PB173 Linux

11 Sokety

Roman Lacko xlacko1@fi.muni.cz

2022-12-02

Obsah

Sokety

IPC

Inter Process Communication (IPC)

- Výmena dát medzi procesmi
- Rôzne mechanizmy s rôznymi vlastnosťami
  - Príbuzné procesy?
  - Aká informácia sa prenesie?
  - Procesy na iných počítačoch?

IPC

Inter Process Communication (IPC)

Poznáme:

- Signály (seminár 6)
- Rúry (seminár 9)
- Notifikácie (seminár 10)

Ešte uvidíme:

- Sokety (dnes)
- Zdieľaná pamäť (nabudúce)

IPC: Model Klient a Server

Častokrát je užitočné modelovať aplikácie
ako procesy v odlišných rolách:

- **Server**
  - Implementuje službu
  - Typicky jedna inštancia v „doméne“ (počítač, používateľ, ...)
  - Systémový alebo používateľský démon
- **Klient**
  - Požaduje službu od servera

⚠ Požiadavky na model

- Viac klientov v jednom čase
- Komunikácia medzi klientom a serverom musí byť bezpečná

IPC: Model Klient a Server

Signály?

- Nespoľahlivé
- Nefektívny prenos dát
- Oprávnenia

Rúry?

- Jednosmerný tok dát
- Vlastná rúra pre každú dvojicu procesov

Sokety: IPC

Klient s **rúrami**

```
int fd_rd = openat(SERVICE_DIR, "service.rd", O_RDONLY);
int fd_wr = openat(SERVICE_DIR, "service.wr", O_WRONLY);

/* Check that ‹fd_rd› and ‹fd_wr› are pipes. */

char request[REQUEST_SIZE];
prepare_request(request, sizeof(request), ...);
ssize_t wr = write(fd_wr, request, sizeof(request));

char response[RESPONSE_SIZE];
ssize_t rd = read(fd_rd, response, sizeof(response));
```

☹ Funguje len s jedným klientom

Sokety: IPC

Klient s **obojsmernou rúrou**

```
HANDLE pipe = CreateFile("\\\\.pipe\\service",
        GENERIC_READ | GENERIC_WRITE, …);

/* Check that ‹pipe› was successfully opened. */

char request[REQUEST_SIZE];
prepare_request(request, sizeof(request), ...);
WriteFile(pipe, request, sizeof(request), NULL, NULL);

char response[RESPONSE_SIZE];
ReadFile(pipe, response, sizeof(response), &rb, NULL);
```

Funguje len s jedným klientom, nie je POSIX

Sokety

- Dve príchute
  - *Aktívny soket* umožňuje obojsmerný prenos dát
  - *Pasívny* soket umožňuje prijímať klientov
    Pre každého klienta vytvorí nový aktívny soket
- Procesy nemusia byť na tom istom systéme (sieťové sokety)

Sokety

Klient so **soketom**

```
int sockfd = socket(/* … */);
connect(sockfd, /* … */);

/* Check that connection succeeded. */

char request[REQUEST_SIZE];
prepare_request(request, sizeof(request), ...);
ssize_t wr = write(sockfd, request, sizeof(request));

char response[RESPONSE_SIZE];
ssize_t rd = read(sockfd, response, sizeof(response));
```

Sokety

- Reprezentácia pomocou deskriptora
- `read()`, `write()` (ale tiež `send()` a `recv()`)
- Dva druhy (oboje môžu byť *aktívne* a *pasívne*):
  - Unixový — IPC na tom istom počítači
  - Sieťový — IPC medzi rôznymi počítačmi

Jednotné rozhranie

Aplikácia môže poskytovať služby lokálne, vzdialene, alebo oboje.

