Systémové funkcie

- Deskriptory súborov
- I/O funkcie
- Iné funkcie pre prácu so súbormi a adresármi

Deskriptory súborov

- Keď proces otvori súbor (či už na čítanie, zápis, ...), dostane na tento súbor deskriptor
- Deskriptor je odkaz na štruktúry v jadre systému, pomocou ktorého sa bude k súboru pristupovať.
- Všetky deskriptory, ktoré proces vlastní, buď zdedil od svojich rodičov alebo tieto deskriptory získal iným systémovým volaním.
- Deskriptor na regulárny súbor sa da získať napríklad systémovým volaním „open“

Deskriptory súborov

- Všetky deskriptory, ktoré proces má, sú uložené v **tabuľke deskriptorov**, ktorá je súčasťou Kontextu procesu.
- Proces potom nepristupuje priamo do tabuľky deskriptorov ale iba cez index.
- Manipulovať s deskriptormi v tabuľke deskriptorov sa dá iba pomocou systémových volaní, ktorým sa ako argument pošle index do tabuľky deskriptorov.

Deskriptory súborov

- Typická tabuľka deskriptorov môže vyzerať takto:

index	súbor	poznámka
0	/dev/ttypn	stdin
1	/dev/ttypn	stdout
2	/dev/ttypn	stderr
3	./subor1	
4	./subor2	

Deskriptory súborov

- V všeobecnosti môže byť na prvých troch pozíciách v tabuľke deskriptorov akýkoľvek iný deskriptor.

index	súbor	poznámka
0	./subor0	stdin
1	./subor1	stdout
2	./subor2	stderr

LL (Low-Level) I/O funkcie

- Linux poskytuje dva druhy vstupno/výstupných funkcií, sú to:

  - knižničné funkcie: printf, fopen, atď.

  - low level (LL) I/O funkcie, ktoré sú systémové volania jadra
- Knižničné funkcie sú implementované pomocou LL I/O funkcií.

LL I/O funkcie

- LL funkcie pracujú s deskriptorom na rozdiel od knižničných funkcií, ktoré pracujú so smerníkom na súbor typu FILE.
- Deskriptor môže reprezentovať:

  súbor

  hardvérové zariadenie

  schránku

  kanál, atď
- Hlavičkové súbory pre základné I/O funkcie sú:

  <fcntl.h>, <sys/types.h>, <sys/stat.h>, <unistd.h>

Otvorenie súboru

- Funkcia int open(char* filename, int flag, mode_t mode), kde

  filename – cesta k súboru, ktorý sa otvára resp vytvorí

  flag – spôsob otvorenia je predefinovaný v fcntl.h nasledovne:

  O_RDONLY, O_WRONLY, O_RDWR alebo O_APPEND (pridanie na koniec), O_TRUNC (prepísanie súboru), O_CREAT (vytvorenie nového súboru), O_EXCL (s O_CREAT ak súbor už existuje vracia chybu)

  mode - prístupové práva pre skupinu a ostatných
- Funkcia vracia deskriptor súboru alebo –1 ak je chyba

Príklad

Create a New File

```
#include <fcntl.h>
#include <stdio.h>
#include <sys/stat.h>
#include <sys/types.h>
#include <unistd.h>
int main (int argc, char* argv[]){
  /* The path at which to create the new file. */
  char* path = argv[1];
  /* The permissions for the new file. */
  mode_t mode = S_IRUSR | S_IWUSR | S_IRGRP | S_IWGRP | S_IROTH;
  /* Create the file. */
  int fd = open (path, O_WRONLY | O_EXCL | O_CREAT, mode);
  if (fd == -1) {
    /* An error occurred. Print an error message and bail. */
   perror (“open”);
   return 1;
  }
  return 0;
}
```

Uzavretie súboru

- int close (int fd), kde argument je deskriptor súboru
- ak sa nepoužije close otvorený súbor sa automaticky po skončení programu uzavrie
- ak je fd deskriptor komunikačnej schránky po uzavretí schránky pre komunikáciu po sieti sa spojenie preruší
- linux ohraničuje počet otvorených súborov procesu na 1024. Otvorené deskriptory používajú prostriedky jadra a preto je vhodné po skončení práce s nimi ich uzavrieť

