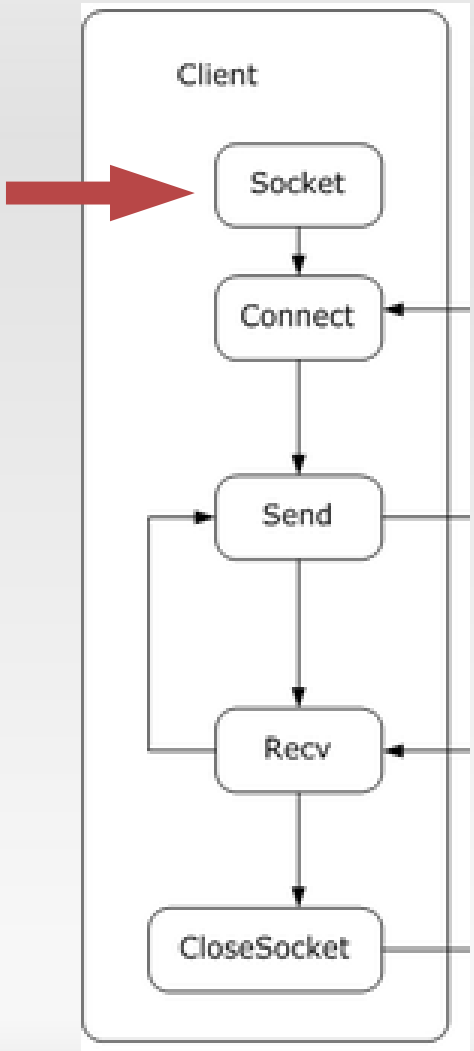
- Berkeley sockets, 1983
- Operační systém 4.2BSD
- C
- Z BSD sockets vycházejí další implementace
  - Winsock – windows
  - atd

TCP Socket flow diagram



Klient: socket()

- Volání socket() - vytvoří socket – datovou strukturu. Vrací deskriptor socketu (číslo)
- `sockfd = socket(int socket_family, int socket_type, int protocol);`
  - `socket_family = PF_UNIX|PF_LOCAL|`**`PF_INET`**`|PF_INET6| PF_IPX|PF_NETLINK|PF_X25|PF_AX25|PF_ATMPVC| PF_APPLETALK|PF_PACKET`
  - `socket_type =` **`SOCK_STREAM|SOCK_DGRAM|`** `SOCK_SEQPACKET|SOCK_RAW|WOCK_RDM|SOCK_PACKET`
  - `protocol = u IP 0, jinak specifikace protokolu`



… umíme vytvořit socket ...

Klient: connect()

- Navázání spojení na daném socketu.
- int connect(int sockfd, const struct sockaddr *serv_addr, socklen_t addrlen);

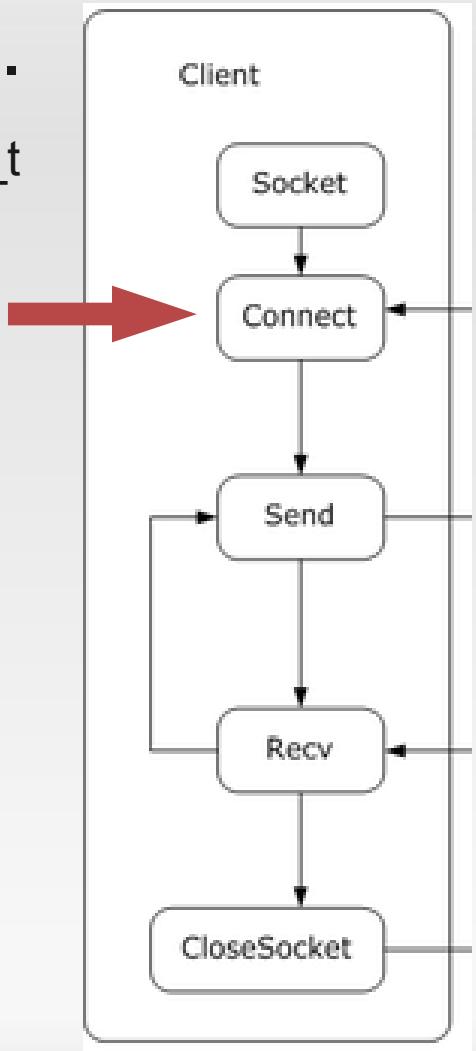
```
struct sockaddr {
    unsigned short    sa_family;    // address family, AF_xxx
    char              sa_data[14];  // 14 bytes of protocol address
};
```

- Ipv4:

```
struct sockaddr_in {
    short            sin_family;   // e.g. AF_INET, AF_INET6
    unsigned short   sin_port;     // e.g. htons(3490)
    struct in_addr   sin_addr;     // see struct in_addr, below
    char             sin_zero[8];  // zero this if you want to
};
```

- Ipv6: sockaddr_in_6

```
struct sockaddr_in6 {
    u_int16_t       sin6_family;   // address family, AF_INET6
    u_int16_t       sin6_port;     // port number, Network Byte Order
    u_int32_t       sin6_flowinfo; // IPv6 flow information
    struct in6_addr sin6_addr;     // IPv6 address
    u_int32_t       sin6_scope_id; // Scope ID
};
```



… umíme se připojit k protistraně ...

