Síťové aplikace (sockety)

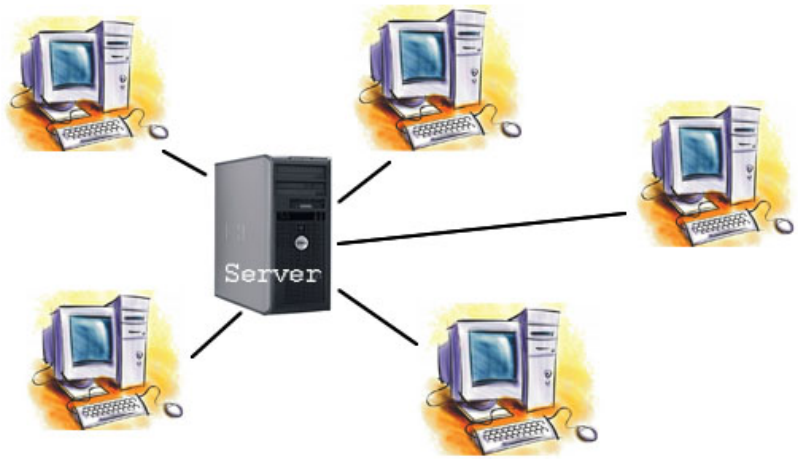
Tématicky zaměřený vývoj aplikací v jazyce C
skupina Systémové programování – Linux

Jiří Novosad

Fakulta informatiky
Masarykova univerzita
novosad@fi.muni.cz

Brno, 25. 11. 2016

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 se tváří jako 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);[1]
```

node adresa cíle spojení

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

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

[1] `EAI_SYSTEM`

Struktury addrinfo a sockaddr

```
struct addrinfo {
  int              ai_flags;
  int              ai_family;    // AF_{INET,INET6,UNIX,UNSPEC}
  int              ai_socktype;  // SOCK_{STREAM,DGRAM,RAW,SEQPACKET}
  int              ai_protocol;  // IPPROTO_{IP,IPV6,TCP,UDP,ICMP,RAW}
  socklen_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_{STREAM,DGRAM,RAW,SEQPACKET}
  int              ai_protocol;  // IPPROTO_{IP,IPV6,TCP,UDP,ICMP,RAW}
  socklen_t        ai_addrlen;
  struct sockaddr *ai_addr;
  char            *ai_canonname;
  struct addrinfo *ai_next;
};

struct sockaddr {
    sa_family_t   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 {
    sa_family_t     sin_family;  // Address family, AF_INET
    in_port_t       sin_port;    // Port number (NBO, 16-bit)
    struct in_addr  sin_addr;    // IPv4 address
    unsigned char   sin_zero[8]; // Same size as struct sockaddr
};
```

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

Struktury pro IPv6

```
struct sockaddr_in6 {
    sa_family_t     sin6_family;   // address family, AF_INET6
    in_port_t       sin6_port;     // port number, Network Byte Order
    uint32_t        sin6_flowinfo; // IPv6 flow information
    struct in6_addr sin6_addr;     // IPv6 address
    uint32_t        sin6_scope_id; // Scope ID
};
```

```
struct in6_addr {
    uint8_t         s6_addr[16];   // IPv6 address
};
```

Překlad adresy na jméno/řetězec – getnameinfo()

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

int getnameinfo(const struct sockaddr *sa, socklen_t salen,
                char *host, size_t hostlen,
                char *serv, size_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`, jinak `NI_MAXHOST`, `NI_MAXSERV` (viz `getnameinfo(3)`)

úkol

- Napište vlastní verzi programu `resolveip`.
- Jediným argumentem je doménové jméno, výstupem jsou odpovídající adresy (IPv4 i IPv6).
- Nápověda: pro převod adresy na řetězec použijte funkci `getnameinfo()`.

úkol

- Napište vlastní verzi programu `resolveip`.
- Jediným argumentem je doménové jméno, výstupem jsou odpovídající adresy (IPv4 i IPv6).
- Nápověda: pro převod adresy na řetězec použijte funkci `getnameinfo()`.
- Za domácí úkol program rozšiřte o získání doménového jména pro každou výstupní 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_`[2], jmenný prostor omezuje množinu protokolů

- `AF_INET`, `AF_INET6`
- `AF_UNIX`, ...
- `addrinfo.ai_family`

type – styl spojení (spojované/nespojované)

- `SOCK_STREAM`, `SOCK_DGRAM`, `SOCK_SEQPACKET`, ...
- `addrinfo.ai_socktype`

protocol – `addrinfo.ai_protocol` nebo 0 (většinou pro danou `domain` a `type` jen jeden možný)

[2] 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()` – `addrinfo.ai_addr`, `addrinfo.ai_addrlen`.

Nespojovaná komunikace (UDP) – jen nastaví implicitní adresu.

úkol

- Napište program `time`, který implementuje RFC 868 klienta (TCP).
- Jediným argumentem programu je time server – hostname nebo IPv4/v6 adresa (`time.fi.muni.cz`).
- `$ ./time time.fi.muni.cz`

  `Fri Nov 25 10:24:42 2016`
- Nápověda: `ctime()`, `ntohl()`

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);
```

`getaddrinfo(3)` může vrátit více adres (např. IPv4 a IPv6).