Unixový soket

Unixový soket

UNIX soket

- Nepomenovaný (≈ anonymná rúra)
- Pomenovaný (≈ pomenovaná rúra)
- Abstraktný (pomenovný soket bez súboru)

Unixový soket: `socketpair()`

Nepomenované sokety

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

int socketpair(int domain, int type, int protocol, int sv[2]);

#define AF_UNIX         /* … */ /* → domain */
#define SOCK_STREAM     /* … */ /* → type */
#define SOCK_DGRAM      /* … */
#define SOCK_SEQPACKET  /* … */
```

Hodnotu `protocol` nastavte na 0.

Unixový soket: `socket()`

Pomenovaný soket

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

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

#define SOCK_NONBLOCK /* … */ /* → type | … */
#define SOCK_CLOEXEC  /* … */
```

V závislosti od typu spojenia ďalšie funkcie:

- Nespojovaný protokol (`SOCK_DGRAM`)
- Spojovaný protokol (`SOCK_STREAM`, `SOCK_SEQPACKET`)

Unixový soket: sendto()

Nespojovaný protokol: Odosielanie

```
struct sockaddr_un {
    sa_family_t sun_family; /* AF_UNIX */
    char sun_path[108];
};

ssize_t sendto(int sock, const void *buf, size_t len, int flags,
        const struct sockaddr *dest, socklen_t addrlen);

#define MSG_DONTWAIT   /* … */

/* write(S, B, L)
 *   = sendto(S, B, L, 0, NULL, 0) */
```

Unixový soket: `bind()`, `recvfrom()`

Nespojovaný protokol: Prijímanie

```
int bind(int sock, const struct sockaddr *addr, socklen_t addrlen)

ssize_t recvfrom(int sock, void *buf, size_t len, int flags,
        struct sockaddr *src, socklen_t *addrlen);

/* read(S, B, L)
 *   ≈ recvfrom(S, B, L, NULL, NULL) */
```

`bind()` vyrobí soket v súborovom systéme

- Pre `sendto()` sa vyhodnocujú práva
- Ak existuje, `bind()` zlyhá

`read()` a `recvfrom()` sa správajú inak pre datagram dĺžky 0.