Klient: Příklad - připojení

```
if ((sFd = socket (AF_INET, SOCK_STREAM, 0)) == ERROR)
    {
    perror ("socket");
    return (ERROR);
    }

/* bind not required - port number is dynamic */
/* build server socket address */
sockAddrSize = sizeof (struct sockaddr_in);
bzero ((char *) &serverAddr, sockAddrSize);
serverAddr.sin_family = AF_INET;
serverAddr.sin_len = (u_char) sockAddrSize;
serverAddr.sin_port = htons (SERVER_PORT_NUM);

if ((serverAddr.sin_addr.s_addr = gethostbyname (serverName)) == ERROR)
    {
    perror ("unknown server name");
    close (sFd);
    return (ERROR);
    }

/* connect to server */
if (connect (sFd, (struct sockaddr *) &serverAddr, sockAddrSize) == ERROR)
    {
    perror ("connect");
    close (sFd);
    return (ERROR);
    }
```

Klient: send(), recv()

- Čtení/zápis ze soketu
- TCP:

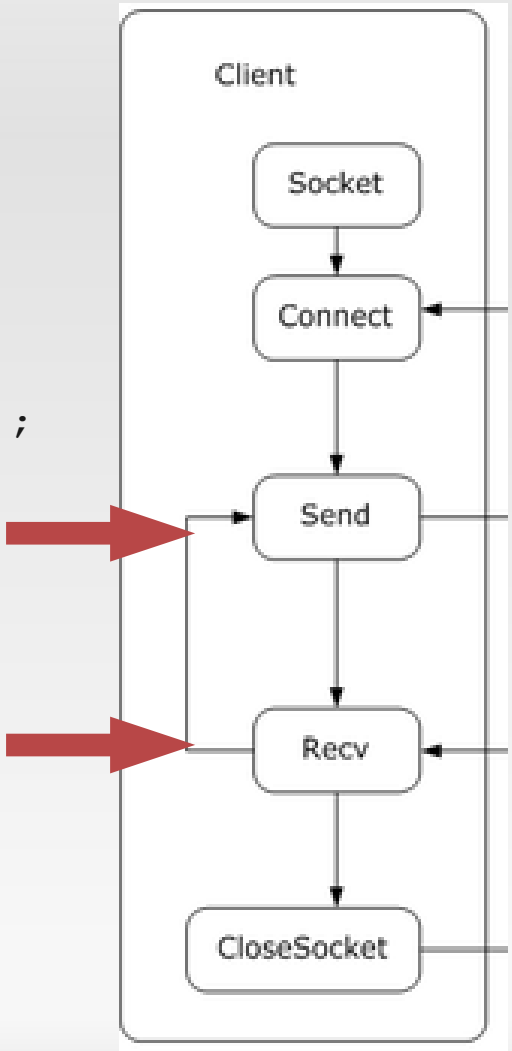
```
ssize_t send(int s, const void *buf, size_t len, int
flags);

ssize_t recv(int s, void *buf, size_t len, int flags);
```

- UDP:

```
sendto

recvfrom
```



… umíme poslat/přijmout data...

Příklad - send/recv

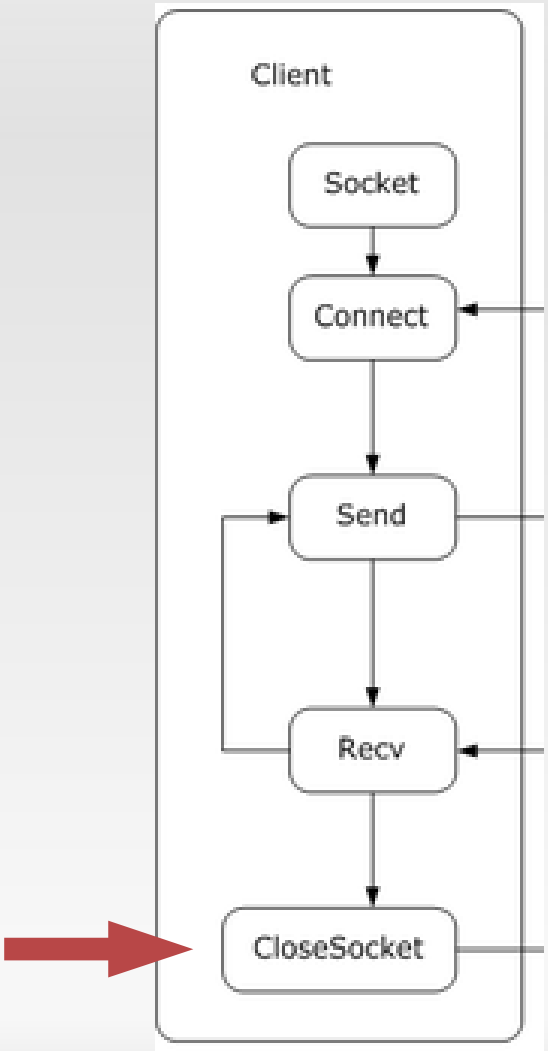
```
/* Send the word to the server */
    echolen = strlen(argv[2]);

    if (send(sock, argv[2], echolen, 0) != echolen) {
           Die("Mismatch in number of sent bytes");
    }

/* Receive the word back from the server */
    fprintf(stdout, "Received: ");
       while (received < echolen) {
          int bytes = 0;
          if ((bytes = recv(sock, buffer, BUFFSIZE-1, 0)) < 1) {
                Die("Failed to receive bytes from server");
          }
      received += bytes;
      fprintf(stdout, buffer);
    }
```