Zápis údajov

- Pre zápis sa používa funkcia

  int write(int fd, char *buffer, unsigned length)

  zapisuje údaje z bufra dĺžky length do deskriptora súboru
- write kopíruje ľubovolný obsah bytov bufra do deskriptora
- funkcia vracia počet zapísaných bytov alebo -1

Príklad

Append a Timestamp to a File

```
#include <fcntl.h>
#include <stdio.h>
#include <string.h>
#include <sys/stat.h>
#include <sys/types.h>
#include <time.h>
#include <unistd.h>
/* Return a character string representing the current date and time. */
char* get_timestamp ()
{
  time_t now = time (NULL);
  return asctime (localtime (&now));
}
```

Príklad

```
int main (int argc, char* argv[])
{
  /* The file to which to append the timestamp. */
  char* filename = argv[1];
  /* Get the current timestamp. */
  char* timestamp = get_timestamp ();
  /* Open the file for writing. If it exists, append to it; otherwise, create a new file. */
  int fd = open (filename, O_WRONLY | O_CREAT | O_APPEND, 0666);
  /* Compute the length of the timestamp string. */
  size_t length = strlen (timestamp);
  /* Write the timestamp to the file. */
  write (fd, timestamp, length);
  /* All done. */
  close (fd);
  return 0;
}
```

Výstup

```
% ./timestamp tsfile
% cat tsfile
Thu Feb 1 23:25:20 2001
% ./timestamp tsfile
% cat tsfile
Thu Feb 1 23:25:20 2001
Thu Feb 1 23:25:47 2001
```

Použité časové funkcie

- time_t time (time_t *)

  základná funkcia. Vracia počet sekúnd od 1.1.1970 po jej realizáciu. Od nej sú odvodené ďalšie, použitelnejšie funkcie
- struct tm * localtime (time_t *)

  Transformuje počet sekúnd do štruktúry tm (rok, mesiac,,deň, hodiny, minúty,....)
- char * asctime (struct tm *)

  robí reťazcové konverzie

Čítanie údajov

- Funkcia read

  size_t read(int fd, char *buffer, unsigned length)

  číta údaje z deskriptora fd súboru do bufra, počet načítaných bytov je v length.
- funkcia vracia počet načítaných bytov alebo –1
- Nasledovný program číta hexadecimálny obsah pamäte pre súbor daný ako argument programu a vytlačí ho.

Príklad

Print a Hexadecimal Dump of a File

```
#include <fcntl.h>
#include <stdio.h>
#include <sys/stat.h>
#include <sys/types.h>
#include <unistd.h>
int main (int argc, char* argv[])
{
  unsigned char buffer[16];
  size_t offset = 0;
  size_t bytes_read;
  int i;
  /* Open the file for reading. */
  int fd = open (argv[1], O_RDONLY);
```

Príklad

```
/* Read from the file, one chunk at a time. Continue until read“comes up short”, that is,
    reads less than we asked for. This indicates that we’ve hit the end of the file. */
do {
/* Read the next line’s worth of bytes. */
bytes_read = read (fd, buffer, sizeof (buffer));
/* Print the offset in the file, followed by the bytes themselves. */
printf (“0x%06x : “, offset);
for (i = 0; i < bytes_read; ++i)
printf (“%02x “, buffer[i]);
printf (“\n”);
/* Keep count of our position in the file. */
offset += bytes_read;
}
while (bytes_read == sizeof (buffer));
/* All done. */
close (fd);
return 0;
}
```

Výstup

```
% ./hexdump hexdump
0x000000 : 7f 45 4c 46 01 01 01 00 00 00 00 00 00 00 00 00
0x000010 : 02 00 03 00 01 00 00 00 c0 83 04 08 34 00 00 00
0x000020 : e8 23 00 00 00 00 00 00 34 00 20 00 06 00 28 00
0x000030 : 1d 00 1a 00 06 00 00 00 34 00 00 00 34 80 04 08
...
```

Pohyb v súbore

- Funkcia lseek umožňuje meniť pozíciu v súbore

  off_t lseek(int fd, off_t offset, int position)

  kde fd je deskriptor súboru

  offset je počet bytov môže byť aj záporné číslo

  position má hodnoty:

  SEEK_SET – druhý argument interpretuje pozíciu v bytoch od začiatku súboru

  SEEK_CUR – druhý argument môže byť kladný alebo záporný a je offset od aktuálnej polohy v súbore

