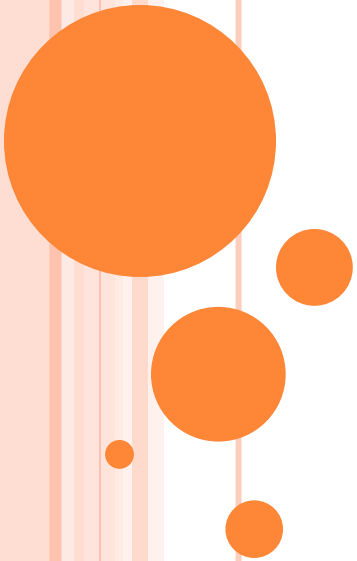


Programovacie jazyky pre vstavané systémy

Ukazovatele na funkcie, sockety

Otázky z minulej prednášky

- Čo je to generický smerník?
- Čo označujú termíny „NULL pointer“, „dangling pointer“ a „wild pointer“?
- Čo rozumieme pod dynamickou správou pamäte?
- Aký je rozdiel medzi funkciami malloc, calloc a realloc?
- Aké typy polí existujú v jazyku C?
- Ako sú implementované dvojrozmerné polia v jazyku C?
- Akými spôsobmi môžeme implementovať dvojrozmerné polia s využitím dynamickej pamäte?
- Ako musí vyzerať hlavička funkcie, ktorá má vrátiť smerník prostredníctvom parametra?

Funkcie vracajúce polia

- V jazyku C **nemôže byť návratový typ funkcie pole**. Funkcia však môže vrátiť ukazovateľ na pole. Napr.:

```
int (*vytvorPole(int n))[] {
    int (*pole)[] = calloc(sizeof(int), n);
    return pole;
}
int (*vytvorPole2D(int m))[][10];

int (*pole)[] = vytvorPole(10);
(*pole)[1] = 5;
int (*pole2D)[][10] = vytvorPole2D(20);
```

- Ak chceme, aby funkcia vracala pole, tak návratový typ definujeme ako **smerník na prvok poľa**:

```
int* vytvorPole(int n) {
    int* pole = calloc(sizeof(int), n);
    return pole;
}

int* pole = vytvorPole(10);
pole[1] = 5;
```

Ukazovateľ na funkciu

- Ukazovateľ nemusí uchovávať len adresu objektu, ale môže uchovávať aj adresu funkcie. Napr.:

```
int min(int a, int b) {
    return a < b ? a : b;
}

int (*f1)(int a, int b) = min;
int (*f2)(int, int) = &min;
int (*f3)(int, int);
f3 = min;

int vys1 = f1(10, 20);
int vys2 = (*f1)(10, 20);
int vys3 = (***f1)(10, 20);

typedef int (*FUN)(int a, int b);
FUN f4 = min;
```

Konverzia funkcie na smerník (1)

- V jazyku C je každý výraz buď objektového alebo funkčného typu.
- Identifikátor funkcie („function designator“) predstavuje výraz funkčného typu.
- Identifikátor funkcie sa pri každom použití implicitne konvertuje na r-hodnotu typu smerník na funkciu s výnimkou prípadov, kedy identifikátor vystupuje ako operand operátora:
  - & (address-of)
  - sizeof – chyba počas kompilácie
  - _Alignof (C11) – chyba počas kompilácie
- V dôsledku predchádzajúcej konverzie platí, že operátor volania funkcie (t.j. operátor ( )) je definovaný pre smerník na funkciu a nie pre identifikátor funkcie.