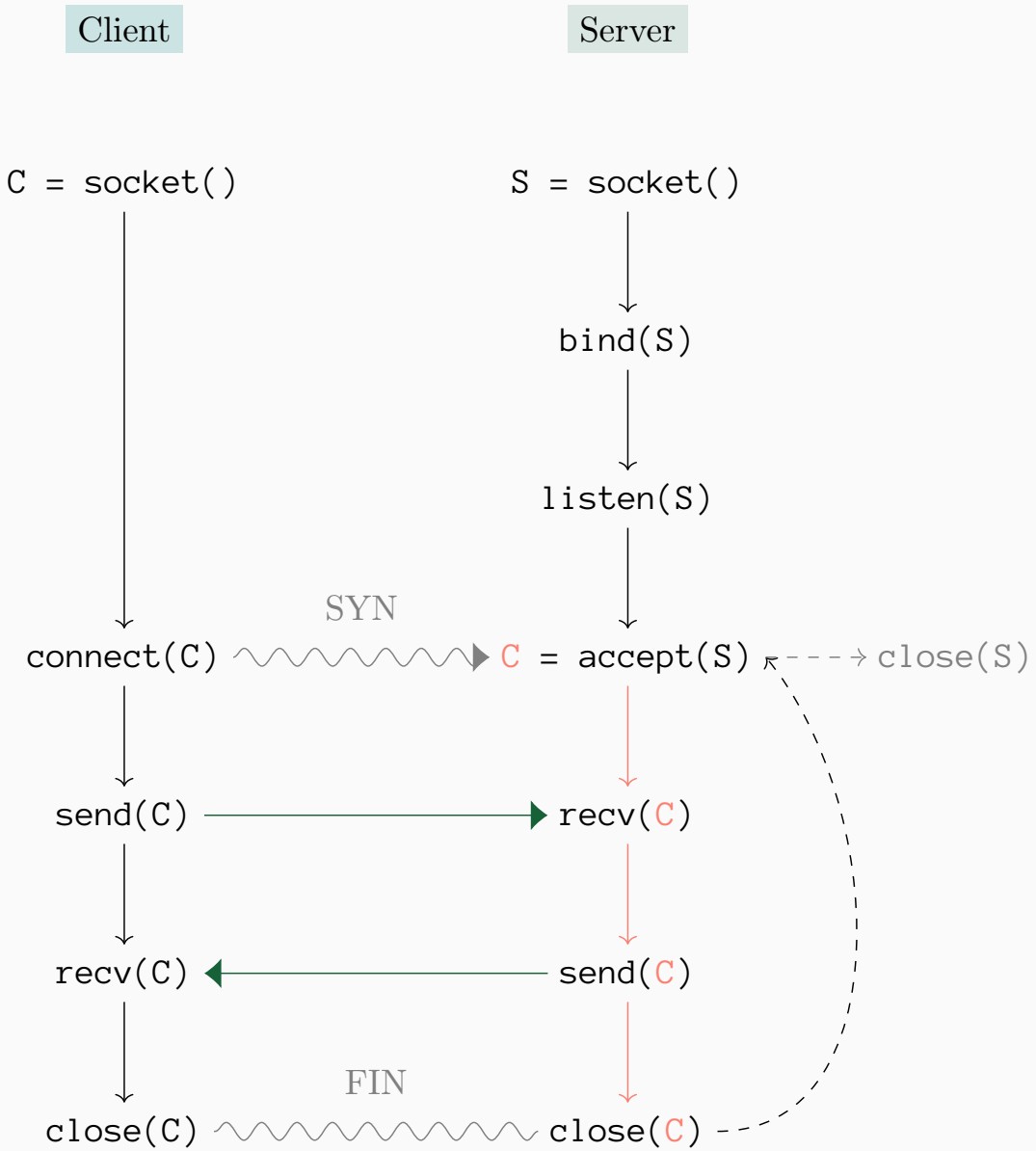
Unixový soket

Abstraktný soket

- sun_path začína `\0` (nulový bajt)
- Zvyšných 107 bajtov je identifikátor soketu

Identifikátor soketu musí byť jedinečný.
Môže viesť k súbehu.

Unixový soket: Klient-Server



Unixový soket: Klient

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

Vytvorí spojenie s cieľovou adresou.

Nespojovaný soket

`connect()` je možné použiť aj na nespojovanom sokete:

- nastaví východziu adresu pre `send()`,
- obmedzí prichádzajúce pakety pre `recv()`.