  SEEK_END – druhý arg. interpretuje ako pozíciu od konca súboru, ak je kladný pozícia je za koncom súboru
- Funkcia vracia novú polohu ako offset od začiatku súboru

- lseek umožňuje zvoliť pozíciu v súbore za jeho koncovú pozíciu
- pri zápise do súboru sa súbor najprv zväčší a dáta sa zapíšu po koniec
- linux si udržuje informáciu o zväčšenej dĺžke súboru hoci aktuálne sa na disku rozšírenie nenachádza
- pri výpise obsahu súboru sa v rozšírenej časti súboru vypíšu 0 byty

Príklad

```
/*  ukazka pouzitia volania lseek. Vytvorime si databazu
    uzivatelov a zapisujeme do nej ich UID a mena. Funckia
    putrec zarucuje, ze bude kazda polozka zaradena na spravne
    miesto*/
#include <fcntl.h>
struct record {
  int uid;
  char login[9];
};
char *logins[] = { "user1", "user2", "user3","user4", "user5" };
```

Príklad

```
main(){
int i, fd;
struct record rec;
/* Otvorime datovy subor na zapis... */
if ((fd = open("datafile", O_WRONLY | O_CREAT, 0644)) < 0) {
    perror("datafile");
    exit(1);
}
for (i = 4; i >= 0; i--) {
/*          Vytvorime polozku pre uzivatela.          */
    rec.uid = i;
    strcpy(rec.login, logins[i]);
```

Príklad

```
/*          Zapiseme tuto polozku na prislusne miesto. */
    putrec(fd, i, &rec);
   }
    close(fd);
    exit(0);
}
/*   putrec - tato funkcia zapise polozku na i-tu poziciu do suboru.*/
putrec(int fd, int i, struct record *r){
/*      Nastavime sa na i-tu polozku v subore.     */
    lseek(fd, (long) i * sizeof(struct record), SEEK_SET);
    write(fd, r, sizeof(struct record));
}
```

Systémové volanie dup

- Umožní ako štandardný vstup/výstup použiť súbor
- Systémové volanie **dup(int fd)** zduplikuje deskriptor, ktorý dostane ako argument a duplikát uloží na **prvú volnú pozíciu** v tabuľke deskriptorov.
- Zduplikovanie deskriptoru neznamená, že sa znova otvorí ten istý súbor. Súbor ostane otvorený iba raz.
- Dôsledok: ostane jediný ukazovateľ na aktuálnu pozíciu v súbore.

Príklad

```
#include <fcntl.h>
#include <sys/stat.h>
int main(int argc, char **argv)
{
  int des1;
  int des2;
  des1=open("subor1" , O_CREAT | O_WRONLY , S_IRUSR |
     S_IWUSR);                                    //vytvorime / otvorime subor
  des2=dup(des1)                                 //zduplikujeme deskriptor
  write(des1,"Toto bude v subore\n",19);      //zapiseme do neho
  write(des2,"Toto tam bude tiez\n",19);      //a este raz
  close(des1);                          //zatvorime prvy deskriptor
  close(des2);                          //a aj druhy deskriptor
  return 0;
}
```

Ako zabezpečiť aby bol súbor štandardný pre I/O

- Zatvoriť v tabuľke deskriptorov (TD) príslušný štandardný deskriptor (sú v tabuľke na prvých troch miestach), tým sa uvolní miesto v TD
- Funkciou dup zduplikovať deskriptor súboru, čím sa duplikát uloží na prvé volné miesto v TD

Príklad

```
#include <fcntl.h>
#include <sys/stat.h>
int main(int argc, char **argv)
{
  if(argc>=2)                    //ak mame aspon jeden argument
  {
     int des;
   des=open(argv[1] , O_CREAT | O_WRONLY , S_IRUSR | S_IWUSR);
   close(1);
   dup(des);   // vymenime standardny vystup za subor, ktory je ako prvy
                        //arg. progr.
   close(des);
  }
```

Príklad pokr.

```
  write(1,"Nejaky ten vypis...\n",20);        //vypisy pojdu bud //na povodny
                                              //standardny vystup alebo do suboru.
  return 0;
}
```

- Ak dostane program ako argument názov súboru, pôjdu všetky jeho výpisy do tohoto súboru, ak nedostane žiaden argument, pôjdu jeho výpisy na štandardný výstup.