Naslouchání – listen()

```
#include <sys/socket.h>
int listen(int sockfd, 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>
#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
- Vhodné použití `struct sockaddr_storage`.
- Do výsledného socketu lze nyní zapisovat a číst z něho.

Vzájemná komunikace

Spojovaná komunikace

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

Nespojovaná komunikace

```
ssize_t sendto(int sockfd, const void *buf, size_t len, int flags,
    const struct sockaddr *dest_addr, socklen_t addrlen);

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 sockfd, 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()`.

Uzavření spojení – send(), recv(), ...

- Co se stane, když druhá strana uzavře spojení?
- Chová se stejně, jako uzavřená roura.

Uzavření spojení – send(), recv(), . . .

- Co se stane, když druhá strana uzavře spojení?
- Chová se stejně, jako uzavřená roura.
- `read()` / `recv()` vrací 0

Uzavření spojení – send(), recv(), . . .

- Co se stane, když druhá strana uzavře spojení?
- Chová se stejně, jako uzavřená roura.
- `read()` / `recv()` vrací 0
- `write()` / `send()`: `SIGPIPE` (!), vrací -1, `errno == EPIPE`
- → `SIG_IGN`, ale ovlivní celý proces, všechny knihovny, . . .
- → nebo flag `MSG_NOSIGNAL`

Shrnutí - klient

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

Shrnutí - server

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

Použití se `select()`, `poll()`, ...

- sockety můžeme používat ve volání `select()` jako jiné deskriptory
- socket, který naslouchá příchozím spojením, bude při pokusu klienta o spojení připraven pro čtení
- pak můžeme volat `accept()`
- pokud klient pokus o spojení zruší, může `accept()` blokovat → `O_NONBLOCK`, `EAGAIN`

Vlastnosti socketu I

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

int getsockopt(int sockfd, int level, int optname,
    void *optval, socklen_t *optlen);
int setsockopt(int sockfd, int level, int optname,
    const void *optval, socklen_t optlen);
```

- SO_REUSEADDR (SOL_SOCKET)
- IPV6_V6ONLY (IPPROTO_IPV6), /proc/sys/net/ipv6/bindv6only
- setsockopt(sockfd, SOL_SOCKET, SO_REUSEADDR, &(socklen_t){ 1 }, sizeof(socklen_t))
- socket(7), ip(7)

Vlastnosti socketu II

Any sane library _must_ have "socklen_t"be the same size as int. Anything else breaks any BSD socket layer stuff. POSIX initially did make it a size_t, and I (and hopefully others, but obviously not too many) complained to them very loudly indeed. Making it a size_t is completely broken, exactly because size_t very seldom is the same size as "int"on 64-bit architectures, for example. And it has to be the same size as "int"because that's what the BSD socket interface is. Anyway, the POSIX people eventually got a clue, and created "socklen_t". They shouldn't have touched it in the first place, but once they did they felt it had to have a named type for some unfathomable reason (probably somebody didn't like losing face over having done the original stupid thing, so they silently just renamed their blunder). –Linus Torvalds

Další zajímavé funkce

užitečné

- `gethostname()` – adresa (hostname) stroje, kde běží proces
- `getpeername()` – adresa komunikujícího partnera (`struct sockaddr`)
- `getsockname()` – adresa, na kterou je nabindovaný socket (`struct sockaddr`)

neužitečné

- `gethostbyname()`, `gethostbyaddr()`, ... – nepoužívat, nahrazeno `getaddrinfo()`, `getnameinfo()`
- `inet_ntoa()`, `inet_aton()` – nepoužívat, nepodporuje IPv6, nahrazeno:
- `inet_ntop()`, `inet_pton()` – závislé na protokolu, používejte `getaddrinfo()`, `getnameinfo()`

Závěr

domácí úkoly a zdroje

Domácí úkol I

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 ze serveru 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/`
- Server i klient se ukončí, jakmile zaznamenají nějaký vstup na stdin.

Domácí úkol II

- Musí fungovat nad IPv4 i IPv6 – server přijímá a klient umí navázat oba typy spojení.
- V dokumentaci popište formát komunikace.
- Termín odevzdání posunuji o jeden týden, tedy na **9. 12. 2016** ve 23:59. Další (desátá, poslední) úloha bude mít stejný termín odevzdání.

Zdroje

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