Konverzia funkcie na smerník (2)

- Uvažujme nasledujúcu definíciu:

```
int min(int a, int b) {
    return a < b ? a : b;
}
```

- Ak niekde v kóde (napr. vo funkcii main) napíšeme

```
int a = min(10, 20);
```

tak v skutočnosti sa vykoná nasledujúci kód:

```
int a = (&min)(10, 20);
```

- Vyššie spomínaná konverzia implikuje, že zápisy:

```
int (*f1)(int a, int b) = min;
int (*f1)(int a, int b) = &min;
```

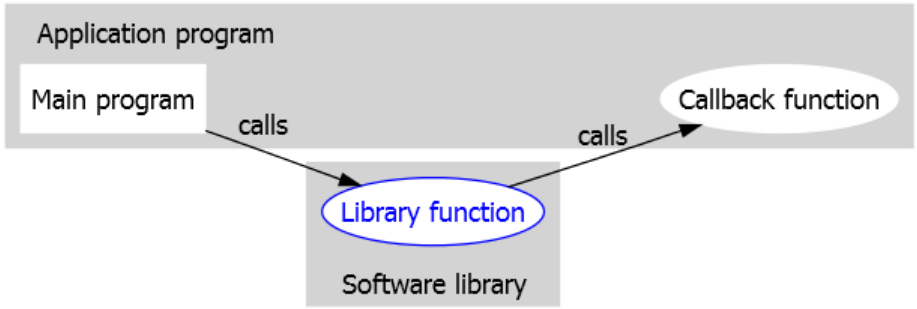
sú formálne totožné.

- Podobne platí:

```
min(10, 20) == (*min)(10, 20) == (&min)(10, 20) ==
f1(10, 20) == (*f1)(10, 20) == (**f1)(10, 20) !=
(&f1)(10,20);
```

Ukazovatele na funkcie – callback

- Callback – časť vykonateľného kódu, ktorá je odovzdaná ako argument do iného kódu, v ktorom sa vykoná v určitom vhodnom čase.
- Callback predstavuje základ pre činnosť udalostne orientovaných systémov (napr. oknové operačné systémy).
- Pomocou callback-u je možné vytvárať „generické“ funkcie (napr. qsort a bsearch v <stdlib.h>), spracovávať signály (<signal.h>), vytárať viacvláknové aplikácie (POSIX <pthreads.h>),...
- Synchrónny callback
- Asynchrónny callback



(Zdroj: Wikipedia)

Synchrónny callback

- Blokovací („bloking“) callback.
- Odovzdávaný kód sa vykoná pred tým, ako skončí kód, do ktorého bol odovzdaný.
- Odovzdávaný kód sa vykonáva v tom istom vlákne ako kód, do ktorého bol odovzdaný.
- Príklady využitia:
  - <stdlib.h>:
    - void qsort(void *base, size_t nitems, size_t size, **int (*compar)(const void *, const void*)**)
    - void *bsearch(const void *key, const void *base, size_t nitems, size_t size, **int (*compar)(const void *, const void *)**)

Synchrónny callback – príklad (1)

```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

void genericSwap(void* a, void* b, size_t size) {
    char tmp[size];
    memcpy(tmp, a, size);
    memcpy(a, b, size);
    memcpy(b, tmp, size);
}

void selectSort(void *base, size_t nitems, size_t size,
                int (*compare)(const void *, const void*)) {
    for (size_t i = 0; i < nitems; i++) {
        void *min = (char *)base + size * i;
        for (size_t j = i + 1; j < nitems; j++) {
            void *item = (char *)base + size * j;
            if (compare(min, item) > 0) {
                min = item;
            }
        }
        genericSwap((char *)base + size * i, min, size);
    }
}
```

Synchrónny callback – príklad (2)

```
26  int compareInt(const void *a, const void *b) {
27      int pomA = *((const int *)a);
28      const int *pomB = (const int *)b;
29
30      if (pomA < *pomB)
31          return -1;
32      else if (pomA == *pomB)
33          return 0;
34      else
35          return 1;
36  }
37
38  int main(int argc, char** argv) {
39  #define SIZE 100
40      int pole[SIZE];
41
42      naplnPole(SIZE, pole);
43      printf("Pole pred utriedenim:\n");
44      vypisPole(SIZE, pole);
45      selectSort(pole, SIZE, sizeof(pole[0]), compareInt);
46      printf("Pole po utriedeni:\n");
47      vypisPole(SIZE, pole);
48
49      return 0;
50  #undef SIZE
51  }
```

