



Szerver

PuTTY:

- Host: os.inf.elte.hu
- login as: {neptun kód}
- password: {almafa1}

jelszó módosítás: passwd





hellow.c

- stdio.h: printf
- vim

```
#include <stdio.h>
int main()
{
    printf("Hello World!\n");
}
```





Fordítás és futtatás

- Fordítás:
 - gcc forrás.c → a.out
 - gcc forrás.c -o cél vagy -océl
- Futtatás:
 - ./cél





printf

stdio.h

printf({formátumstring}[, {változólista}])

escape szekvenciák:

- \n - új sor
- \t - tabulátor
- \" - idézőjel
- \' - aposztróf
- \0 - null karakter
- \r - kurzor a sor elejére
- \\ - backslash





printf

`printf({formátumstring}, {változólista})`

formátumleírók:

- **%d**
- **%i** - egész
- **%f** - valós
- **%c** - karakter
- **%s** - string





string

- null-terminált karaktersorozat:
 $\text{str2}[5] = '\backslash 0' \equiv \text{str2}[5] = 0;$
- deklaráció (statikus):
 - `char str2[80];`
 - `char str2[]="Hajra Vasas!";`





string.h

- strcpy
- strlen
- strcmp
- strcat





strcpy

strcpy({cél mutató}, {forrás mutató | konstans})

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

int main()
{
    char s1[20],s2[20],s3[20];
    strcpy(s1, "STRCPY forras");
    strcpy(s2, s1);
    strcpy(s3, &s1[0]);
    printf("s1=%s, s2=%s, s3=%s\n",s1,s2,s3);
}
```



mutató

* - „értéke” operátor

& - „címe” operátor

mutató típusú változó:

char *ptr_str;

mutató értékadás:

ptr_str = &str[0]

ptr_str = str;

mutató hivatkozás:

ptr_str

&ptr_str[0]

ptr_str+6

&ptr_str[6]

elemre hivatkozás:

ptr_str[0]

*ptr_str

ptr_str[11]

*(ptr_str+11)





mutató példa

```
#include <stdio.h>
```

```
int main()
```

```
{
```

```
    char s1[] = "Forras szoveg";
```

```
    char *s2, *s3;
```

```
    s2 = s1;
```

```
    s3 = &s1[0];
```

```
    printf("s1=%s, s2=%s, s3=%s\n",s1,s2,s3);
```

```
    s2 = s1+7;
```

```
    s3 = &s1[7];
```

```
    printf("s1=%s, s2=%s, s3=%s\n",s1,s2,s3);
```

```
}
```





Dinamikus memória foglalás string számára



stdlib.h

malloc:

{mutató típusú változó} =

 ({elemtípus}*)malloc({memóriaméret})

free:

free({mutató})





malloc példa

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

int main()
{
    char s1[] = "Forras";

    char *s2;
    s2 = malloc(10); //=(char*)malloc(10*sizeof(char))

    char *s3 = malloc(10);

    strcpy(s2, s1);
    strcpy(s3, s2);
    printf("s1=%s, s2=%s, s3=%s\n", s1, s2, s3);

    free(s2);
    free(s3);
}
```





Dinamikus memória foglalás string számára



stdlib.h

calloc:

{mutató típusú változó} =

 ({elemtípus}*)calloc({elemszám},{elem-memóriaméret})

realloc:

{mutató típusú változó} =

 ({elemtípus}*)realloc({mutató},{új memóriaméret})





string.h további függvények

- `strlen({mutató})`
- `strcmp({mutató1}, {mutató2})`
- `strcat({cél mutató}, {forrás mutató | konstans})`



Feladat

Csinálunk length függvényt!

- hívás: hossz = length(str);
- értéke: a string hossza (strlen)





length függvény

```
int length(char *str)
{
    int len = 0;
    while (str[len] != 0)
        len++;
    return len;
}
```





length függvény

Használjuk ki, hogy a paraméter egy mutató!

```
int length(char *pointer)
{
    int len = 0;
    while (*pointer != 0)
    {
        pointer++;
        len++;
    }
    return len;
}
```





length függvény

A mutató léptetését a ciklusfeltételeben is elvégezhetjük.

```
int length(char *pointer)
{
    int len = 0;
    while (*pointer++ != 0)
        len++;
    return len;
}
```





length függvény

Mivel az =0 a hamis a !=0 pedig az igaz, optimalizáljuk a ciklusfeltételt.

```
int length(char *pointer)
{
    int len = 0;
    while (*pointer++)
        len++;
    return len;
}
```





length függvény

Ha elmentjük a kezdő pointert, nem kell a len változó.

```
int length(char *pointer)
{
    char *start;
    start = pointer;
    while (*pointer++);
    return --pointer-start;
}
```





Paraméterek feldolgozása

```
int main(int argc, char **argv)
```

argv elemei:

- 0.: amit beírtunk a futtatáshoz: ./a.out, vagy ..//gy0/a.out
- 1..argc-1: a megadott paraméterek a megadásuk sorrendjében
- argc-edik paraméter: null





struktúra

típus deklaráció:

struct típusnév

{

mezőtípus1 mezőnév1;

