1. gyakorlat

Szerkesztés, fordítás

A félév feladatait C-ben fogjuk megoldani, de használhatnak C++ -t is. A C és a C++ között lesz néhány különbség, amelyet menet közben megemlítünk. A szerveren fogunk dolgozni. A kódot vagy a szerveren szerkesztjük a vi,joe,mc segítségével, vagy ftp-vel a kliens gépről másoljuk át. A fordításhoz a gcc vagy g++ fordítókat használjuk. Futtatáshoz ne felejtsünk el futási jogot adni a programnak.

int printf(const char *format [, argument]...);

Formátum sztringek megadása a kiíráshoz

Karakter	Argumentum típus	Nyomtatási kép
d,i	int	elõjeles decimális egész
o	int	elõjel nélküli oktális egész
x,X	int	hexadecimális egész
u	int	elõjel nélküli decimális egész
c	int	unsigned char-rá
s	char*	sztring karaktereit
f	double	[-]*m.dddddd*, ahol a *d*-k száma az elõírt pontosságtól függ
e,E	double	[-]*m.dddddd e± xx* vagy [-]*m.dddddd E± xx*,
g,G	double	ugyanaz, mint %e vagy %E, ha ugyanaz, mint %f.
p	void*	pointer-érték
%		nincs konverzió, maga a % karakter nyomtatódik

Javasolt feladat a gyakorlatra:

1. „Hello világ” program készítése

Oktatóknak

```
include file-ok: <stdlib.h> <stdio.h>

fordítás: gcc  fnev.c –Wall –o fnev   (C program fordítása)

          g++ fnev.cc –o fnev       (C++ fordítás)

debugg: gcc –g fnev.c –o fnev  //
           gdb ./file //r futtat,s leptet

futtatás: futtatási jog (chmod),  ./fnev
```

Szövegkezelés

A C-ben a szövegeket karakter tömbökben tároljuk. A szöveg végét 0 karakter jelzi. A tömb deklarálása történhet konstans szöveggel vagy dinamikus helyfoglalással. C-ben a helyfoglalást a malloc függvénnyel végezhetjük el.

```
char szoveg1[]=”Ez a szoveg” vagy char szoveg2[80]=”Ez a szoveg”

void *malloc(int size); //lefoglalás
void free(void*);                    //felszabadítás

char * szoveg3; szöveg3=(char*)malloc(80*sizeof(char))

int scanf(   const char *format [,   argument]...);
```

Készíthetünk saját szövegkezelő függvényeket, de használhatjuk a könyvtári szolgáltatásokat is.

```
size_t strlen( const char *string );
char *strcpy(char *strDestination,  const char *strSource );
char *strcat(char *strDestination, const char *strSource);
```

Javasolt feladat a gyakorlatra:

1. Készítsen programot, amelyik meghatározza egy szöveg hosszát
    a. konstans szöveg – 0 vég keresés, minden kódot main-ben
    b. „szöveghossz” függvény készítése 0 végjel keresésével
    c. szöveghossz meghatározása strlen függvény használatával
    d. beolvasott szöveg hosszának meghatározása

Oktatóknak:

A beolvasáshoz ne használjuk a gets (nem figyeli a hosszt!)

```
char *gets( char *buffer);
```

Mutató és referencia közötti különbség.

Röviden a mutató egy változó memóriacímét tartalmazza (helyfoglalás!), a referencia pedig az objektumra vonatkozó álnévként fogható fel . A C-ben referencia típusú paraméterátadásról még nem beszélhettünk, így ott a címszerinti paraméterátadást, az adott típusú változóra mutató mutatóval valósították meg. Figyeljük meg a main függvényt paraméterlistáját!

```
int main( int argc, const char *argv[] );
int main( int argc, const char *argv[],  char *envp[] );
```

Javasolt feladat a gyakorlatra:

```
1. Készítsen programot, amelyik a main paraméterlistájában beérkező
argumentumokban megcseréli az ott levő alma szót körtére.
```

Oktatóknak:

Figyeljünk arra, hogy a „körte” hosszabb, mint az „alma” szó!

