PRÍLOHA A

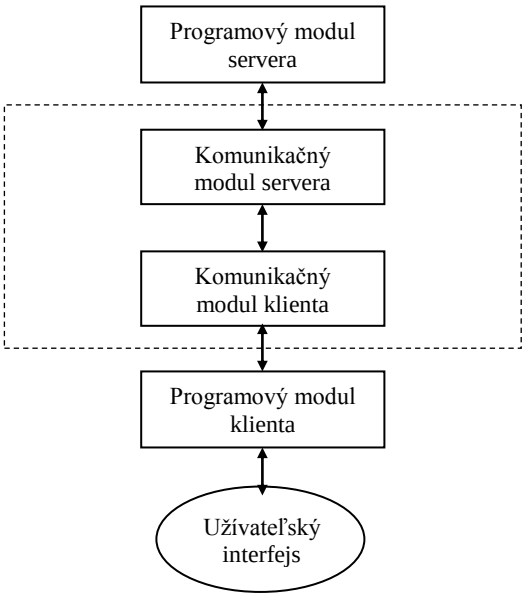
KOMUNIKÁCIA V SIETI – SCHRÁNKY

Komunikačné prostriedky, uvedené v kapitole o komunikácií medzi procesmi sú vhodné pre komunikáciu medzi procesmi v jednom systéme. V prípade potreby komunikácie medzi procesmi, bežiacich na rôznych uzloch siete sa používajú sieťové komunikačné prostriedky. V tejto prílohe uvedieme schránky (angl. *socket*).

Požiadavkou pre zvládnutie problematiky komunikácie v sietí je základná znalosť komunikačných protokolov a ich fungovanie.

A.1 Schránky (sockets)

Veľká časť komunikácií medzi procesmi využíva model klient-server. V tomto modeli jeden z procesov je klientom, ktorý požaduje nejakú službu a druhý proces je serverom, ktorý tu službu poskytuje. Model takejto komunikácie je ukázaný na obrázku A.1.



Obrázok A.1: Model klient-server

Pred započatím komunikácie je nutne, aby klient vedel adresu servera, zatiaľ čo server nepotrebuje vedieť adresu klienta. Po ustanovení spojenia obidva procesy môžu posielať a dostávať dáta.

Schránka (angl. *socket*) je koncový bod spojenia medzi dvomi komunikujúcimi procesmi. Každý z účastníkov spojenia vytvára svoju schránku.

Kroky, ktoré vykoná **klient**, aby nadviazal spojenie so serverom sú:

1. vytvorí schránku pomocou systémového volania **`socket()`**,
2. spoji schránku s adresou servera pomocou systémového volania **`connect()`**,
3. posiela a dostáva dáta. Existuje viacero spôsobov ako to vykonať, ale najjednoduchším spôsobom je použitie systémových volaní **`read()`** a **`write()`**.

Kroky, ktoré vykoná **server**, aby nadviazal spojenie s klientom sú:

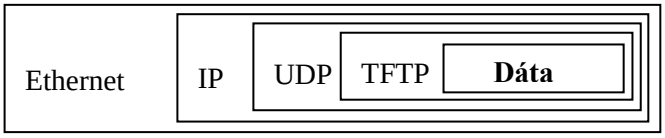
1. vytvorí schránku pomocou systémového volania **`socket()`**,
2. spoji schránku s adresou pomocou systémového volania **`bind()`**. Pre schránku servera v Internete adresa pozostáva z čísla portu na hostiteľskom počítači.
3. čaká pre spojenie pomocou systémového volania **`listen()`**,
4. akceptuje spojenie pomocou systémového volania **`accept()`**. Toto volanie je obyčajne blokujúce kým sa klient spoji so serverom,
5. dostáva a posiela dáta.