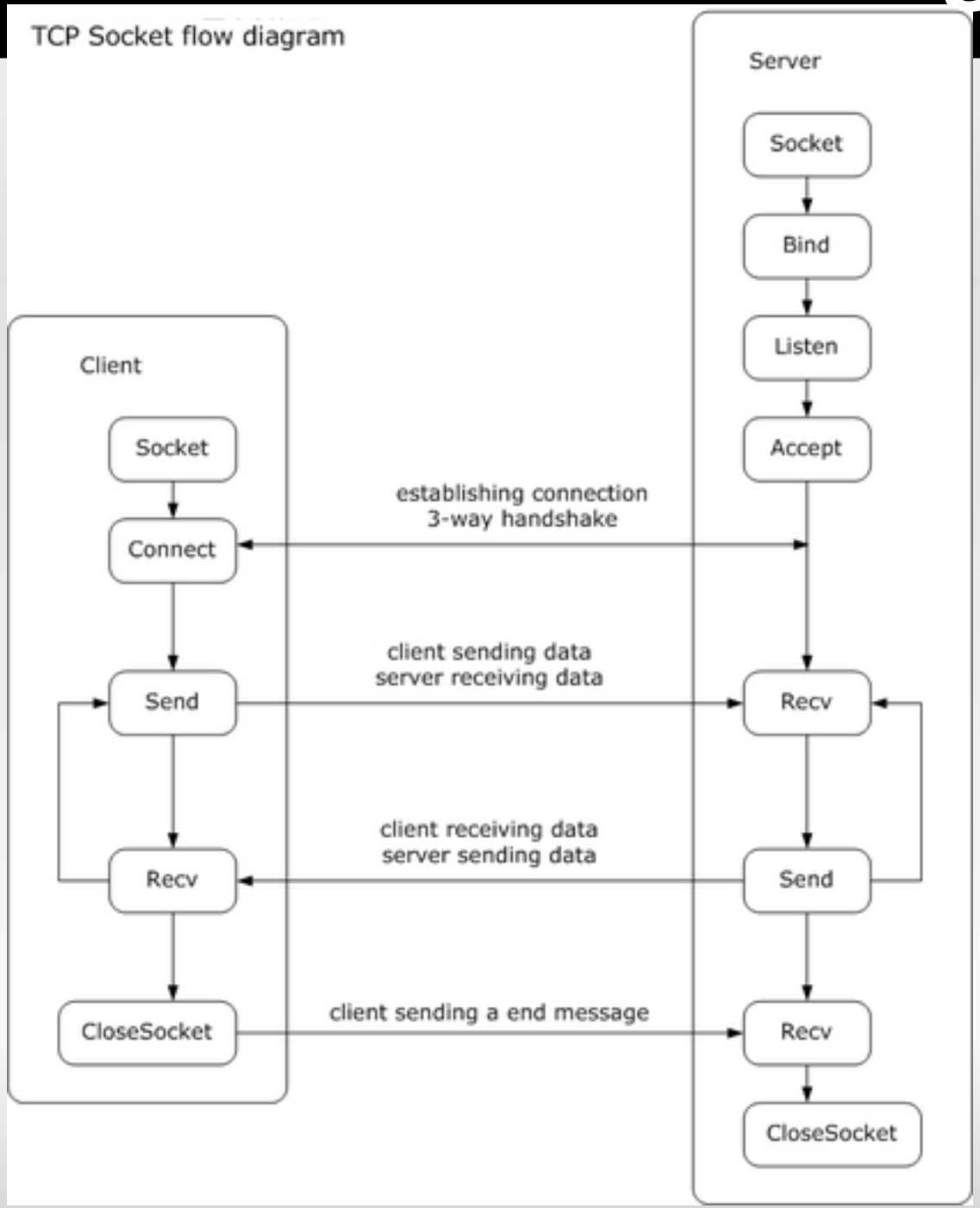
close()

- int close(int fd);
- zavře soket a uklidí po něm
- **Nezapomínat**, zejména u multi-threaded serveru!!!



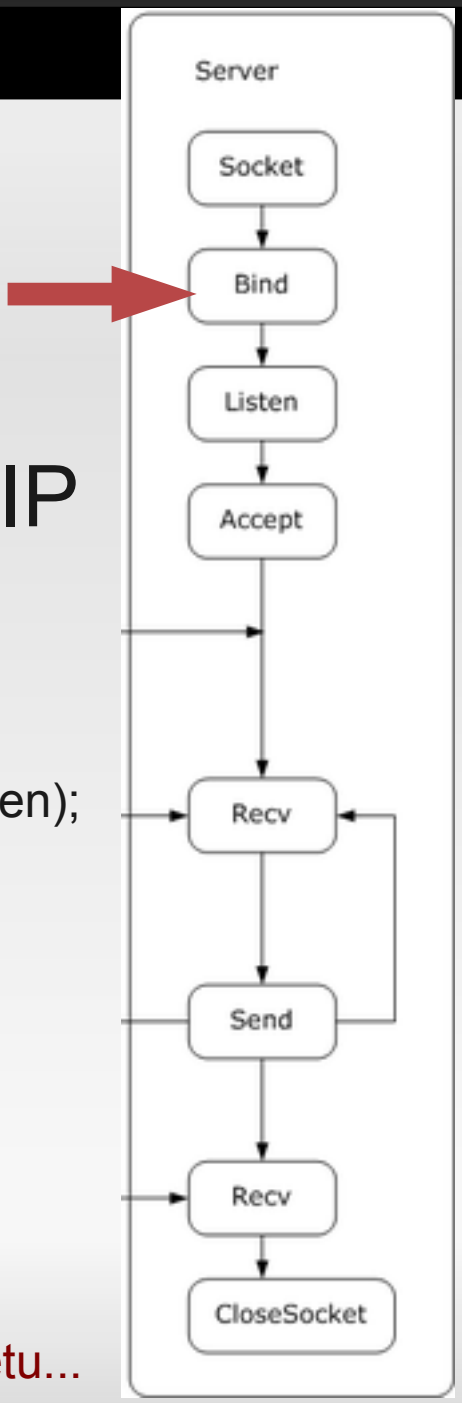
… jsme povinni zavřít soket ...