Asynchrónny callback

- Oneskorený („deferred“) callback.
- Odovzdávaný kód sa môže vykonať v ľubovoľnom čase po tom, ako skončí kód, do ktorého bol odovzdaný.
- Odovzdávaný kód sa môže vykonávať v inom vlákne.
- Príklady využitia:
  - <signal.h>:
    - void (*signal(int sig, void (*func)(int)))(int)
  - <pthread.h>:
    - int pthread_create(pthread_t *thread, const pthread_attr_t *attr, void *(*start_routine) (void *), void *arg)
  - implementácia obsluhovačov udalostí („event handlers“)

Asynchrónny callback – príklad

```c
#include <stdio.h>
#include <signal.h>
#include <unistd.h>

static int koniec = 0;

void spracujSignal(int signal) {
    if (signal == SIGINT) {
        koniec = 1;
    }
}

int main(int argc, char *argv[]) {

    signal(SIGINT, spracujSignal);

    while (!koniec) {
        printf("Cakam na prerusenie\n");
        sleep(1);
    }
    printf("Program bol preruseny\n");
    sleep(3);

    return 0;
}
```

Čítanie zložitejších deklarácií/definícií (podľa P. Herout – Učebnica jazyka C)

1. Nájdeme identifikátor a od neho čítame doprava.
2. Ak natrafíme na samostatnú pravú zátvorku „)“ (nie na dvojicu „( )“) vraciame sa na korešpondujúcu ľavú zátvorku a od nej čítame opäť doprava, pričom preskakujeme všetko už prečítané.
3. Ak natrafíme na bodkočiarku, vraciame sa na najľavejšie doposiaľ spracované miesto a od neho čítame doľava.

- Príklady:
  - int* a[10];
  - int (*a)[10];
  - int* (*a)[10];
  - double* f(int, int);
  - double (*f)(int, int);
  - double* (*f)(int, int);
  - ~~double f()[];~~
  - ~~double* f()[];~~
  - ~~double (*f)()[];~~
  - double (*f(int, int))[];
  - double* (*(*f)())[];
  - ~~double* (*f)()[];~~
  - ~~int f[](int, int);~~
  - ~~int (f[])(int, int);~~
  - int (*f[])(int, int);
  - char *(*(**f[][8])())[];
  - int (*(*)())() //abstraktný deklarátor
  - typedef int (*(*(*FUN_ARR)())[])(int, int);