2. gyakorlat

Alacsonyszintű I/O

Az állományok létrehozása, megnyitása, írása, olvasása, zárása rendszerhívásokon keresztül is megvalósítható. Az egyes fájlokra fájlleírókkal hivatkozhatunk, amelyek egész számok. A standard bement, kimenet és hiba a 0,1,2-es számokkal reprezentált. A be-, ki- és hibacsatorna átirányítható. Az open függvény végrehajtása után megkapjuk a fájlleírót(hiba esetén -1-et), amelyet a többi rendszerhívásnál felhasználhatunk.

```
int open(const char *name,   int mode, unsigned attrib);
```

O_RDONLY	nyitás csak olvasásra
O_WRONLY	nyitás csak írásra
O_RDWR	nyitás olvasásra és írásra
O_APPEND	Megadása esetén minden írási mûveletet megelõzõen a file-pozíció a file végére lesz állítva.
O_CREAT	Ha a file nem létezik, akkor létre kell hozni.
O_EXCL	Ha a file létezik és O_CREAT kérelem volt, hibával tér vissza.
O_TRUNC	Ha a file létezik, akkor levágja 0 hosszúságúra.
O_BINARY	A file-t bináris kezelési módban nyitja meg.
O_TEXT	A file-t szöveges kezelési módban nyitja meg.

Létrehozásnál az attrib értékei:

S_IWRITE	engedélyesés írásra
S_IREAD	engedélyezés olvasásra
S_READ | S_IWRITE	engedélyezés írásra és olvasásra

```
int read(int handle, void *buf, unsigned len);
int write(int handle, void *buf, unsigned len);
int close(int handle);
```

Hiba esetén errno változó ($?) tartalmazza a hibakódot. perror() függvény pedig a hibaüzenetet írja ki.

```
void perror(const char *s);
```

Javasolt feladat a gyakorlatra:

Készítsen programot, amelyik beolvassa egy file tartalmát. Használjon rendszerhívást!

Segítség a feladathoz:

```
int f; open(„fnev”,O_RDONLY) ;read(f,sor,hossz) //rendszerhívással
FILE *f; f=fopen(„fnev”,”r”);fgets(sor,hossz,f);fclose(f);
//folyamszerű
```

Könyvtárkezelés

A könyvtárkezeléshez használható függvényeket a dirent.h –ban találjuk meg.

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

DIR *opendir(const char * pathname);
int chdir(const char *pathname);
struct dirent *readdir(DIR *dirp);
int closedir(DIR *dirp);


ahol a

struct  dirent {
    ino_t    d_ino;
    off_t    d_off;
    unsigned short d_reclen;
    char     d_name[1]; /* first char of name
};

A változó deklarációja:
struct típus változó    // C++-ban elég lenne: típus változó
```

Struct szerkezettel deklarált változók a C-ben és a C++-ban kicsit különbözőek. A C-ben értéktípusú, összetett szerkezet, a C++-ban pedig egy objektum jön létre, ahol az adattagok publikus hozzáférésűek.

Az egyes könyvtárbejegyzések tulajdonságait a stat függvénnyel, a stat struct-on keresztül olvashatóak ki.

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

int stat(const char *restrict path, struct stat *restrict buf);

struct stat{
     dev_t     st_dev     Device ID of device containing file.
     ino_t     st_ino     File serial number.
     mode_t    st_mode    Mode of file (see below).
     nlink_t   st_nlink   Number of hard links to the file.
     uid_t     st_uid     User ID of file.
     gid_t     st_gid     Group ID of file.
     dev_t     st_rdev    Device ID (if file is character or block
                            special).
     off_t     st_size    For regular files, the 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 status change.
     blksize_t st_blksize A file system-specific preferred I/O block
                       size for this object. In some file system types,
                       this may vary from file to file.
     blkcnt_t  st_blocks  Number of blocks allocated for this object.
}
function S_ISDIR(m: Word  ):Boolean
```

Javasolt feladat a gyakorlatra:

1. Készítsen programot, amelyik a könyvtár tartalmát kilistázza (file és könyvtárnevek)

a. aktuális könyvtáré
b. hierarchikusan, tabulált kiírással

Segítség a feladathoz:
A hierarchikus kiíráshoz használjunk rekurzív függvényt, a tabulálást leíró formátum string legyen paraméter. Egyszerűbb a megoldás, ha a chdir-rel mindig az aktuális könyvtárba lépünk és nem tároljuk a teljes útvonalat.