Opakování: socket flow diagram



Server: bind()

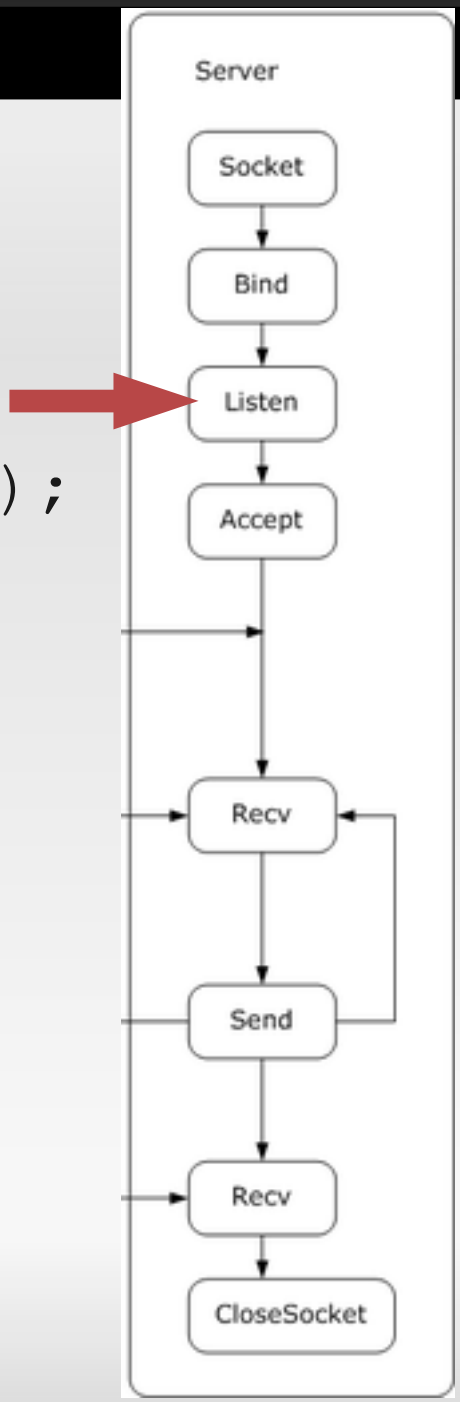
- Volání bind() namapuje adresu k soketu
- Budete používat primárně na spojení IP adresy (pozor – NE hostname) se soketem
- int bind(int sockfd, const struct sockaddr *my_addr, socklen_t addrlen);



… umíme namapovat adresu k soketu...

Server: listen()

- Naslouchej na soketu, zda je nové spojení
- `int listen(int sockfd, int backlog);`
  - `sockfd už znáte`
  - `backlog = délka pro frontu příchozích spojení`



… umíme naslouchat ...

Příklad č. 1 – spuštění serveru

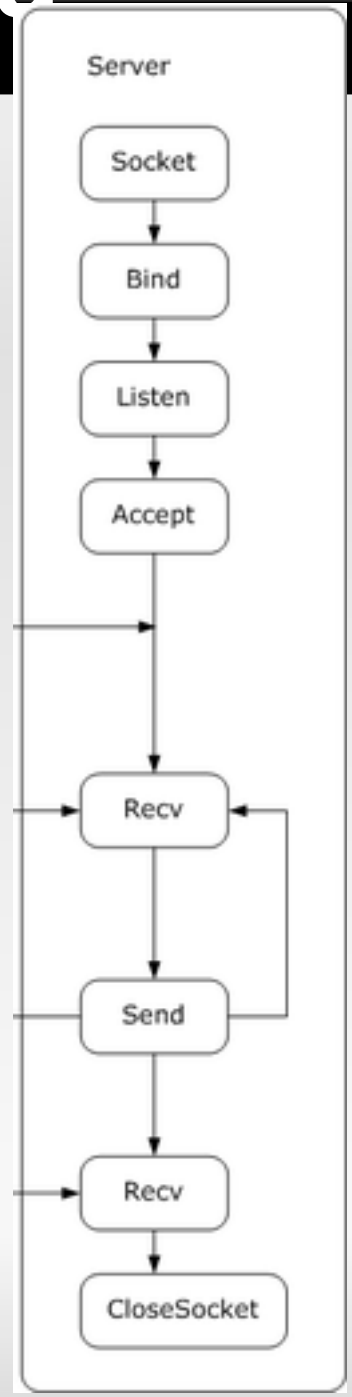
```
listenfd=socket(AF_INET,SOCK_STREAM,0);
 //   bzero(&servaddr,sizeof(servaddr));

servaddr.sin_family = AF_INET;

servaddr.sin_addr.s_addr=htonl(INADDR_ANY);
servaddr.sin_port=htons(32000);

bind(listenfd,(struct sockaddr *)&servaddr,sizeof(servaddr));

listen(listenfd,1024);
```



Vložka č. 1 – Network byte order

- Big endian vs. Little endian
- Network byte order – BIG-endian (definuje IP)
- Host byte order – host = váš hostitelský OS
- Je zavedena sada funkcí pro portovatelnost programů

Funkce pro převod NBO na HBO

- The **htonl**() function converts the unsigned integer hostlong from host byte order to network byte order.
- The **htons**() function converts the unsigned short integer hostshort from host byte order to network byte order.
- The **ntohl**() function converts the unsigned integer netlong from network byte order to host byte order.
- The **ntohs**() function converts the unsigned short integer netshort from network byte order to host byte order.

