# Síťové aplikace (sockety)

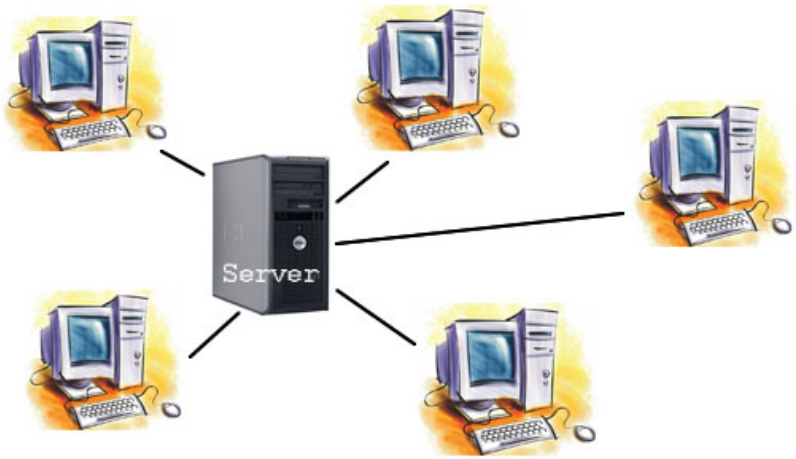
## Tématicky zaměřený vývoj aplikací v jazyce C
## skupina Systémové programování – Linux

Martin Husák, Petr Velan, Martin Drašar

Fakulta informatiky
Masarykova univerzita
husakm@ics.muni.cz

Brno, 23. listopadu 2015

# Klient-server model komunikace




- Klient žádá o služby server, který je poskytuje.
- Klient i server jsou aplikace, které běží na stejném nebo různých počítačích.
- Tento model využívá většina dnešních protokolů a aplikací.

# Sockety

síťová komunikace

# Sockety – teorie

- Tak jako skoro všechno v Linuxu (UNIXu), socket je vlastně soubor – má svůj file descriptor a programy do něho zapisují nebo z něho čtou.
- Místo `open()` se používá systémové volání `socket()`.
- Místo `read()`, `write()` je lepší používat `send()` a `recv()`.
- Komunikující procesy nemusí být na stejném počítači.

**Typy socketů**

- Internet sockets
- UNIX sockets
- ...

# Byte order

- Little endian vs. Big endian
- Data na síti jsou vždycky v Network-Byte-Order (Big endian)
- Funkce pro převod mezi Network-Byte-Order a Host-Byte-Order:

```
#include <arpa/inet.h>

uint32_t htonl(uint32_t hostlong);
uint16_t htons(uint16_t hostshort);
uint32_t ntohl(uint32_t netlong);
uint16_t ntohs(uint16_t netshort);
```

# Příprava spojení – getaddrinfo()

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

int getaddrinfo(const char *node, const char *service,
                const struct addrinfo *hints,
                struct addrinfo **res);
```

**node** adresa cíle spojení

- doménové jméno
- adresa v tečkové notaci (IPv4)
- adresa v hexa formátu (IPv6)

**service** číslo portu/název služby podle `/etc/services`

# Struktury addrinfo a sockaddr

```
struct addrinfo {
    int              ai_flags;
    int              ai_family;
    int              ai_socktype;
    int              ai_protocol;
    size_t           ai_addrlen;
    struct sockaddr *ai_addr;
    char            *ai_canonname;
    struct addrinfo *ai_next;
};
```

# Struktury addrinfo a sockaddr

```
struct addrinfo {
    int              ai_flags;
    int              ai_family;
    int              ai_socktype;
    int              ai_protocol;
    size_t           ai_addrlen;
    struct sockaddr *ai_addr;
    char            *ai_canonname;
    struct addrinfo *ai_next;
};

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

struct sockaddr_storage {
    sa_family_t     ss_family;   // address family
    // padding - something big enough to store both IPv4 and IPv6
};
```

# Struktury pro IPv4

```
struct sockaddr_in {
    short int          sin_family;  // Address family, AF_INET
    unsigned short     int sin_port;// Port number, Network Byte Order
    struct in_addr     sin_addr;    // Internet address
    unsigned char      sin_zero[8]; // Same size as struct sockaddr
};
```

```
struct in_addr {
    uint32_t           s_addr;       // 32-bit int, Network Byte Order
};
```

# Struktury pro IPv6