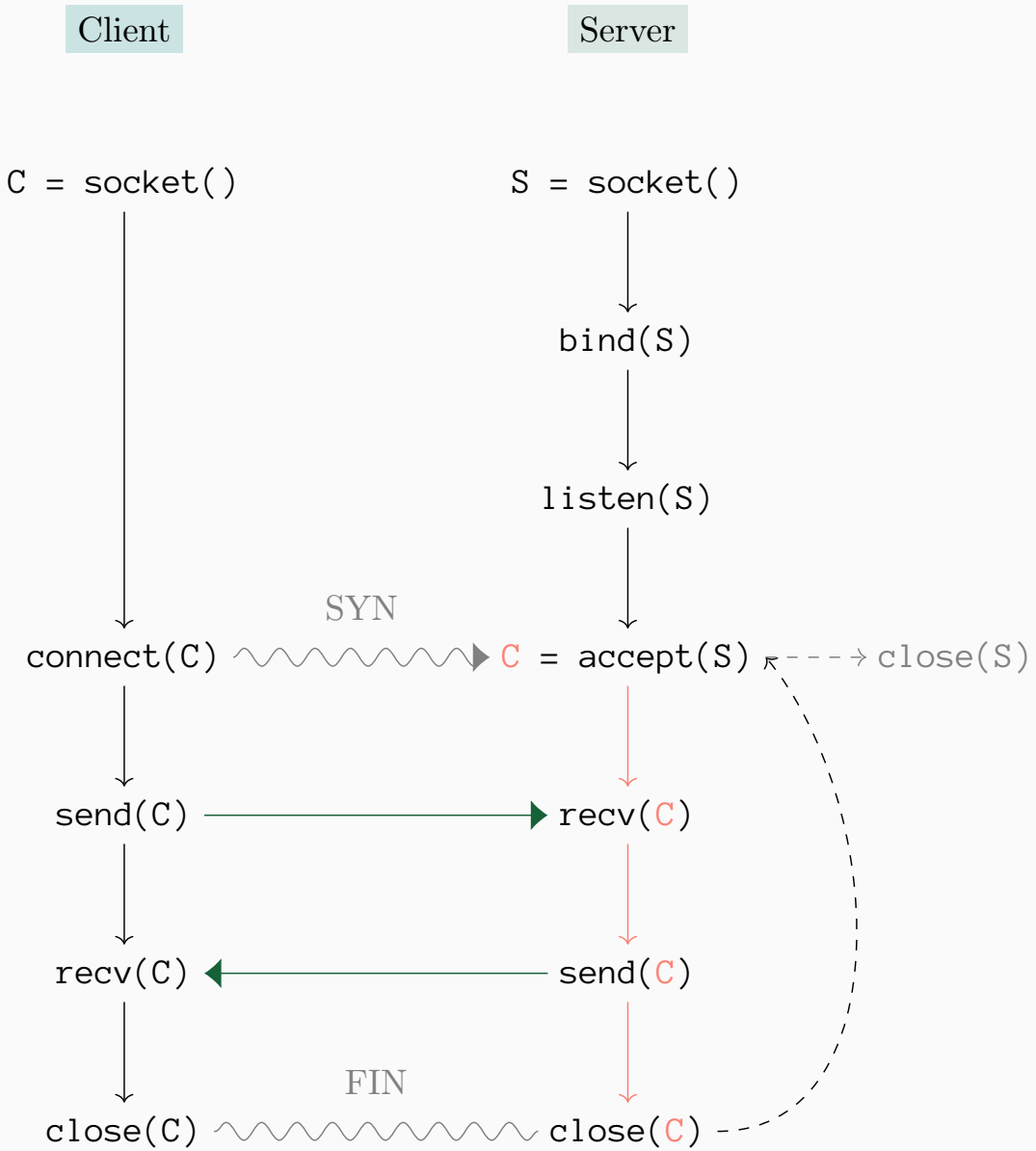
Unixový soket: Klient

```c
ssize_t send(int sockfd, const void *buf, size_t len, int flags);
/* write(S, B, L) = send(S, B, L, 0)
 *                = sendto(S, B, L, 0, NULL, 0) */

#define MSG_DONTWAIT /* … */
#define MSG_NOSIGNAL /* … */

ssize_t recv(int sockfd, void *buf, size_t len, int flags);
/* read(S, B, L) ≈ recv(S, B, L, 0)
 *               ≈ recvfrom(S, B, L, 0, NULL, NULL) */
```

Unixový soket: Server



Unixový soket: Server

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

int bind(int sock, const struct sockaddr *addr, socklen_t addrlen)

int listen(int sock, int backlog);

int accept(int sock, struct sockaddr *addr, socklen_t *addrlen);
int accept4(… sock, … *addr, … *addrlen, int flags);
```

! Nový deskriptor

Volanie accept() vráti **nový** deskriptor pre klienta.

Unixový soket: Server

Pozor na rozdiel

Nespojovaný soket:
Server číta a zapisuje priamo.

Spojovaný soket:
Server volá accept( ), dostane deskriptor pre klienta,
na ktorom volá I/O operácie.

Unixový soket: Server

Graceful Shutdown (Nenásilné ukončenie)

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

int shutdown(int sock, int how);

#define SHUT_RD   /* … */
#define SHUT_WR   /* … */
#define SHUT_RDRW /* … */
```

Typické použitie:

- `shutdown()` na hlavnom sokete zablokuje nové spojenia
- server dokončí rozbehnuté požiadavky
- `close()` na hlavnom deskriptore

Sieťové programovanie

Sieťové spojenie

- Soket `AF_INET`, `AF_INET6`, ...

Problémy:

- Ako pripojiť soket na inom počítači?
- Počítač s iným systémom a endianitou

Zariadenie v sieti nemôže predpokladať endianitu ostatných zariadení!