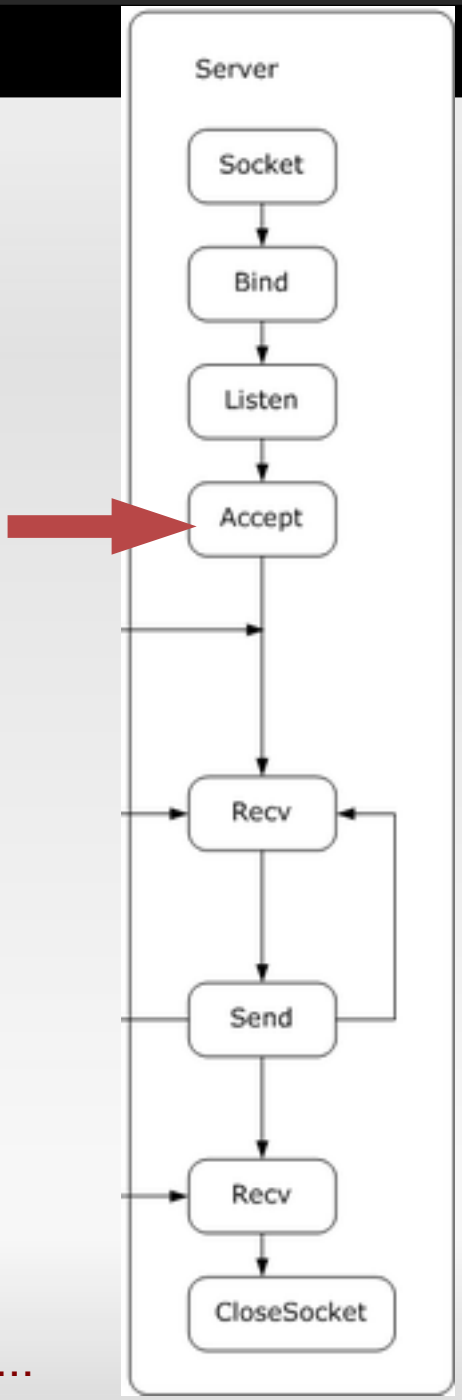
accept()

- přijmi spojení na daném soketu

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

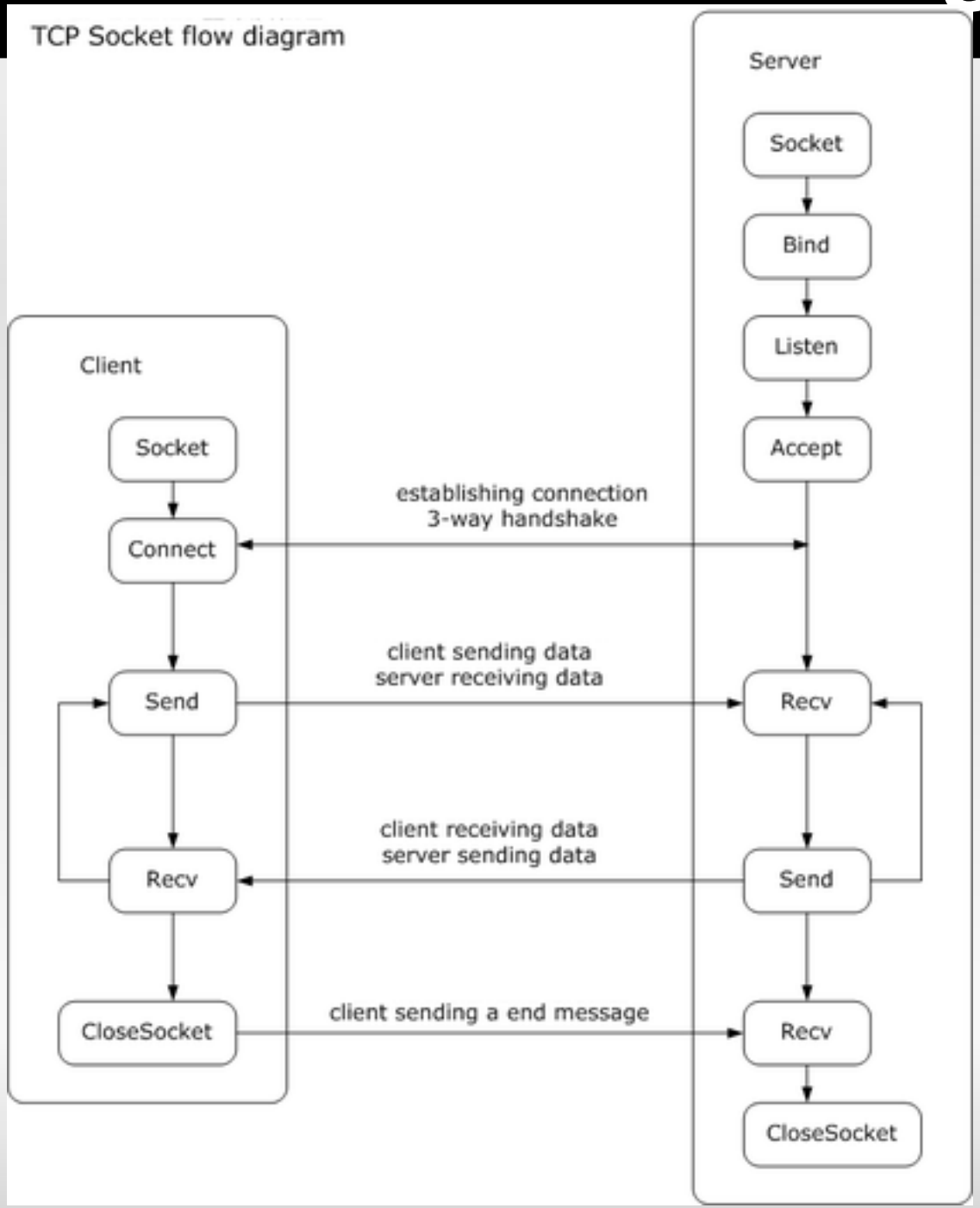
- Vrátí **NOVÝ** soket!
- Pokračování příkladu:

```
clilen=sizeof(cliaddr);
connfd = accept(listenfd,(struct sockaddr *)&cliaddr,&clilen);
```