Existujú dva typy schránok - **prúdové** (stream socket) a **datagramové** (datagram socket). Prúdové schránky vykonávajú komunikáciu ako nepretržitý prúd znakov (napr. prúd znakov z klávesnice). Datagramové schránky čítajú naraz celú správu (napr. posielanie stránok dokumentu na tlačiareň). Každý typ schránky využíva svoj protokol. Prúdové schránky využívajú TCP (Transmission Control Protocol), ktorý je spoľahlivý, prúdovo orientovaný protokol. Datagramové schránky využívajú UDP (Unix Datagram Protocol), ktorý je nespoľahlivý a správovo orientovaný.

Príprava dát prebieha nasledovným spôsobom: najskôr k paketu dát je pridaná hlavička prvého protokolu (napr. TFTP), potom tento celok je „obalený“ ďalším protokolom (napr. UDP), potom sa znova „obalí“ ďalším protokolom (napr. IP) a nakoniec sa „obalí“ posledným protokolom na hardvérovej úrovni (fyzikálna vrstva) napr. protokolom Ethernet.

Po doručení paketu, hardvér oddelí hlavičku Ethernetu, jadro OS oddelí IP a UDP hlavičky, TFTP program oddelí TFTP hlavičku a dáta sú v pôvodnom tvare. Zapuzdrenie dát je ukázané na obrázku A.1.

Pri použití schránok v programe, programátor sa nemusí starať o fyzický prenos dát, ten je zabezpečený komunikačným podsystémom.



Obrázok A.2 : Zapuzdrenie dát

A.1.1 Vytvorenie schránky - systémové volanie `socket()`

Systémové volanie `socket()` vytvára koncové spojenie pre komunikáciu medzi dvomi procesmi. Princíp tvorby a využitie schránky je v mnohom podobný práce so súbormi – po úspešnom volaní sa vráti deskriptor schránky, ktorý sa používa ďalej pri práci s ňou. Rozdiel spočíva len v tom, že dáta, ktoré vysiela jeden proces sa ukladajú priamo do bufra druhého procesu. Tvar systémového volania pre vytvorenie schránky je nasledovný:

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

int socket(int domain, int type, int protocol);
```

Návratová hodnota z volania je deskriptor schránky.

Argumenty volania:

- `domain` – špecifikuje komunikačnú doménu, t.j. vyberá skupinu protokolov, ktoré sa budú využívať pri komunikácií. Tieto protokoly sú definované v `<sys/socket.h>`.

  Existujú dve široko využívané adresné domény:
  - **unixová** doména – v tomto prípade komunikujúce procesy zdieľajú súborový systém a nejedná sa o sieťovú komunikáciu. Adresa v tomto prípade je reťazec, reprezentujúci položku v súborovom systéme.
  - **Internetová** doména – procesy bežia v rôznych uzloch počítačovej siete. Adresa pozostáva z **internetovej adresy** hostiteľského počítača (32 bitová unikátna adresa, nazývaná IP adresa) a **čísla portu** na tom počítači. Čísla portov sú 16 bitové čísla bez znamienka. Nižšie čísla portov sú rezervované pre štandardné služby, napr. číslo portu pre FTP server je 21. Čísla nad 2000 sú obyčajne voľné.

  Komunikačné domény (skupiny protokolov), ktoré sú použiteľné sú:

Meno domény	Účel
PF_UNIX, PF_LOCAL	Lokálna komunikácia
PF_INET	IPv4 Internetové protokoly
PF_INET6	IPv6 Internetové protokoly

PF_IPX	IPX - Novellovské protokoly
PF_NETLINK	Komunikácia medzi procesmi jadra a užívateľskými procesmi
PF_X25	Rozhranie k protokolu ITU-T X.25 / ISO-8208
PF_AX25	Protokol AX.25 pre amatérske rádio
PF_ATMPVC	Protokoly pre prístup k ATM PVCs
PF_APPLETALK	Protokoly Appletalk
PF_PACKET	Nízkoúrovňové rozhranie paketov

- `type` – typ schránky (socket type). Dva procesy komunikujú len ak ich schránky sú rovnakého typu a rovnakej domény.

Označenie typu	Význam
SOCK_STREAM	Poskytuje spoľahlivé prúdové (stream) bajtové spojenie, ktoré zaručuje poradie správ.
SOCK_DGRAM	Datagramové spojenie (nespoľahlivé spojenie s pevnou dĺžkou správy), nezaručuje neopakovanie a poradie
SOCK_SEQPACKET	Spoľahlivé spojenie s pevnou maximálnou dĺžkou správy, ktoré garantuje poradie správ. Prijímateľ musí prečítať cely paket pri každom čítaní.
SOCK_RAW	Poskytuje priamy prístup k sieťovému protokolu, hlavne pre ďalšie spracovanie.
SOCK_RDM	Spoľahlivé datagramové spojenie, ktoré negarantuje poradie paketov. Nie je implementované.
SOCK_PACKET	Zastaralý, nemal by sa využívať v nových programoch

Niektoré typy schránok nemusia implementovať všetky typy protokolov. Napr. `SOCK_SEQPACKET` nie je implementovaný pre AF_INET.

Schránky typu `SOCK_STREAM` sú full-duplexový bajtový prúd, podobný rúram. Nezachovávajú hranice záznamu. Takáto schránka môže byť v stave „pripojená“ predtým ako nejaké dáta sú poslané alebo získané z nej. Pripojenie k inej schránke sa vytvorí pomocou volania `connect()`. Potom sa dáta čítajú/zapisujú pomocou `read()` a `write()` alebo niektorým variantom volaní `send()` a `recv()`. Po ukončení práce sa prúd musí uzatvoriť pomocou volania `close()`.

Komunikačné protokoly, ktoré implementujú `SOCK_STREAM` zabezpečujú to, že dáta sa nestrácajú a neduplikujú.

Schránky typu `SOCK_DGRAM` a `SOCK_RAW` umožňujú posielať datagramy príjemcom, označených vo volaní `sendto()`. Získavanie datargamov umožňuje volanie `recvfrom(2)`, ktoré vracia ďalší datagram spolu s adresou odosielateľa.

`SOCK_PACKET` je zastaralý typ, ktorý sa v súčasnosti nahradzuje volaním `packet()`. Vracia dáta priamo z drivera zariadenia.

Najčastejšie používané protokoly v doméne Internet sú TCP a UDP a tie sú používane pre schránky typu `SOCK_STREAM` a `SOCK_DGRAM`, resp.

- `protocol` – špecifikuje protokol, využívaný so schránkou. Obyčajne existuje len jeden protokol zo skupiny protokolov, ktorý podporuje daný typ schránky. V takom prípade sa argument `protocol` môže mať hodnotu 0. Ak existuje viac podporujúcich protokolov z danej skupiny, špecifikuje sa konkrétny protokol pomocou jeho čísla. Zobrazenie (!mapovanie“) čísiel na mena protokolov sa dá zistiť pomocou systémového volania `getprotoent()`, v manuáli systému `(man 5 protocols)` alebo priamo zo súboru `/etc/protocols`.

A.1.2 Inicializácia komunikácie pomocou schránky - systémové volanie `connect()`

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

int   connect(int  sockfd,  const  struct  sockaddr
              *serv_addr,  socklen_t addrlen);
```

Systémové volanie `connect()` previaže schránku `sockfd` s adresou, špecifikovanou pomocou `serv_addr`. Argument `addrlen` určuje rozmer adresy `addrlen`. Formát adresy závisí od adresného priestoru schránky.

Ak schránka je typu SOCK_DGRAM, potom dáta sú posielané a získavané len z adresy `serv_addr`.

Ak schránka je typu SOCK_STREAM alebo SOCK_SEQPACKET, potom toto volanie sa pokúša spojiť sa so schránkou, naviazanou na adrese `serv_addr`.