POSIX

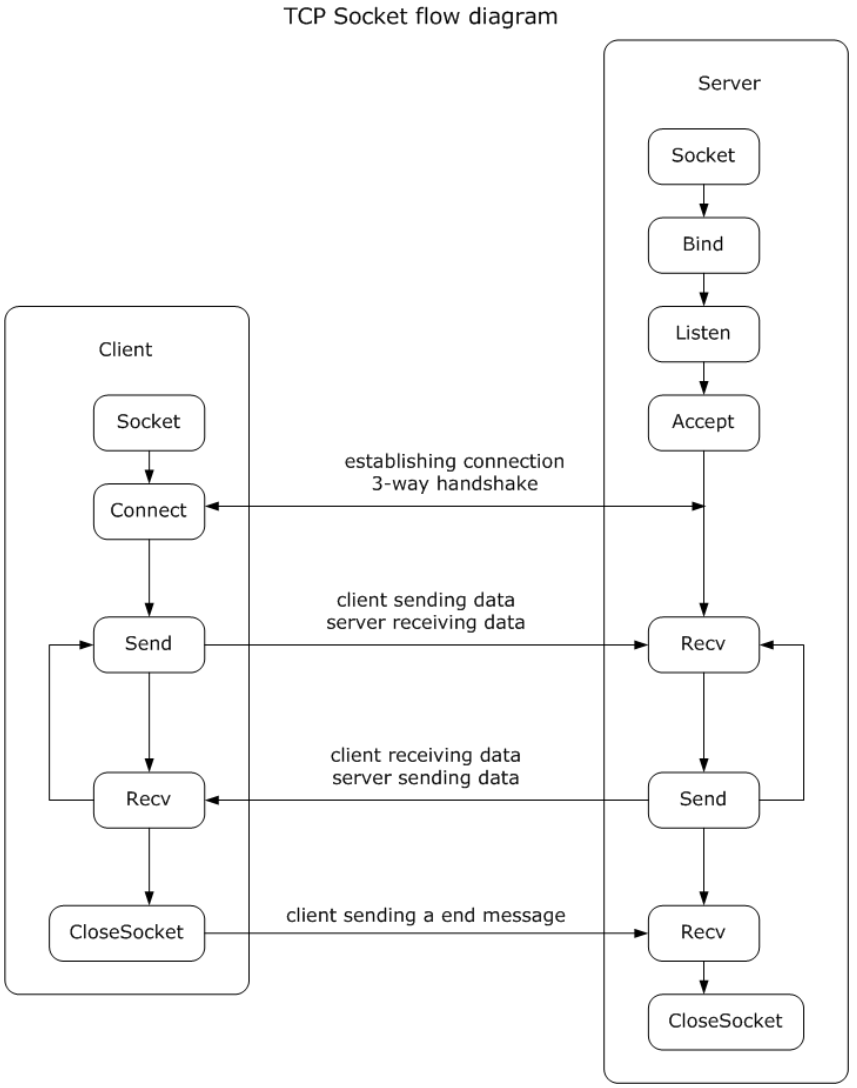
- Portable Operating System Interface – skupina štandardov definovaných IEEE Computer Society pre udržiavanie kompatibility rôznych operačných systémov (predovšetkým operačné systémy vytvorené na báze UNIXU).
- Definuje rozhranie ako pre programátorov (API), tak aj pre používateľov (intepreter príkazov, jednotné rozhranie rôznych softvérových utilít).
- Súčasťou POSIX je API pre paralelizáciu procesov (Pthreads) ako aj API pre medziprocesovú komunikáciu (IPC) s využitím lokálnych alebo internetových socketov.

POSIX sockety (1)

- Sockety predstavujú prostriedok IPC, ktorý na komunikáciu medzi procesmi používa model klient-server.
- Socket predstavuje abstraktnú reprezentáciu koncového bodu sieťovej komunikačnej cesty.
- V rámci POSIX je socket reprezentovaný ako popisovač súboru (file descriptor), pomocou ktorého je možné poslať dáta po sieti.
- POSIX sockety umožňujú komunikáciu v rôznych doménach, pričom jednotlivé domény sa líšia hlavne spôsobom adresácie.
- V danej komunikačnej doméne je server prístupný na základe svojej unikátnej adresy.
- V rámci POSIX je definovaných niekoľko typov domén. My sa zameriame na internetovú doménu, t.j. sockety umožňujúce komunikáciu pomocou protokolov založených na IPv4.