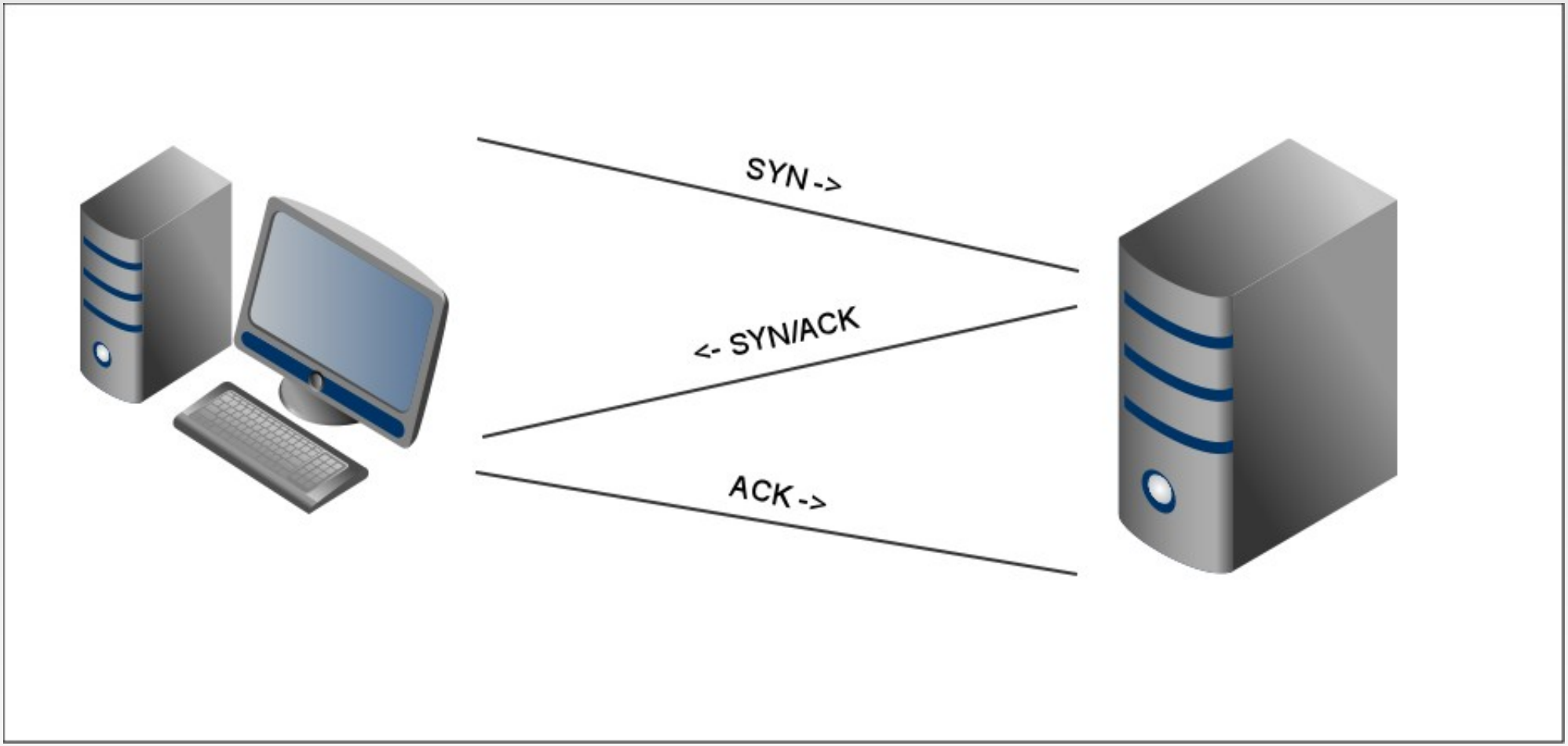


… umíme přijmout spojení ...