- Tabuľka deskriptorov bude teraz vyzerať takto:

index	súbor	poznámka
0	/dev/ttypn	stdin
1	./subor	stdout
2	/dev/ttypn	stderr

Získanie informácií o súbore

- Funkcia
  int lstat(char *path, struct stat *buf),
  získa informácie o súbore danom v path
- Hlavičkové súbory sys/types.h, sys/stat.h
  štruktúra stat:

```
struct stat {
    mode_t st_mode;      /* File mode (type, perms) */
    ino_t st_ino;        /* Inode number */
    dev_t st_dev;        /* ID of device containing */
                         /* a directory entry for this file */
    dev_t st_rdev;       /* ID of device */
    nlink_t st_nlink;    /* Number of links */
```

Získanie informácií o súbore

```
 uid_t st_uid;          /* User ID of the file's owner */
 gid_t st_gid;          /* Group ID of the file's group */
 off_t st_size;         /* File size in bytes */
 time_t st_atime;       /* Time of last access */
 time_t st_mtime;       /* Time of last data modification */
 time_t st_ctime;       /* Time of last file status change */
 /* Times measured in seconds since */
 /* 00:00:00 UTC, Jan. 1, 1970 */
 long st_blksize;       /* Preferred I/O block size */
 blkcnt_t st_blocks;  /* Number of 512 byte blocks allocated*/
}
```

Získanie informácií o súbore

- Nasledovné makrá testujú hodnoty st_mode:

  S_ISBLK (mode) block device

  S_ISCHR (mode) character device

  S_ISDIR (mode) directory

  S_ISFIFO (mode) fifo (named pipe)

  S_ISLNK (mode) symbolic link

  S_ISREG (mode) regular file

  S_ISSOCK (mode) socket

Získanie informácií o súbore

- Keď je už súbor pre čítanie a zápis otvorený miesto lstat sa volá funkcia

  int fstat(int fd, struct stat *buf), kde

  fd je deskriptor súboru
- Nasledovný program alokuje bufer a potom načíta obsah súboru do bufra

Príklad

Read a File into a Buffer

```
#include <fcntl.h>
#include <stdio.h>
#include <sys/stat.h>
#include <sys/types.h>
#include <unistd.h>
/* Read the contents of FILENAME into a newly allocated buffer. The size of the
    buffer is stored in *LENGTH. Returns the buffer, which the caller must free. If
    FILENAME doesn't correspond to a regular file, returns NULL. */
char* read_file (const char* filename, size_t* length)
{
  int fd;
  struct stat file_info;
  char* buffer;
  /* Open the file. */
  fd = open (filename, O_RDONLY);
```

Príklad

```
 /* Get information about the file. */
 fstat (fd, &file_info);
*length = file_info.st_size;
 /* Make sure the file is an ordinary file. */
 if (!S_ISREG (file_info.st_mode)) {
   /* It's not, so give up. */
   close (fd);
   return NULL;
 }
 /* Allocate a buffer large enough to hold the file's contents. */
 buffer = (char*) malloc (*length);
 /* Read the file into the buffer. */
 read (fd, buffer, *length);
 /* Finish up. */
 close (fd);
 return buffer;
}
```

Funkcie pre prácu so súbormi

Hlavičkové súbory funkcií: stdio.h, unistd.h, sys/types.h, sys/stat.h

- Nasledovné funkcie sú ekvivalentné so shellovskými príkazmi pre prácu so súbormi:

  int remove(const char *path);

  int rename(const char *old, const char *new);

  Používajú systémové volania (unistd.h):

  int unlink(cons char *path) – vymaže položku v adresári (nebude vyditelná s ls), ak súbor používa iný proces obsah súboru sa z disku nevymaže

  int link(const char *path1, const char *path2)

  int chmod(char *path, mode_t mode)

  mode – určuje prístupové práva (trojmiestne oktálne číslo)

Testovanie prístupu k súboru

- Umožňuje funkcia

  int access(char* filen, int mode), kde mode môže mať hodnoty:

  R_OK – prístup čítania

  W_OK – prístup zápisu

  X_OK – prístup výkonu

  F_OK – existencia súboru
- Návratová hodnota je 0 ak OK alebo –1 ak chyba a errno je rovné preddef. hodnotám