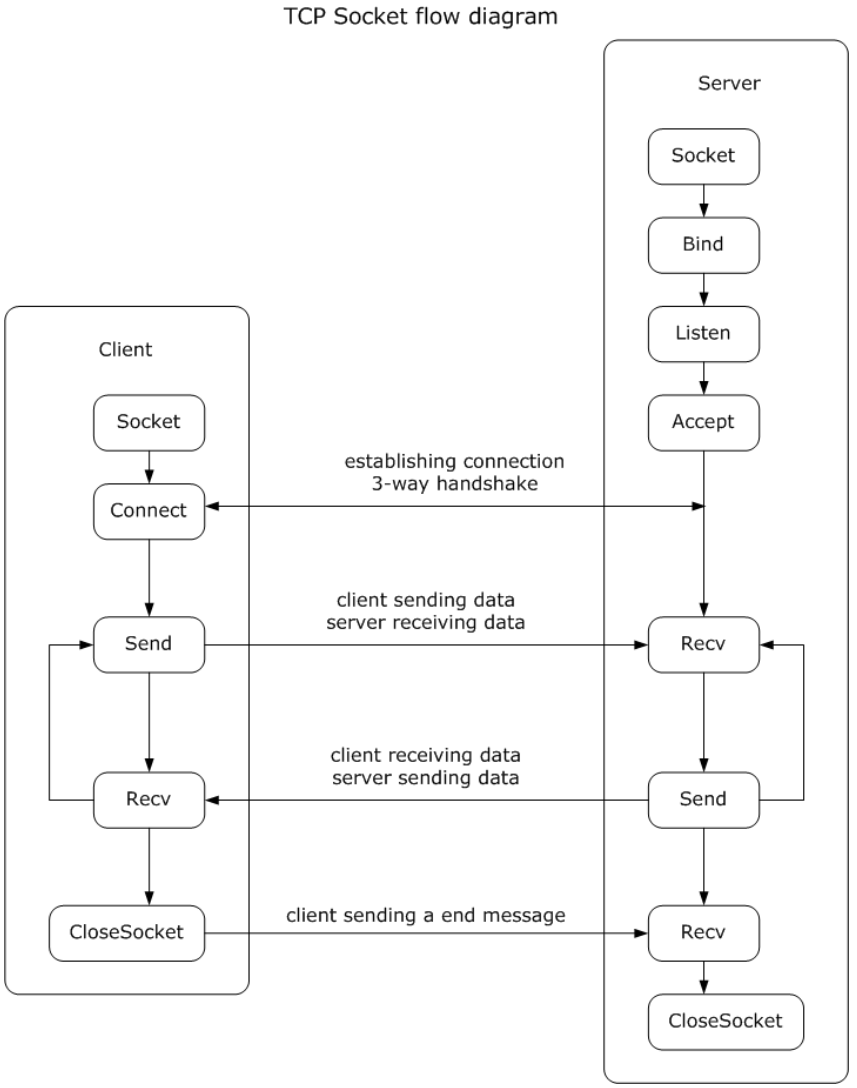
Sockety – schéma komunikácie (1)

- Práca so socektmi je veľmi podobná práci sú súbormi.
- Klient – na základe informácii o adrese servera (IP adresa a port) vytvorí socket, pomocou ktorého sa pripojí na server. Po skončení komunikácie uzavrie socket.



Sockety – schéma komunikácie (2)

- Server – musí vytvoriť socket, na ktorom bude čakať na spojenie. Ak sa k nemu pripojí klient, vytvorí sa nový socket, pomocou ktorého bude server komunikovať s týmto klientom.



FUNKCIA SOCKET <SYS/SOCKET.H>

- int socket(int domain, int type, int protocol)
  - funkcia vytvorí socket a vráti jeho popisovač (v prípade chyby vráti hodnotu -1 a nastaví hodnotu globálnej premennej errno);
  - parametrami funkcie sú:
    - domain – komunikačná doména:
      - AF_UNIX, AF_LOCAL
      - **AF_INET**
      - AF_INET6
      - ...
    - type – typ socketu:
      - **SOCK_STREAM**
      - **SOCK_DGRAM**
      - SOCK_SEQPACKET
      - **SOCK_RAW**
      - SOCK_RDM
      - ~~SOCK_PACKET~~
    - protocol – špecifikuje konkrétny protokol, ktorý sa má použiť s príslušným socketom (ak existuje viacej protokolov pre danú doménu a daný typ socketu). Ak chceme použiť základné nastavenia, použijeme hodnotu 0.
  - https://linux.die.net/man/2/socket