Všeobecne, schránky založené na nespojitých (connectionless) protokoloch sa môžu pripojiť úspešne len raz. Schránky založené na spojitých (connection-based) protokoloch môžu využiť `connect()` viac krát.

```
struct sockaddr_in {
       short int sin_family;          // skupina adries
       unsigned short int sin_port;   // číslo portu
       struct in_addr sin_addr;       // Internetova adresa
       unsigned char sin_zero[8];     // Rovnaký rozmer ako struct sockaddr
};
```

A.1.3 Spojenie deskriptora schránky s adresou – systémové volanie `bind()`

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

#include <sys/socket.h>

int   bind(int   sockfd,  const  struct  sockaddr
           *my_addr,  socklen_taddrlen);
```

Systémové volanie **`bind()`** previaže `sockfd` s lokálnou adresou `my_addr`, ktorá má dĺžku `addrlen`. Po vytvorení schránky pomocou `connect()`, schránka existuje, ale nie je pomenovaná.

Návratová hodnota z úspešného volania je 0, ináč –1 a `errno` je nastavené na číslo chyby.

Predtým ako schránka typu SOCK_STREAM začne prijímať správy je potrebné jej priradiť lokálnu adresu.

Pravidlá, ktoré platia pri priraďovaní adries sa líšia podľa skupín adries. Pomenovanie pre Internetovu doménu je zložitejšia ako pre UNIX doménu. Je potrebne vedieť meno hostiteľského počítača a komunikačný port.

Skutočná štruktúra, ktorá sa odovzdáva v argumente `my_addr` je závislá od skupiny adries. Jej štruktúra môže vyzerať podobne:

```
struct sockaddr {
     sa_family_t sa_family;    // skupina adries, hodnota typu AF_XXX
     char        sa_data[14]; // až 14 bajtov, závisí od adresy, ktorú
                              // používa protokol
    }
```

Jediný význam tejto štruktúry je pretypovať ukazovateľ štruktúry z `my_add`, aby sa predišlo varovných upozornení kompilátora. Táto štruktúra je pomôcka ako urobiť prepojenie bez použitia zložitejšej štruktúry `sockaddr_in`.

A.1.4 Čakanie na prichádzajúce spojenie – systémové volanie `listen()`

Schránka, ktorá bola vytvorená pomocou systémového volania `socket()` ukazuje svoju pripravenosť akceptovať prichádzajúce spojenie pomocou systémového volania `listen()`.Toto volanie sa aplikuje len pre schránky typu `SOCK_STREAM` alebo `SOCK_SEQPACKET`.

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

int listen(int sockfd, int backlog);
```

Argumenty volania sú:

- `sockfd` – deskriptor schránky,
- `backlog` – parameter, ktorý definuje rozmer frontu, v ktorom čakajú prichádzajúce spojenia. Ak príde spojenie a front je plný, klient môže dostať chybovú správu `ECONNREFUSED` alebo v prípade, že použitý protokol podporuje opätovné vysielanie, volanie môže skončiť úspešne.

A.1.5 Prijatie spojenia – systémové volanie `accept()`

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

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

Systémové volanie `accept()` sa používa pre schránky, založené na spojitých protokoloch (`SOCK_STREAM`, `SOCK_SEQPACKET`). Vyberie prvé z neobslúžených čakajúcich spojení z frontu a vráti deskriptor novej schránky, ktorá sa využije pre vstup/výstup, ak bude potrebné. Nová schránka nie je v stave čakania. Pôvodná schránka nie je ovplyvnená týmto volaním.

Argumenty volania:

- `sockfd` je schránka, ktorá bola vytvorená pomocou volania `socket()`, prepojená k lokálnej adrese volaním `bind()` a na ktorej sa čakalo pomocou `listen()`.
- `addr` je ukazovateľ na štruktúru `sockaddr`. Táto štruktúra po volaní je vyplnená s adresou schránky s ktorou je uskutočnené spojenie. Presný formát závisí od skupiny adries.
- `addrlen` argument, ktorý na začiatku obsahuje rozmer štruktúry, na ktorú ukazuje `addr`. Po návrate z volania obsahuje aktuálnu dĺžku (v bajtoch) vrátenej adresy. Ak `addr` je NULL, štruktúra sa nevyplní.