```
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
};
```

```
struct in6_addr {
    unsigned char   s6_addr[16];   // IPv6 address, Network Byte Order
};
```

`IN6ADDR_*` konstanty jsou narozdíl od IPv4 konstant v Network Byte Order.

## úkol

- Napište vlastní verzi programu `resolveip`.
- Nápověda: využijte funkci `inet_ntop()`
- Stačí převod doménové jméno → adresa (IPv4 i IPv6).

# Vytvoření socketu – socket()

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

Cílem je získat filedescriptor socketu pro použití v dalších funkcích.

**domain** – jmenný prostor specifikuje způsob adresace socketů v rámci rodiny protokolů – konstanty mají předponu `PF_`[1], jmenný prostor omezuje množinu protokolů

- `PF_UNIX`, `PF_UNSPEC`
- `PF_INET`, `PF_INET6`
- ...

**type** – styl spojení (spojované/nespojované)

- `SOCK_STREAM`, `SOCK_DGRAM`, `SOCK_RAW`
- ...

**protocol** – jeden z množiny možných protokolů

[1] používá se i `AF_` (address family) pro `getaddrinfo()`

# Připojení – connect()

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

Jako parametry se používají filedescriptor socketu (získaný pomocí `socket()`) a hodnoty vrácené funkcí `getaddrinfo()`.
Chování u spojované a nespojované komunikaci se lehce liší.

# Rezervace portu – bind()

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

Slouží k asociaci vytvořeného socketu s portem – pamatujte, že porty $< 1024$ jsou rezervovány a smí je používat pouze root. Nemůžete se ani spoléhat na to, že vámi vybraný port je volný.

Pomocí `getaddrinfo()` tentokrát musíte získat informace o sobě:

```
memset(&hints, 0, sizeof(hints));
hints.ai_family = AF_UNSPEC; // use IPv4 or IPv6, whichever
hints.ai_flags = AI_PASSIVE; // fill in my IP for me
getaddrinfo(NULL, "8080", &hints, &res);
```

# Naslouchání – listen()

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

Pouze pro spojovanou komunikaci, tedy typy `SOCK_STREAM` a `SOCK_SEQPACKET`.

`backlog` specifikuje délku fronty, ve které se shromažďují požadavky na připojení.

# Přijetí spojení – accept()

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

Opět pouze spojovaná komunikace.
Vezme první spojení z fronty čekajících a vytvoří **nový** socket.
Původní socket stále zůstává ve stavu `listen`.
V 2. a 3. parametru vrací `accept()` informace o druhé straně spojení. `addrlen` musíte inicializovat na počet bytů alokovaných pro `*addr` – `accept()` tam víc dat nezapíše.
Až v této chvíli lze do socketu zapisovat a číst z něho.

# Vzájemná komunikace

## Spojovaná komunikace

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

## Nespojovaná komunikace

```
int sendto(int s, const void *msg, int len, unsigned int flags,
        const struct sockaddr *to, int tolen);
ssize_t recvfrom(int sockfd, void *buf, size_t len, int flags,
        struct sockaddr *src_addr, socklen_t *addrlen);
```

Kontrola skutečně odeslaných dat je na vás!

`sockaddr` v `recvfrom()` by ve skutečnosti měla být `sockaddr_storage` a `addrlen` inicializována na `sizeof(struct sockaddr_storage)`

# Uzavření spojení – close() a shutdown()

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

#include <sys/socket.h>
int shutdown(int s, int how);
```

`close()` funguje stejně jako na jakýkoli jiný filedescriptor.
`shutdown()` umožňuje částečně omezit provoz socketu (příjem|odesílání). Pro uvolnění filedescriptoru je nakonec třeba zavolat `close()`.

# Shrnutí - klient

1. specifikace cíle spojení a `getaddrinfo()`
2. `socket()`
3. (`bind()` – rezervace lokálního portu)
4. `connect()`
   - spojovaná komunikace
   - filtrovaná nespojovaná komunikace
5. `send()` | `sendto()`
6. `recv()` | `recvfrom()`
7. `close()`

# Shrnutí - server

1. `getaddrinfo()` – informace o serveru
2. `socket()`
3. `bind()` – rezervace lokálního portu
4. (`connect()` – filtr nespojované komunikace)
5. `listen()` – pouze spojovaná komunikace
6. `accept()` – pouze spojovaná komunikace
7. `recv()` | `recvfrom()`
8. `send()` | `sendto()`
9. `close()`

# Další zajímavé funkce

- `getnameinfo()`
- `getpeername()`
- `gethostname()`
- `gethostbyname()`
- `gethostbyaddr()`

## Závěr

Projekt a zdroje

# Projekt

**Komunikace tankclient–tank**

- Implementujte sokety pro tank (server) a tankclient (klient)
- Tankclient bude tanku posílat příkazy přes UDP
- Posílejte stejné příkazy jako u komunikace server–tank
- U komunikace server-tank přidejte prázdný příkaz 'no'
- Prázdný příkaz použijte, pokud tanku nedošly žádné povely
- Příkaz který tank akceptoval pošle zpět klientovi jako odpověď
- Zamyslete se nad způsobem bufferování příkazů z klienta

# Zdroje

- beej.us/guide/bgnet/