FUNKCIA BIND <SYS/SOCKET.H>

- int bind(int socket, const struct sockaddr *address, socklen_t address_len)
  - funkcia priradí socketu adresu (je to dôležité v prípade, ak chceme na príslušnom sockete prijímať spojenia), pričom v prípade úspechu vráti hodnotu 0 inak hodnotu -1 (a nastaví hodnotu globálnej premennej errno);
  - parametrami funkcie sú:
    - socket – platný popisovač socketu;
    - address – ukazovateľ na štruktúru sockaddr, ktorá nesie informáciu o adrese, na ktorú má byť socket priradený; veľkosť a formát závisia od použitej komunikačnej domény. V prípade domény **AF_INET** použijeme nasledujúcu štruktúru:

```
struct sockaddr_in
{
    sa_family_t sin_family;      //AF_INET alebo AF_INET6
    unsigned short sin_port;     //port (musí byť uložený ako big-endian)
    struct in_addr sin_addr;     //IPv4 adresa
};

struct in_addr
{
    unsigned long s_addr;        //4 bajty predstavujúce IP adresu
};                               //užitočná konštanta: INADDR_ANY
```

    - address_len – veľkosť štruktúry, na ktorú ukazuje argument address.
  - https://linux.die.net/man/2/bind

Funkcia listen <sys/socket.h>

- int listen(int sockfd, int backlog)
  - funkcia označí príslušný socket ako pasívny, čo znamená, že tento socket môže byť použitý na prijatie spojenie od klienta, pričom v prípade úspechu vráti hodnotu 0 inak hodnotu -1 (a nastaví hodnotu globálnej premennej errno);
  - parametrami funkcie sú:
    - sockfd – socket, na ktorom sa budú prijímať spojenia;
    - backlog – maximálna dĺžka frontu žiadostí o spojenie, ktoré ešte neboli spracované;
  - pasívne sockety nie je možné používať na výmenu dát medzi procesmi, ale len na prijatie požiadaviek o spojenie
  - https://linux.die.net/man/2/listen

Funkcia accept <sys/socket.h>

- int accept(int sockfd, struct sockaddr *addr, socklen_t *addrlen)
  - pomocou funkcie čaká server na požiadavku vytvorenia spojenia s klientom, pričom v prípade úspechu vráti hodnotu 0 inak hodnotu -1 (a nastaví hodnotu globálnej premennej errno);
  - parametrami funkcie sú:
    - sockfd – socket, pomocou ktorého chceme komunikovať so serverom;
    - addr – ukazovateľ na štruktúru sockaddr, ktorá nesie informácie o adrese servera, na ktorý sa chceme pripojiť;
    - address_len – veľkosť štruktúry, na ktorú ukazuje argument addr
  - https://linux.die.net/man/2/connect

Funkcia connect <sys/socket.h>

- int connect(int sockfd, const struct sockaddr *addr, socklen_t addrlen)
  - pomocou funkcie nadväzuje klient spojenie so serverom. Funkcia je blokovacia, pričom vracia popisovač socketu, reprezentujúci spojenie s príslušným klientom (v prípade chyby vráti hodnotu -1 a nastaví hodnotu globálnej premennej errno);
  - parametrami funkcie sú:
    - sockfd – socket, na ktorom sa prijímajú spojenia;
    - addr – ukazovateľ na štruktúru sockaddr, do ktorej sa uložia informácie o adrese klienta po úspešnom nadviazaní spojenia;
    - address_len – veľkosť štruktúry, na ktorú ukazuje argument addr
  - https://linux.die.net/man/2/accept

Funkcie pre posielanie správ

- <unistd.h>:
  - ssize_t **write**(int fd, const void *buf, size_t count)
  - ssize_t **read**(int fd, void *buf, size_t count)
    - sockety typu SOCK_STREAM
    - https://linux.die.net/man/2/write
    - https://linux.die.net/man/2/read