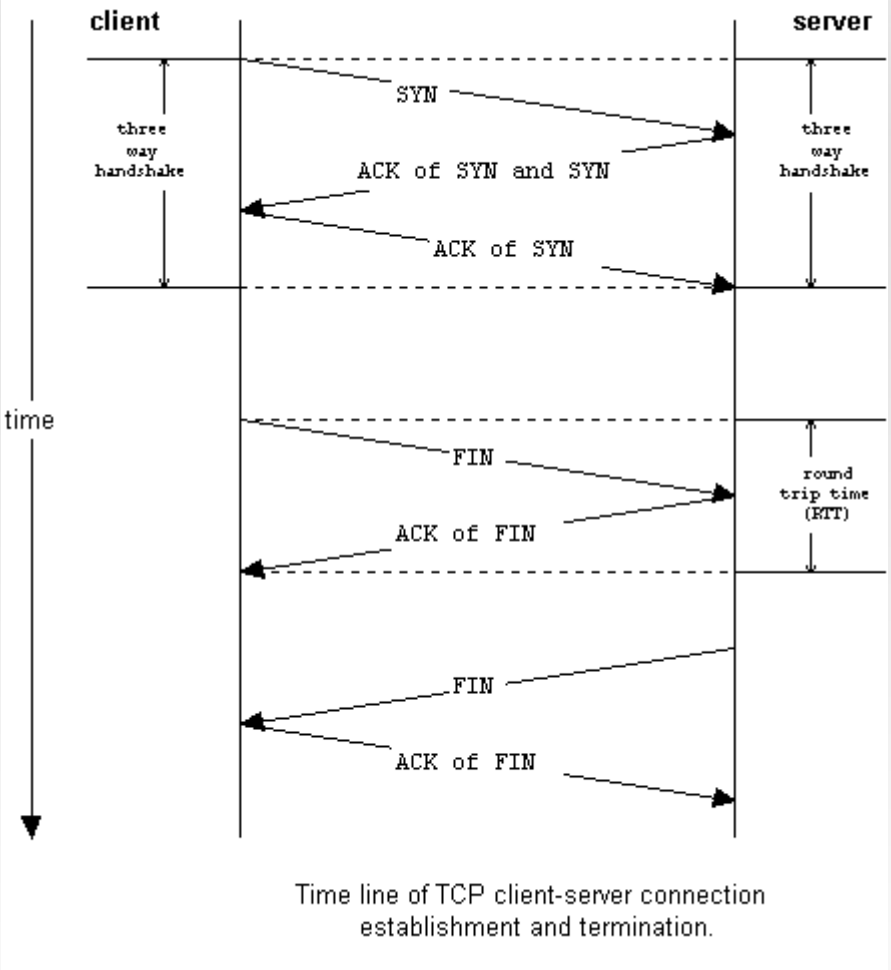
Opakování: socket flow diagram



TCP three-way handshake

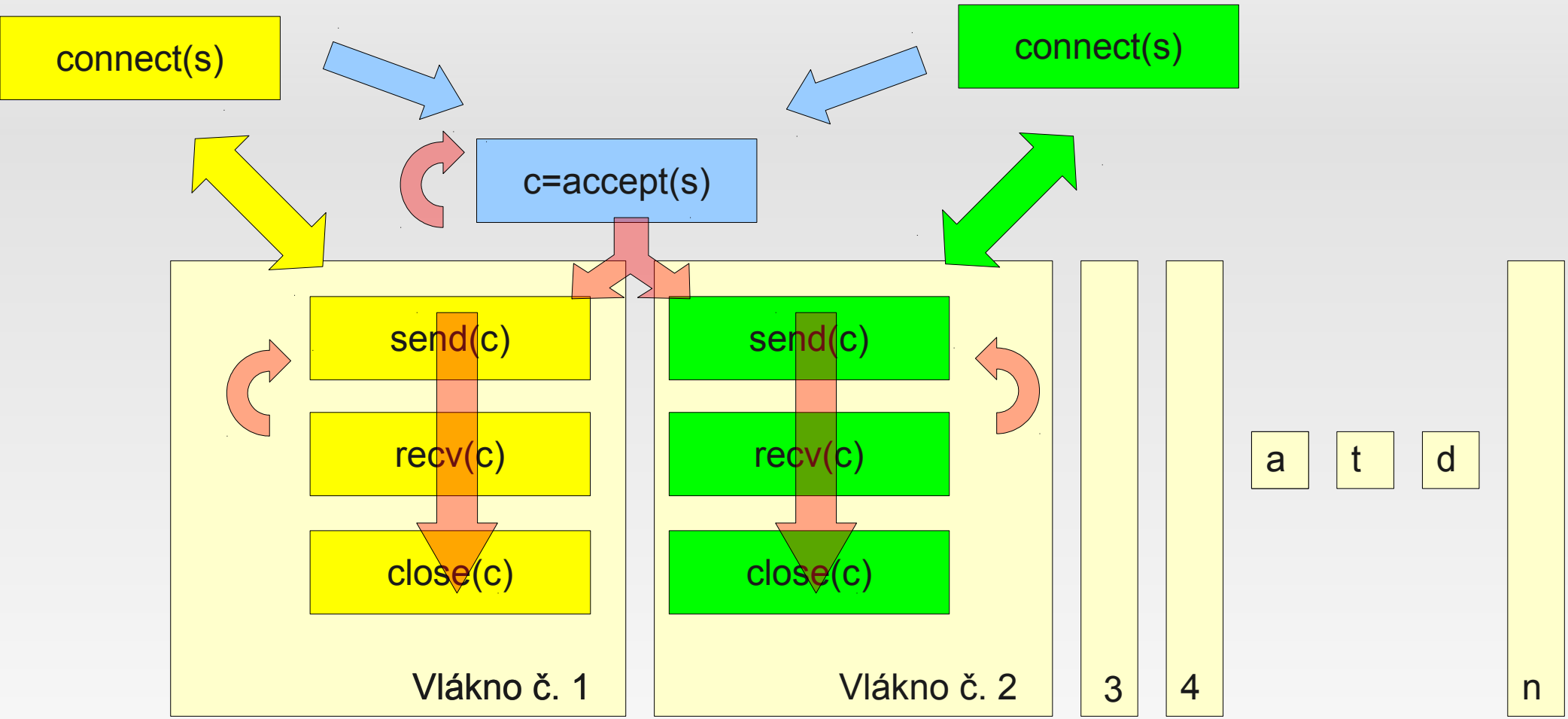


Oboustranné ukončení spojení



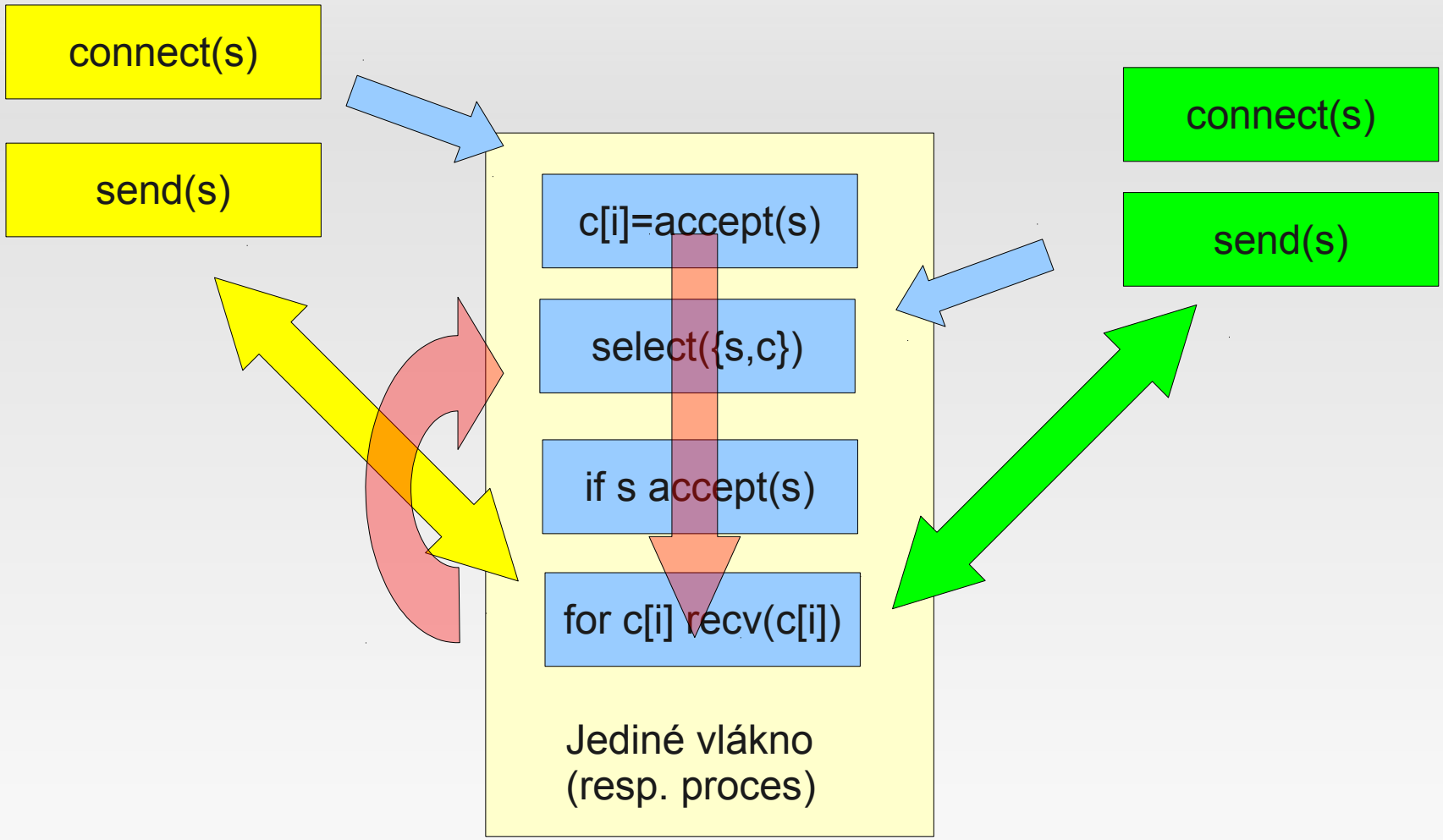
Time line of TCP client-server connection establishment and termination.

Souběžné zpracování požadavků



- Pozor – vytvoření vlákna v OS ”trvá”
- Pro aplikace s velkým množstvím současných spojení – různé metody alokace threadů, např. thread pool

TCP – souběžné zpracování bez vláken



select()/poll()

- Volání select() sleduje I/O operace na vícero deskriptorech
- Změna na deskriptoru vyvolá návrat z funkce
- Poll() - obdobná funkčnost
- Rozdíl je v předávaných parametrech, v různých implementacích v různých systémech

Vaše možnosti

- fork()
- select()/poll()
- Thready – doporučeno – nepotřebujete žádnou synchronizaci, jde pouze o volbu knihovny a volání správných metod
  - Např. Knihovna pthreads