Ak vo fronte nečaká žiadne spojenie a schránka nie je označená ako neblokujúca, volanie `accept()` zablokuje volajúceho procesu kým príde nejaké spojenie. Ak schránka je označená ako neblokujúca, volanie skončí s chybou EAGAIN.

A.1.6 Prenos dát – čítanie a zápis z/do schránky

Na prenos dát pomocou schránky sa môže použiť niekoľko systémových volaní. Najjednoduchšími volaniami sú `read()` a `write()`, ktoré sú rovnaké ako v prípade diskových súborov.

```
#include <unistd.h>

ssize_t read(int fd, void *buf, size_t count);
ssize_t write(int fd, const void *buf, size_t count);
```

Argumenty volania sú:

- `fd` – deskriptor schránky, vrátený z volania `socket()` v klientskom programe a z volania `accept()` v serverovom programe.
- `buf` – bufer na uloženie správy,
- `count` – počet bajtov, ktoré sa majú poslať alebo veľkosť bufra pri čítaní.

Zápis obyčajne prebieha asynchrónne t.j. po zápise dát do cache pamäte sa riadenie vráti programu a skutočný zápis sa vykoná potom. Po úspešnom vykonaní zápisu sa vráti skutočný počet zapísaných bajtov. Ináč sa vráti –1 a `errno` sa nastaví na číslo chyby.

Funkcia `read()` vracia počet prečítaných bajtov, ktoré sú umiestnené v bufri. Systém vracia počet požadovaných bajtov ak deskriptor ukazuje na prúd, ktorý obsahuje toľko bajtov pred koncom prúdu (`end-of-file`). Ak návratová hodnota je 0, to znamená, že bol dosiahnutý koniec prúdu. V prípade chyby návratová hodnota je –1 a `errno` sa nastaví na číslo chyby.

Iná alternatíva volaní pre zápis do schránku sú volania `send()`, `sendto()` a `sendmsg()` a pre čítanie zo schránky `recv()`, `recvfrom()` a `recvmsg()`.

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

ssize_t send(int s, const void *buf, size_t len, int flags);
ssize_t sendto(int  s,  const  void  *buf, size_t len, int
          flags, const struct sockaddr *to, socklen_t tolen);
ssize_t sendmsg(int s, const struct msghdr *msg, int flags);
```

Systémové volanie `send()` sa môže použiť len ak schránka je v pripojenom stave t.j. prijímateľ je známy. Jediný rozdiel oproti volaniu `write()` je prítomnosť argumentu `flags`, ktorý je príznakom. Ak `flags` je 0, volanie je ekvivalentné volaniu `write()`.

Podobne `send(s,buf,len,flags)` je ekvivalentné s volaním `sendto(s,buf,len,flags,NULL,0)`.

Ak volanie `sendto()` je použité v spojitom režime (`SOCK_STREAM`, `SOCK_SEQPACKET`), argumenty *to* a `tolen` sú ignorované a ak ich hodnoty nie sú `NULL` a 0, vznikne chyba `EISCONN`. Ak schránka nie je skutočne pripojená, vznikne chyba `ENOTCONN`. Ináč rozmer adresa cieľa sa zadaná argumentom `tolen`. Pre `sendmsg()` adresa cieľa je zadaná argumentom `msg.msg_name` a dĺžka argumentom `msg.msg_len`.

Pre `send()` a `sendto()` správa je v `buf` a jej dĺžka v `len`. `sendmsg()` dovoľuje posielanie aj pomocných (riadiacích) dát.

Definícia štruktúry `msghdr` je nasledovná:

```
struct msghdr {
     void         *msg_name;       /* voliteľná adresa */
     socklen_t     msg_namelen;    /* veľkosť adresy */
     struct iovec *msg_iov;        /* scatter/gather pole */
     size_t        msg_iovlen;     /* počet prvkov v msg_iov */
     void         *msg_control;    /* pomocné dáta, pozri ďalej */
     socklen_t     msg_controllen; /* dĺžka bufra pomocných dát */
     int           msg_flags;      /* príznaky získanej správy */
     };
```

Riadiace dáta je možne poslať pomocou členov štruktúry `msg_control` a `msg_controllen`. Maximálna dĺžka bufra pre riadiace informácie je obmedzená (pozri `<sys/socket.h>`).

Vo volaní `sendmsg()` na správu sa ukazuje pomocou členov poľa `msg.msg_iov`.

Ak správa je príliš dlhá aby ju protokol spracoval automaticky, vráti sa chyba `EMSGSIZE` a správa sa nepošle.

Ak správa nebola doručená, `send()` nevracia indikáciu tohto stavu. Lokálne sa chyba hlasí návratovou hodnotou –1.

Ak sa správa nezmestí do bufra, `send()` sa zablokuje, pokiaľ schránka **nebola** nastavená v neblokujúcom režime. Ak režim je neblokujúci - vráti sa chyba EAGAIN.

Argument `flags` je bitová operácia OR medzi zadanými príznakmi. Argument môže byť 0 alebo môže byť špecifikovaný explicitne. Najčastejšie používanými príznakmi sú:

- `MSG_CONFIRM` (len pre Linux 2.3) – upozorňuje linkovú vrstvu, že bolo získané potvrdenie o doručení správy. Platí len pre schránky typu `SOCK_DGRAM` a `SOCK_RAW`,
- `MSG_DONTROUTE` – používa sa len pre diagnostické účely, znamená prenos bez routovania,
- `MSG_DONTWAIT` – umožňuje neblokujúcu operáciu. Ak operácia sa zablokuje, vráti sa stav EAGAIN,

- MSG_MORE (od Linux 2.4.4) – volajúci proces má viac dát na poslanie. Tento príznak sa používa s TCP schránkami.

  Od Linux 2.6. tento príznak je podporovaný aj pre UDP schránky. Upozorňuje jadro aby zoskupilo všetky dáta, zaslané vo volaniach s týmto nastaveným príznakom v jednom datagrame. Takýto datagram je zaslaný keď sa vykonáva volanie bez nastavenia tohto príznaku.

- MSG_NOSIGNAL – požaduje, aby sa nevysielal signál SIGPIPE pri chybách prúdovo orientovaných schránok keď druhý koniec spojenia je prerušený. Aj tak sa v takom prípade vráti chyba EPIPE.

- MSG_OOB – posiela dáta mimo spojenia (out of band) schránkam, ktoré podporujú tento spôsob (SOCK_STREAM). V tomto prípade chceme, aby tieto dáta boli spracované predtým ako budú nejaké iné dáta uložené do bufra.

Systémové volania recvfrom() a recvmsg() sa používajú pre získanie správ zo schránky nezávislé či schránka je založená na spojitom (connection based) alebo nespojitom (connectionless) protokole.

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

ssize_t recv(int s, void *buf, size_t len, int flags);
ssize_t recvfrom(int s, void *buf, size_t len, int flags,
                 struct sockaddr *from, socklen_t *fromlen);
ssize_t recvmsg(int s, struct msghdr *msg, int flags);
```

Ak argument from je NULL a použitý protokol poskytuje adresu zdroja, adresa je doplnená.

Argument fromlen je výstupným argumentom, inicializovaný na rozmer bufra, spojený s argumentom from a modifikovaný po návrate z volania tak, aby odzrkadľoval aktuálny rozmer uloženej adresy.

Systémová volanie recv() sa normálne využíva so spojitými protokolmi a je identické s volaním recfrom(), v ktorom argument from je NULL.

Všetky tri funkcia pri úspešnom volaní vracajú dĺžku správy. Ak správa je prilíč dlhá, aby sa zmestila do bufra, zvyšné bajty môžu byť znehodnotené podľa typu schránky z ktorej prišla správa.