Állományok attribútumai

Az egyes állományok attribútumait a stat struct-ton keresztül olvashatjuk ki. A tulajdonos, a csoport és a külvilág jogosultságait az st_mode-ból érhetjük el. A következő konstansok felhasználhatóak az értékek kiolvasásához.

File mode bits:

```
S_IRWXU     read, write, execute/search by owner
S_IRUSR     read permission, owner
S_IWUSR     write permission, owner
S_IXUSR     execute/search permission, owner
S_IRWXG     read, write, execute/search by group
S_IRGRP     read permission, group
S_IWGRP     write permission, group
S_IXGRP     execute/search permission, group
S_IRWXO     read, write, execute/search by others
S_IROTH     read permission, others
S_IWOTH     write permission, others
S_IXOTH     execute/search permission, others
S_ISUID     set-user-ID on execution
S_ISGID     set-group-ID on execution
S_ISVTX     on directories, restricted deletion flag
```

Az állomány módosításának idejének kiírásához használjuk a ctime függvényt.

```
#include <time.h>
char * ctime ( const time_t * timer )

A userek és csoportok nevének meghatározásához
#include <pwd.h>
#include <grp.h>

  struct passwd *getpwuid(uid_t uid);
  struct passwd {
        char *pw_name;    /* user's login name */
        char *pw_passwd;  /* no longer used */
        uid_t pw_uid;        /* user's uid */
        gid_t pw_gid;        /* user's gid */
        char *pw_age;     /* not used */
        char *pw_comment; /* not used */
        char *pw_gecos;   /* typically user's full name */
        char *pw_dir;     /* user's home dir */
        char *pw_shell;   /* user's login shell */
    };
  struct group *getgrgid(gid_t gid);

  struct group {
         char *gr_name;     /* the name of the group */
         char *gr_passwd;   /* the encrypted group password */
         gid_t gr_gid;      /* the numerical group ID */
         char **gr_mem;     /* vector of pointers to member names */
    };
```

Javasolt feladat a gyakorlatra:

Készítsen programot, amelyik az ls –al parancs eredményéhez hasonlót ír ki.

Segítség a feladathoz:

```
Készítsünk függvényeket az attribútumok meghatározásához, a usernevek
kiolvasásához stb.
char * atr(int a)
{
  char* atrb=(char*)malloc(12*sizeof(char));
  if (a & S_IFDIR){strcpy(atrb,"d");}else {strcpy(atrb,"-");}
  if (a & S_IRUSR){strcat(atrb,"r");}else {strcat(atrb,"-");}
  if (a & S_IWUSR){strcat(atrb,"w");}else {strcat(atrb,"-");}
      …….
}
```

3. gyakorlat

Folyamatok létrehozása

Új folyamatot a fork függvénnyel hozhatunk létre. A gyermek folyamat majdnem pontosan azonos a szülő folyamattal – a PID számuk azért különbözik! (A gyerek elérheti a szülő PID-jét) A szülő és gyerek folyamatok egymással párhuzamosan futnak.

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

pid_t fork(void); //gyerek folyamat létrehozása, -1, ha nem sikerült
pid_t getpid(void); //aktuális folyamat PID-je
pid_t getppid(void); //szülő folyamat PID-je

A szülő folyamat megvárhatja a gyerek folyamatot.