Host-Byte Order (HBO)

- Poradie bajtov na platforme
- Big Endian, Little Endian, Mixed Endian, ...

Network-Byte Order (NBO)

- Poradie bajtov podľa TCP
- **Vždy** Big Endian

NBO používajú typicky aj súborové systémy

Prevod endianity (POSIX)

```
#include <arpa/inet.h>
uint32_t htonl(uint32_t hostlong);  /* HBO to NBO Long */
uint16_t htons(uint16_t hostshort); /*          … Short */
uint32_t ntohl(uint32_t netlong);
uint16_t ntohs(uint16_t netlong);
```

Neštandardné funkcie

```
#include <endian.h>
uintβ_t htoτβ(uintβ_t v); /* τ ∈ {be, le}, β ∈ {16, 32, 64} */
uintβ_t τtohβ(uintβ_t v);

uint32_t htobe32(uint32_t v); /* ≈ htonl(v) */
```

Sieťové programovanie: Preklad mien

Preklad adresy a služby

```
#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);
```

- `hints`: jeden prvok (môže byť `NULL`)
- `res`: *spájaný zoznam* adries

Sieťové programovanie: Preklad mien

Preklad adresy a služby

```
struct addrinfo {
    int ai_flags;
    int ai_family;   /* → socket() */
    int ai_socktype; /*   …        */
    int ai_protocol; /*   …        */

    struct sockaddr *ai_addr; /* → connect(), bind() */
    socklen_t ai_addrlen;     /*   …                 */
    char *ai_canonname;
    struct addrinfo *ai_next;
};
```

- `AI_UNSPEC` pre `hints` ak chceme každú adresu
- `hints` môže byť `NULL`

Záznam pre adresy

Videli sme už `struct sockaddr_un`.

```
struct sockaddr {
    sa_family_t sa_family;
    char sa_data[/* … */];
};

struct sockaddr_storage {
    sa_family_t sa_family;
    char __ss_padding[_SS_PADSIZE];
    /* Alignment. */
};
```

Sieťové programovanie: Adresy

IP v4

```
struct sockaddr_in {
    sa_family_t sin_family;  /* AF_INET */
    in_port sin_port;        /* Port (NBO) */
    struct in_addr sin_addr;
};

struct in_addr {
    uint32_t s_addr;         /* Address (NBO) */
};
```

```
man 7 ip
```

Sieťové programovanie: Adresy

IP v5

Sieťové programovanie: Adresy

IP v6

```
struct sockaddr_in6 {
    sa_family_t sin6_family;   /* AF_INET6 */
    in_port_t sin6_port;       /* Port (NBO) */
    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];
};
```

```
man 7 ipv6
```

Reverzný preklad

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

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

#define NI_NUMERICHOST /* … */
#define NI_NUMERICSERV /* … */

#define NI_MAXHOST     /* … */
#define NI_MAXSERV     /* … */
```

Sieťové programovanie: Pripojenie

Ak už máme adresu, zvyšok je rovnaký ako pre UNIX sokety.

Klient (spojovaný protokol)

- `getaddrinfo()` zistíme cieľovú adresu a nastavenie
- Na výsledkoch postupne skúsime `socket()` a `connect()`
  - Obvykle stačí prvá kombinácia, ktorá funguje

Server (spojovaný protokol)

- `getaddrinfo()` s `AI_PASSIVE` v `hints->ai_flags`
- `socket()`, `bind()`, `listen()`, `accept()`
- Adresu klienta ako reťazec získame z `getnameinfo()` s `NI_NUMERICHOST`

Návrh protokolu

Návrh protokolu

Typické ciele návrhu:

- jednoduchosť
- škálovateľnosť
- efektivita
- rozšíriteľnosť

V princípe dva základné smery:

- binárne správy
- textové správy

Návrh protokolu: Binárne správy

Binárne správy

- ✔ malé správy
- ✔ štrukturované dáta
- ✔ prijateľná rozšíriteľnosť [1]
- ✘ ťažko čitateľné človekom
- ✘ ošetrenie platformových závislostí (endianita, šírka typov)

HTTP/2, NTP, NFS, ...

[1] Pri dodržaní rozumných návrhových vzorov

Návrh protokolu: Binárne správy

