Síťové aplikace (sockety)

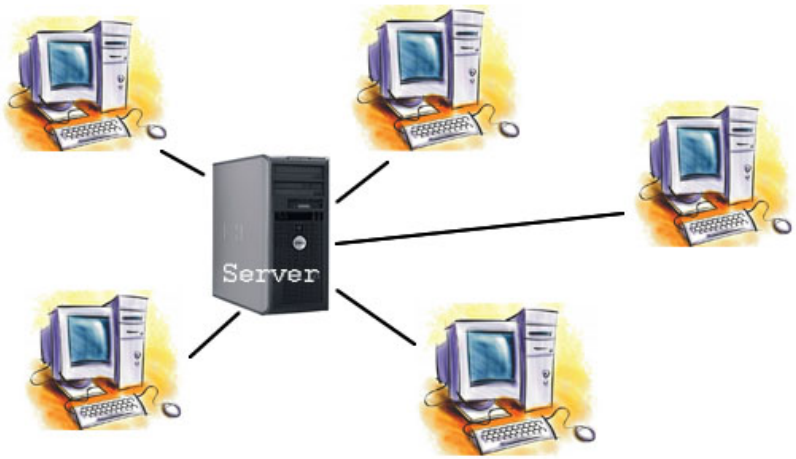
Tématicky zaměřený vývoj aplikací v jazyce C
skupina Systémové programování – Linux

Jiří Novosad

Fakulta informatiky
Masarykova univerzita
novosad@fi.muni.cz

Brno, 11. 11. 2014

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 naprostá 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);
void freeaddrinfo(struct addrinfo *res);

const char *gai_strerror(int errcode);
```

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;    // AF_{INET,INET6,UNIX,UNSPEC}
  int              ai_socktype;  // SOCK_{DGRAM,RAW,STREAM,SEQPACKET}
  int              ai_protocol;  // IPPROTO_{IP,IPV6,TCP,UDP,ICMP,RAW}
  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;    // AF_{INET,INET6,UNIX,UNSPEC}
  int              ai_socktype;  // SOCK_{DGRAM,RAW,STREAM,SEQPACKET}
  int              ai_protocol;  // IPPROTO_{IP,IPV6,TCP,UDP,ICMP,RAW}
  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
};
```

Překlad adresy na jméno/řetězec – getnameinfo()

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

int getnameinfo(const struct sockaddr *sa, socklen_t salen,
                char *host, socklen_t hostlen,
                char *serv, socklen_t servlen, int flags);
```

- **sa** adresa cíle spojení (např. z `getaddrinfo()` `h->ai_addr`)
- **salen** délka adresy (např. `h->ai_addrlen`)
- **host** buffer pro doménové jméno / IP adresu
- **serv** buffer pro jméno / číslo služby
- **flags** přepínače
  - `NI_NUMERICHOST`, `NI_NUMERICSERV` – číselná adresa / služba
  - ...
- **hostlen** délka bufferu `host`, pro `NI_NUMERICHOST` lze použít `INET6_ADDRSTRLEN`/`INET_ADDRSTRLEN`

úkol

- Napište vlastní verzi programu `resolveip`.
- Stačí převod doménové jméno → adresa/y (IPv4 i IPv6).
- Nápověda: pro převod adresy na řetězec použijte funkci `getnameinfo()`.
- Za domácí úkol program rozšiřte o získání doménového jména pro každou IP adresu, využijte opět `getnameinfo()`.

Vytvoření socketu – socket()

```
#include <sys/types.h>
#include <sys/socket.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 `AF_`[1], jmenný prostor omezuje množinu protokolů

- `AF_INET`, `AF_INET6`
- `AF_UNIX`
- ...

type – styl spojení (spojované/nespojované)

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

protocol – jeden z množiny možných protokolů, většinout pro danou `domain` a `type` jen jeden → 0

[1] používá se i `PF_` (protocol family)

Připojení – connect()

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

Jako parametry se používá filedescriptor socketu (získaný pomocí `socket()`) a hodnoty vrácené funkcí `getaddrinfo()`.

Nespojovaná komunikace – jen nastaví default adresu.

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 adresou a 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);
```

- Označí socket pro akceptování příchozích spojení.
- 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.
- Do výsledného socketu lze nyní 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, socklen_t 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

- `gethostname()` – adresa (hostname) stroje, kde běží proces
- `getpeername()` – adresa komunikujícího partnera (`struct sockaddr`)
- `getsockname()` – adresa, na kterou je nabindovaný socket (`struct sockaddr`)
- `gethostbyname()`, `gethostbyaddr()`, `inet_ntop()`, `inet_pton()`, ... – nepoužívat, nahrazeno `getaddrinfo()`, `getnameinfo()`

Závěr

domácí úkoly a zdroje

Domácí úkol

Dropbox – 2. část

- Program z minulé hodiny upravte tak, aby pracoval jako centrální synchronizační server.
- Server zajišťuje synchronizaci všech připojených klientů.
- Argumenty: `-s`, číslo portu, adresář se soubory
- `./dropbox -s 3210 fromdir/`
- Klient po připojení nakopíruje soubory do adresáře `todir/`, pak průběžně získává nová data od serveru tak, jak se na něm mění obsah adresáře `fromdir/`
- Argumenty: `-c`, adresa serveru, port na serveru, cílový adresář.
- `./dropbox -c server.address.com 3210 todir/`
- V dokumentaci popište formát komunikace.

Zdroje

- beej.us/guide/bgnet/
- www.kame.net/newsletter/19980604/