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

 pid_t wait(int *status);
 pid_t waitpid(pid_t pid, int *status, int options);
```

Javasolt feladat a gyakorlatra

Készítsünk egyszerű apa-fiú programot, amelyikben az egyes folyamatok kiírják a saját PID számukat és a szülő bevárja a gyereket.

Segítség a feladathoz:
Mi az, amit a gyermek-folyamat fork után a szülőtől örököl?

- *A folyamatot futtató felhasználóra vonatkozó információkat (a futtató felhasználó azonosítóját, a futtató felhasználó csoportjának az azonosítóját)*
- *Effektiv user id-et (ha a program setuid bites, akkor ez eltérhet a programot futtató felhasználó azonosítójától)*
- *Effektiv csoport azonosítót*
- *Folyamat-csoport azonosítóját*
- *Munkadirectory*
- *Signal-kezelo eljárások*
- *umask értéket (ld. később)*

Mi az, ami a fork után eltér a szülő és a gyermek között?

- *Folyamat-azonosító*
- *Szülo folyamat azonosítója*
- *A gyermek folyamatnak saját másolata van a szülo folyamat fájldeszkriptorjairól*
- *Ha a szülo valamikorra egy ALARM signalt kért, azt a gyermek nem fogja megkapni.*

A wait függvény helyett már a waitpid használata javasolt

A waitpid függvény hívásához:

The value of pid can be:

< -1 meaning wait for any child process whose process group ID is equal to the absolute value of pid.

-1 meaning wait for any child process.

0 meaning wait for any child process whose process group ID is equal to that of the calling process.

> 0 meaning wait for the child whose process ID is equal to the value of pid.

The value of options is an OR of zero or more of the following constants*:*

WNOHANG return immediately if no child has exited.
WUNTRACED also return if a child has stopped (but not traced via ptrace(2)). Status for traced children which have stopped is provided even if this option is not specified.
WCONTINUED (Since Linux 2.6.10) also return if a stopped child has been resumed by delivery of SIGCONT.

File-ok zárolása

Ha több folyamat használja ugyanazt a file-t és legalább egy írni is akar bele, zárolni kell. Írásra csak 1 folyamat zárolhatja. *Párhuzamos folyamatok kölcsönös kizárása az flock() vagy fcntl() rendszerhívásnál valósul meg!*

Az flock() rendszerhíváshoz a #include <sys/file.h> csatolás szükséges.
A függvény alakja: int flock(int fd, LOCK_SH vagy LOCK_EX vagy LOCK_UN) A close() oldja az flock-ot. flock -1-et ad eredményül, ha sikertelen a próbálkozás.

Az fcntl() használata kicsit összetettebb, a C rendszerkönyvtára valósítja meg, de az flock() hívással ellentétben használható NFS-en keresztül is, és az esetleges holtpont problémát is kezeli.

```
#include <sys/types.h>
#include <unistd.h>
#include <fcntl.h>
int fcntl(int fd, int cmd, ...);

struct flock {
   short l_type; // F_RDLCK,F_WRLCK, F_UNLCK
   short l_whence; // SEEK_SET,SEEK_CUR,  SEEK_END
   off_t l_start;
   off_t l_len;
   pid_t l_pid;
};
```

A cmd-k F_SETLKW (vár, amíg lezárhatja)

F_SETLK (nem vár, megy tovább, ha nem sikerült)
F_GETLK –kiolvassa az adatokat

Javasolt feladat a gyakorlatra

Készítsünk programot, amelyben a szülő és a gyerek folyamat is ugyanabba a file-ba ír egy-egy mondatot sokszor ismételve. (A mondatok különbözőek) Módosítsuk a feladatot - file zárolással oldjuk meg, hogy várják be, amíg a másik befejezi az írást.

Segítség a feladathoz:
Ki kell tölteni az flock struct minden elemét a hívásnál.

```
lockoláshoz: struct flock lock_data; beállítandó:
lock_data.l_start=0;lock_data.l_len=100000;
      lock_data.l_whence=SEEK_SET; lock_data.l_type=F_WRLCK; (írásra
lezárja)    //feoldáshoz F_UNLCK

rc=fcntl(f,F_SETLKW,&lock_data); //rc-ben sikerült
```

Folyamat szülőtől eltérő feladattal

A létrejött gyerek folyamatot helyettesíthetjük egy másik fájlban tárolt program végrehajtásával, az exec függvények egyikével. Az új végrehajtandó program „megkapja ”a régi folyamat memóriaterületét, tehát nem tér vissza a gyerek folyamat hívás utáni részébe.

```
int execl(char const *path, char const *arg0, ...);

int execle(char const *path, char const *arg0, ..., char const * const
*envp);

int execlp(char const *path, char const *arg0, ...);

int execlpe(char const *path, char const *arg0, ...);

int execv(char const *path, char const * const * argv);

int execve(char const *path, char const * const *argv, char const * const
*envp);

int execvp(char const*path, char const * const *argv);

int execvpe(char const *path, char const * const *argv, char const * const
*envp);
```

Javasolt feladat a gyakorlatra

Készítsünk egyszerű shell-ként működő programot. Olvassunk be egy parancsot, majd hajtsuk végre.

Segítség a feladathoz:
Figyeljünk arra, hogy az exec hívás ne kerüljön a szülő ágába!
`execv(cmd,arg); // char* arg[]={„”,””,…,NULL};`
int system(const char *command);`//parancsvégrehajtás, de visszatér a hívás helyére`