Ak správa ešte neprišla, volania čakajú na ňu, pokiaľ schránka nie je nastavená ako neblokujúca. Obyčajne návrat z volania sa uskutoční a vrátia sa prítomné dáta a nečaká sa aby prišlo celé množstvo požadovaných dát.

Argument flags je bitová operácia OR medzi zadanými príznakmi - MSG_DONTWAIT, MSG_ERRQUEUE, MSG_OOB, MSG_PEEK, MSG_TRUNC, MSG_WAITALL, MSG_EOR,

MSG_TRUNC, MSG_CTRUNC, MSG_OOB a MSG_ERRQUEUE. Úplný popis jednotlivých príznakov čitateľ nájde v manuálových stránkach systému.

A.1.7 Ukončenie práce so schránkou

```
#include <unistd.h>

int close(int fd);
```

Po použití schránka môže byť zrušená pomocou systémového volania close(), ktorého argument je deskriptor schránky.

Po úspešnom volaní je vrátená hodnota 0, ináč sa vráti hodnota –1 a nastaví sa globálna premenná errno.

Ak v schránke sú nejaké dáta pre schránku typu SOCK_STREAM pri volaní close(), systém bude pokračovať v prijímaní dát.

A.1.8 Konverzia poradia bajtov adresy hostiteľského počítača a siete

Rôzne počítačové architektúry majú rozdielne poradie bajtov pri reprezentácií dát. Poradie bajtov je dôležité pri vyjadrovaní Internetovej adresy. Nasledovné funkcie konvertujú 16 bitovoé (short integer) a 32-bitové (long integer) hodnoty pre účely správnej reprezentácie adries pri ich konverzie medzi sieťovým formátom a formátom, používanom na hostiteľskom počítači.

```
#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);
```

Funkcia htons() sa používa na konverziu „host-to-network“ pre celé čísla typu short integer a htonl() pre celé čísla typu long integer.

Funkcia ntohs() sa používa na konverziu „network-to-host“ pre celé čísla typu short integer a ntohl() pre celé čísla typu long integer.

A.1.9 Pomocné funkcie

Niekedy je dôležite vedieť na ktorom hostiteľskom počítači sa vykonáva náš program. nasledovné systémové volania umožnia získať túto informáciu:

```
#include <netdb.h>
#include <sys/socket.h>        /* pre AF_INET */

struct hostent *gethostbyname(const char *name);
struct hostent *gethostbyaddr(const void *addr, int len,
                              int type);
```

Funkcia `gethostbyname()` vracia informáciu v štruktúre typu `hostent` pre zadané meno počítača.

```
    struct hostent {
        char    *h_name;         /* oficiálne meno host. počítača */
        char    **h_aliases;     /* zoznam aliasov */
        int     h_addrtype;      /* typ adresy hosťa */
        int     h_length;        /* dlžka adresy */
        char    **h_addr_list;   /* zoznam adries */
}
#define h_addr  h_addr_list[0]   /* pre spätnú kompatibilitu */
```

Argument `name` je buď meno počítača, alebo štandardná adresa v tvare s bodkami (podľa IPv4 alebo IPv6). Nasledujúci fragment programu ukazuje použitie funkcie `gethostbyname()`:

```
.
.
if (inet_aton(host, &inaddr) == 1)
            servaddr.sin_addr = inaddr;  /* zápis s bodkami */
   else if ( (hp = gethostbyname(host)) != NULL)
        memcpy(&servaddr.sin_addr, hp->h_addr, hp->h_length);
   else
        fprintf(strderr, "Neplatne meno host. pocitaca: %s",
                host);
.
.
```

Funkcia `gethostbyaddr()` vracia informáciu v štruktúre typu `hostent` pre zadanú adresu `addr` s dĺžkou `len` a typ adresy `type`. Platné typy adries sú AF_INET a AF_INET6. Argument `addr` je ukazovateľ na štruktúru, ktorej typ je závislý na type adresy.