- <sys/socket.h>:
  - ssize_t **send**(int sockfd, const void *buf, size_t len, int flags)
  - ssize_t sendto(int sockfd, const void *buf, size_t len, int flags, const struct sockaddr *dest_addr, socklen_t addrlen)
  - ssize_t sendmsg(int sockfd, const struct msghdr *msg, int flags)
    - https://linux.die.net/man/2/send

  - ssize_t **recv**(int sockfd, void *buf, size_t len, int flags)
  - ssize_t recvfrom(int sockfd, void *buf, size_t len, int flags, struct sockaddr *src_addr, socklen_t *addrlen)
  - ssize_t recvmsg(int sockfd, struct msghdr *msg, int flags)
    - https://linux.die.net/man/2/recv

Funkcia close <unistd.h>

- int close(int fd):
  - funkcia uzatvára súbor, na ktorý ukazuje popisovač fd, pričom v prípade úspechu vráti hodnotu 0 inak hodnotu -1 (a nastaví hodnotu globálnej premennej errno);
  - funkcia sa používa aj pre uzatvorenie socketu;
  - https://linux.die.net/man/2/close

Užitočné funkcie pre prácu so socketmi (1)

- <netdb.h>
  - **struct hostent *gethostbyname(const char *name)**
    - funkcia vracia smerník na štruktúru typu hostent pre daný hostname
    - parametre funkcie:
      - name – názov hostiteľa alebo jeho IP adresa
  - struct hostent *gethostbyaddr(const void *addr, socklen_t len, int type)
    - funkcia vracia smerník na štruktúru typu hostent pre danú štruktúru typu in_addr
    - parametre funkcie:
      - addr – smerník na štruktúru typu in_addr
      - len – veľkosť štruktúry na ktorú ukazuje argument addr
      - type – typ komunikačnej domény (AF_INET)
  - obe funkcie vrátia v prípade neúspechu hodnotu NULL, pričom vhodne nastavia globálnu premennú h_errno
  - štruktúra hostent je definovaná nasledovne:

```
struct hostent {
    const char *h_name;          //oficiálny názov hostiteľa
    char **h_aliases;            //zoznam aliasov ukončený hodnotou NULL int
    h_addrtype;                  //typ adresy hostiteľa (AF_INET)
    int h_length;                //veľkosť adries v bajtoch (4 pre IPv4)
    char **h_addr_list;          //zoznam IP adries hostiteľa
    #define h_addr h_addr_list[0] //IP adresa hostiteľa
};
```

  - https://linux.die.net/man/3/gethostbyname

Užitočné funkcie pre prácu so socketmi (2)

- <netdb.h>
  - int getaddrinfo(const char *node, const char *service, const struct addrinfo *hints, struct addrinfo **res);
  - void freeaddrinfo(struct addrinfo *res);
    - https://linux.die.net/man/3/getaddrinfo
  - int getnameinfo(const struct sockaddr *sa, socklen_t salen, char *host, size_t hostlen, char *serv, size_t servlen, int flags);
    - https://linux.die.net/man/3/getnameinfo

Internetové sockety a endianita

- Problémy s endianitou:
  - x86 architektúra – little-endian
  - sieťové protokoly – big-endian („network byte order“)
- Riešenie – konverzné funkcie:
  - <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);
    - https://linux.die.net/man/3/htons

Ďalšie materiály

- moodle:
  - ukážka práce so socketmi typu SOCK_STREAM:
    - s – synchrónna komunikácia klient server
    - a – asynchrónna komunikácia klient server s využitím funkcie select (https://linux.die.net/man/2/select)
    - a_t – asynchrónna komunikácia s využitím funkcie select a vlákien
- Základy práce so socketmi:
  - http://kifri.fri.uniza.sk/~chochlik/frios/frios/sk/cvicenia/sockety.html
- Podrobný sprievodca internetovými socketmi:
  - http://www.beej.us/guide/bgnet/output/html/singlepage/bgnet.html