```
enum message_type {
    MSG_QUERY,
    MSG_INSERT,
    /* … */
};

struct message_payload_query {
    enum message_type msg_type;
    char msg_status[8];
};

struct message_payload_insert {
    enum message_type msg_type;
    uint64_t msg_target_id;
    uint64_t msg_value_id;
};
```

Návrh protokolu: Text

Textové správy

- ✔ jednoduchšie ladenie
- ✔ jednoduchá rozšíriteľnosť
- ✘ nutné zložitejšie parsovanie
- ✘ typicky väčší dátový prenos než binárne správy

⚠ Nevymýšľajte koleso

Vyhnite sa vlastnému formátu; použite radšej niečo existujúce
XML, HTTP, JSON, ...

HTTP/1, SMTP, XMPP

Návrh protokolu: Text

HTTP

```
PUT /api/v2/users/10/description HTTP/1.1
Host: kontr.cz
Authenticate: Bearer ...
```

JSON

```
{
    "command": "modify_user",
    "attribute": "description",
    "user_id": 10,
    "auth": ...
}
```

Návrh protokolu

Čo je obsahom správ?

- Dôležitý aspekt návrhu
- Ovplyvňuje ostatné charakteristiky

Sémantické typy správ (všeobecne)

- *Príkazy* definujú stav dialógu medzi stranami
- *Kontrolné správy* upravujú stav dialógu
  (napr. “OK”, “Error...”)
- *Dáta a metadáta* sú všeobecné doplňujúce správy v rámci stavu, niekedy súčasťou kontrolných správ (napr. “OK + výsledok výpočtu”)

Návrh protokolu

Protokol

- Špecifikuje typy správ a ich formát
- Definuje stavy komunikácie a prechody medzi nimi

Bezstavové rozhranie

Moderný prístup k návrhu protokolov.
Server neudržuje žiadne informácie o stave.

Nemusí byť vhodný na každé použitie.

Ďalšie možnosti

Pokročilé nastavenie soketu

```c
#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);

#define SOL_SOCKET /* … */
```

`level` určuje úroveň, na ktorej sa nastavenie uplatní

- `SOL_SOCKET` - generické API alebo UNIX
- `IPPROTO_{IP,IPV6}` - nastavenie pre sieťovú vrstvu
- `IPPROTO_{TCP,UDP}` - nastavenie pre transportnú vrstvu

Pokročilé nastavenie soketu

Užitočné nastavenia

```
// man 7 socket
#define SO_REUSEADDR /* … */
#define SO_REUSEPORT /* … */

// man 7 unix
#define SO_PEERCRED /* … */

// man 7 ipv6
#define IPV6_V6ONLY /* … */
```

Informácie o protokole

```c
#include <netdb.h>

struct protoent *getprotoent(void);
void setprotoent(int stayopen);
void endprotoent(void);

struct protoent *getprotobyname(const char *name);
struct protoent *getprotobynumber(int proto);

struct protoent {
    char *p_name;
    char **p_aliases;
    int p_proto;
};
```

Doplňujúce správy

```
ssize_t sendmsg(int sockfd, const struct msghdr *msg, int flags);
ssize_t recvmsg(int sockfd, struct msghdr *msg, int flags);

struct msghdr {
    void *msg_name;             /* Peer address */
    socklen_t msg_namelen;
    struct iovec *msg_iov;      /* IO vector */
    struct size_t msg_iovlen;
    void *msg_control;          /* Ancillary messages */
    size_t msg_controllen;
    int msg_flags;
};
```

Doplňujúce správy

Príklady použitia doplňujúcich správ:

- (IP) Informácie z hlavičiek rámcov
- (UNIX) Prenos zdrojov iným procesom
- (UNIX) Prenos oprávnení medzi procesmi

Záver

Zdroje

- Beej's Guide to Network Programming
- Implementing AF-independent application