Príklad A. 1: Program-server pre internetovú doménu s použitím TCP

```
/* Číslo portu sa zadáva jako argument príkazového riadku */

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

```
#include <netinet/in.h>
#include <string.h>

   int main(int argc, char *argv[])
   {
        int sockfd, newsockfd, portno, clilen;
        char buffer[256];
        struct sockaddr_in serv_addr, cli_addr;
        int n;

        if (argc < 2) {
            fprintf(stderr,"CHYBA, chyba cislo portu\n");
            return 1;
        }

       // vytvorenie schránky
        sockfd = socket(PF_INET, SOCK_STREAM, 0);
        if (sockfd < 0)
            perror("CHYBA pri otvarani schranky");

        memset( &serv_addr,0, sizeof(serv_addr));
        portno = atoi(argv[1]);

       // nastavenie polžziek štruktúry adresy servera
        serv_addr.sin_family = PF_INET;
        serv_addr.sin_addr.s_addr = INADDR_ANY;
        serv_addr.sin_port = htons(portno);

       //  previazanie deskriptora schránky s adresou hosta
        if (bind(sockfd, (struct sockaddr *) &serv_addr,
                                  sizeof(serv_addr)) < 0)
                 perror("CHYBA pri bind");

       // deklaruje ochotu dostávat správy a dlžku frontu správ
        listen(sockfd,5);
        clilen = sizeof(cli_addr);
        newsockfd = accept(sockfd, (struct sockaddr *) &cli_addr,
                           &clilen);
        if (newsockfd < 0)
             perror("CHYBA pri accept");
        memset(&buffer, 0, 256);

        // čítanie zo schránky
        n = read(newsockfd, buffer, 255);
        if (n < 0) perror("CHYBA pri citani zo schranky ");
        printf("Sprava: %s\n", buffer);

        // zápis do schránky
        n = write(newsockfd,"Dostal som tvoju spravu!", 24);
        if (n < 0) perror("CHYBA pri zapisu do schranky");
```

```
     return 0;
}
```

Príklad A.2: Program-klient pre internetovú doménu s použitím TCP

```
#include <stdio.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <netdb.h>
#include <string.h>


int main(int argc, char *argv[])
{
    int sockfd, portno, n;
    struct sockaddr_in serv_addr;
    struct hostent *server;

    char buffer[256];

     if (argc < 3) {
       fprintf(stderr,"usage %s hostname port\n", argv[0]);
       return 0;
    }
    portno = atoi(argv[2]);
    sockfd = socket(PF_INET, SOCK_STREAM, 0);
    if (sockfd < 0)
        perror("CHYBA pri otvarani schranky ");

    server = gethostbyname(argv[1]);
    if (server == NULL) {
        fprintf(stderr,"CHYBA, nie je taky host\n");
        return 0;
    }
    memset( &serv_addr,0, sizeof(serv_addr));
    serv_addr.sin_family = PF_INET;
    memcpy((char *)server->h_addr,
             (char *)&serv_addr.sin_addr.s_addr,
              server->h_length);
    serv_addr.sin_port = htons(portno);

    if (connect(sockfd, (struct sockaddr *)&serv_addr,
                sizeof(serv_addr)) < 0)
        perror("CHYBA pri connect");
    printf("Zadaj spravu: ");
    memset(&buffer,0,256);

    fgets(buffer, 255, stdin);
```

```
    // zápis do schránky
    n = write(sockfd, buffer, strlen(buffer));
    if (n < 0)
         perror("CHYBA pri zapise do schranky");
    memset(&buffer,0,256);
    n = read(sockfd,buffer,255);
    if (n < 0)
         perror("CHYBA citani zo schranky");
    printf("%s\n", buffer);
    return 0;
}
```

A.2 Záver

V tejto časti bol popísaný jeden z najstarších prostriedkov pre komunikáciu medzi procesmi v sieti - schránka. Pomocou schránok je možné vytvárať omnoho širšie služby, ktoré potrebujú výmenu dát v sieti.