Príklad

Check File Access Permissions

```
#include <errno.h>
#include <stdio.h>
#include <unistd.h>
int main (int argc, char* argv[])
{
  char* path = argv[1];
  int rval;
  /* Check file existence. */
  rval = access (path, F_OK);
  if (rval == 0)
  printf (“%s exists\n”, path);
  else {
  if (errno == ENOENT)
  printf (“%s does not exist\n”, path);
  else if (errno == EACCES)
  printf (“%s is not accessible\n”, path);
  return 0;
  }
```

Príklad

```
  /* Check read access. */
  rval = access (path, R_OK);
  if (rval == 0)
  printf (“%s is readable\n”, path);
  else
  printf (“%s is not readable (access denied)\n”, path);
  /* Check write access. */
  rval = access (path, W_OK);
  if (rval == 0)
  printf (“%s is writable\n”, path);
  else if (errno == EACCES)
  printf (“%s is not writable (access denied)\n”, path);
  else if (errno == EROFS)
  printf (“%s is not writable (read-only filesystem)\n”, path);
  return 0;
}
```

Rýchly prenos súborov

- Funkcia sendfile umožňuje efektívne prenášať súbory pomocou ich deskriptorov.
- Pritom môže deskriptor špecifikovať súbory, schránky alebo iné zariadenia.
- Prototyp funkcie:

  int sendfile (ind fdw, int fdr, off_t* offset, int size), kde

  fdw – deskriptor pre zápis

  fdr – deskriptor pre čítanie

  offset – miesto v súbore, od ktorého sa začína čítať, 0 od začiatku

  size – veľkosť v bytoch

Príklad

File Copy Using *sendfile*

```
#include <fcntl.h>
#include <stdlib.h>
#include <stdio.h>
#include <sys/sendfile.h>
#include <sys/stat.h>
#include <sys/types.h>
#include <unistd.h>
int main (int argc, char* argv[])
{
  int read_fd;
  int write_fd;
  struct stat stat_buf;
  off_t offset = 0;
```

Príklad

```
  /* Open the input file. */
  read_fd = open (argv[1], O_RDONLY);
  /* Stat the input file to obtain its size. */
  fstat (read_fd, &stat_buf);
  /* Open the output file for writing, with the same permissions as the source file. */
  write_fd = open (argv[2], O_WRONLY | O_CREAT, stat_buf.st_mode);
  /* Blast the bytes from one file to the other. */
  sendfile (write_fd, read_fd, &offset, stat_buf.st_size);
  /* Close up. */
  close (read_fd);
  close (write_fd);
  return 0;
}
```

fsync a fdatasync

- Cash pamäť urýchľuje prácu s I/O operáciami na disku. Pri náhlom spadnutí systému sa informácie z cash pamäti strácajú
- Funkcie triedy fsync() zapisujú všetky údaje zo súboru, ktorého deskriptor je argumentom funkcií okamžite na disk
- Návrat z funkcie fsync je až keď sú údaje fyzicky na disku
- Funkcia fsync garantuje zaznamenanie modifikácie súboru, funkcia fdatasync nie
- Ak je súbor otvorený s flagom O_SYNC funkcia fsync zabezpečí synchrróny I/O

Príklad

```
#include <fcntl.h>
#include <string.h>
#include <sys/stat.h>
#include <sys/types.h>
#include <unistd.h>
const char* journal_filename = “journal.log”;
void write_journal_entry (char* entry)
{
  int fd = open (journal_filename, O_WRONLY | O_CREAT |
     O_APPEND, 0660);
  write (fd, entry, strlen (entry));
  write (fd, “\n”, 1);
  fsync (fd);
  close (fd);
}
```

Funkcie pre prácu s adresármi Hlavičkový súbor: unistd.h

- int chdir(char *path)
- int getcwd(char* buffer, int length)
- int mkdir(char *path, mode_t mode) ,
  mode ako vo funkcii open určuje prístupové práva
- int rmdir(char* path)
- mode – určuje prístupové práva (trojmiestne oktálne číslo)

Výpis obsahu adresára

- Linux poskytuje funkcie pre čítanie obsahu adresárov. Tieto funkcie nepatria medzi L-L I/O

  Postup:
- Adresár treba najskôr otvoriť funkciou

  DIR* opendir(char* path)
- Adresér sa číta funkciou

  dirent* readdir(DIR* dir)
- Uzavretie adresára funkciou

  closedir(DIR* dir)
- Hlavičkové súbory sú <sys/types.h> a <dirent.h>

Príklad

Print a Directory Listing

```
#include <assert.h>
#include <dirent.h>
#include <stdio.h>
#include <string.h>
#include <sys/stat.h>
#include <sys/types.h>
#include <unistd.h>
/* Return a string that describes the type of the file system entry PATH. */
const char* get_file_type (const char* path)
{
  struct stat st;
  lstat (path, &st);
  if (S_ISLNK (st.st_mode))
  return “symbolic link”;
```

Príklad

```
  else if (S_ISDIR (st.st_mode))
  return “directory”;
  else if (S_ISCHR (st.st_mode))
  return “character device”;
  else if (S_ISBLK (st.st_mode))
  return “block device”;
  else if (S_ISFIFO (st.st_mode))
  return “fifo”;
  else if (S_ISSOCK (st.st_mode))
  return “socket”;
  else if (S_ISREG (st.st_mode))
  return “regular file”;
  else
  /* Unexpected. Each entry should be one of the types above. */
  assert (0);
}
```

Príklad

```
int main (int argc, char* argv[])
{
  char* dir_path;
  DIR* dir;
  struct dirent* entry;
  char entry_path[PATH_MAX + 1];
  size_t path_len;
  if (argc >= 2)
  /* If a directory was specified on the command line, use it. */
  dir_path = argv[1];
  else
  /* Otherwise, use the current directory. */
  dir_path = ".";
```

Príklad

```
/* Copy the directory path into entry_path. */
strncpy (entry_path, dir_path, sizeof (entry_path));
path_len = strlen (dir_path);
/* If the directory path doesn't end with a slash, append a slash. */
if (entry_path[path_len - 1] != '/') {
entry_path[path_len] = '/';
entry_path[path_len + 1] = '\0';
++path_len;
}
/* Start the listing operation of the directory specified on the
command line. */
dir = opendir (dir_path);
```

Príklad

```
  /* Loop over all directory entries. */
  while ((entry = readdir (dir)) != NULL) {
    const char* type;
    /* Build the path to the directory entry by appending the entry name to the path
       name. */
    strncpy (entry_path + path_len, entry->d_name,
    sizeof (entry_path) - path_len);
    /* Determine the type of the entry. */
    type = get_file_type (entry_path);
    /* Print the type and path of the entry. */
    printf ("%-18s: %s\n", type, entry_path);
  }
  /* All done. */
  closedir (dir);
  return 0;
}
```

Výstup

```
% ./listdir /dev
directory            : /dev/.
directory            : /dev/..
socket               : /dev/log
character device : /dev/null
regular file         : /dev/MAKEDEV
fifo                 : /dev/initctl
character device : /dev/agpgart
...
```

Skenovanie a usporiadanie v adresáry

- Platforma BSD obsahuje dve ďalšie funkcie

  int scandir(char *dir, struct dirent ***namelist, int(*filter)( ), int(*compar)( ))

  int alphasort(void *a, void *b) – abecedne usporiada, môže byť použitá ako compar() funkcia
- scandir() skenuje adresár dir s filtrom daným v 3 argumente a usporiadaným funkciou compar(),

  zoznam skenovania je v poli namelist
- funkcia vracia počet vybraných položiek v adresáry

Príklad – výpis obsahu adresára v opačnom poradí

```
#include <dirent.h>
    main(){
        struct dirent **namelist;
        int n;
        n = scandir(".", &namelist, 0, alphasort);
        if (n < 0)
            perror("scandir");
        else {
            while(n--) {
                printf("%s\n", namelist[n]->d_name);
                free(namelist[n]);
            }
            free(namelist);
        }
    }
```

- Ako príklad funkcii filtra() je nasledovná funkcia, ktorá hľadá súbory s rozšírením .c, .h

```
int file_select(struct dirent *entry)
{
 char *ptr;
 char *rindex(char *s, char c);
 if ((strcmp(entry->d_name, “.”) = = 0) ||
 (strcmp(entry->d_name, “..”) = = 0))
  return FALSE;
 ptr = rindex(entry->d_name, ‘.’)
```

```
    if ((ptr != NULL) && ((strcmp(ptr, ".c") = = 0) ||
    (strcmp(ptr, ".h") = = 0)
    return TRUE;
    else
    return FALSE;
}
```

- rindex() je funkcia, ktorá vracia posledný výskyt charakteru c v reťazci s