...

mezőtípusN mezőnévN;

}





struktúra

változó deklaráció:

struct típusnév azonosító

mezőhivatkozás: azonosító.mezőnév

pointer változó deklaráció:

struct típusnév *pointerazonosító

mezőhivatkozás:

(*pointerazonosító).mezőnév

pointerazonosító->mezőnév





Hibakezelés

`<errno.h>`

`errno`

a legutóbb bekövetkezett hiba

`<stdlib.h>`

`perror({saját hibaüzenet}):`

hibaüzenet kiíratása. Formátuma:

"{saját hibaüzenet}: {errno szerinti üzenet}"





Fájlkezelés

<unistd.h>

access({fájl},{mód})

az aktuális felhasználó hozzáférési jogának ellenőrzése:

mód:

- R_OK: olvasási jog,
- W_OK: írási jog,
- X_OK: végrehajtási jog,
- F_OK létezés ellenőrzése.





Fájlkezelés stdio.h függvények használatával

stdio.h>

- FILE típus → FILE {pointer} → pl. FILE *f;
- fopen
- fclose
- fprintf (text)
- fputs (text)
- fgets (text)
- fwrite (bin)
- fread (bin)
- fseek
- feof



Megnyitás és lezárás

f=fopen({fájlnév},{mód})

megnyitás

fclose({fájl pointer})

lezárás





Szöveges (text) fájlkezelés

{mód} (fopen):

"r" - létező fájl olvasásra

"w" - új fájlt hoz létre írásra

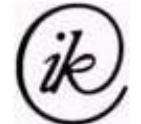
"a" - hozzáírásra

"r+" - létező fájl olvasásra és írásra

"w+" - új fájlt hoz létre olvasásra és írásra

"a+" - olvasásra és hozzáírásra





Szöveges (text) fájlkezelés

fputs({string},f)

írás fájlba

fprintf(f,{formátumstring},{változólista})

formázott írás fájlba

fgets({char[] mutató},{max. karakter},f)

olvasás fájlból

feof(f)

fájl végének lekérdezése





Bináris fájlkezelés



{mód}: (fopen)

"rb" - létező fájl olvasásra

"wb" - új fájlt hoz létre írásra

"ab" - hozzáírás

"rb+" - létező fájl olvasásra és írásra

"wb+" - új fájlt hoz létre olvasásra és írásra

"ab+" - olvasásra és hozzáírásra





Bináris fájlkezelés



fwrite({kezdőcím}, {elemméret}, {elemszám}, f)
írás

fread({kezdőcím}, {elemméret}, {elemszám}, f)
olvasás

Egy teljes tömb (t) tartalmának kiírása vagy
beolvasása, elemtípustól függetlenül:

fread|fwrite(t,sizeof(t[0]),sizeof(t)/sizeof(t[0]),f)

Egy változó (v) tartalmának kiírása vagy
beolvasása, elemtípustól függetlenül:

fread|fwrite(&v, sizeof(v), 1, f)





Bináris fájlkezelés

fseek(f,{elmozdulás},{viszonyítási pont})
pozícionálás

{viszonyítási pont}:

- SEEK_SET: file elejétől
- SEEK_CUR: mutató aktuális pozíciójától
- SEEK_END: file végétől





Fájlkezelés rendszerhívások használatával



{fileazonosító}=open({fájlnév},{mód})
{fileazonosító}=open({fájlnév},{mód},
{engedélyek})

megnyitás

close({fileazonosító})

lezárás:





open

{mód}:

- O_RDONLY: csak olvasásra
- O_WRONLY: csak írásra
- O_RDWR: írásra és olvasásra
- O_APPEND: hozzáfűzésre
- O_TRUNC: minden adatot töröl a fájlból
- O_CREAT: ha nem létezik, létrehozza
- O_EXCL: O_CREAT-tel kombinálandó, létre KELL hozni a fájlt





open

{engedélyek}: S_I[R|W|X][USR|GRP|OTH])

- S_IRUSR: a tulajdonos (owner) olvasási jogát igazra állítja
- S_IWUSR: a tulajdonos írási jogát igazra állítja
- S_IXUSR: a tulajdonos végrehajtási (execution) jogát igazra állítja
- S_IRGRP: a csoport (group) olvasási jogát igazra állítja
- S_IWGRP: a csoport írási jogát igazra állítja
- S_IXGRP: a csoport végrehajtási jogát igazra állítja
- S_IROTH: a többi felhasználó (other users) olvasási jogát igazra állítja
- S_IWOTH: a többi felhasználó írási jogát igazra állítja
- S_IXOTH: a többi felhasználó (other users) végrehajtási jogát igazra állítja





Fájlkezelés rendszerhívások használatával



read(f, {kezdőcím}, {olvasandó byte-ok})
olvasás

write(f, {kezdőcím}, {írandó byte-ok})
írás

lseek(f,{elmozdulás byte-ban},{viszonyítási pont})
pozícionálás

{viszonyítási pont}:

- SEEK_SET: file elejétől
- SEEK_CUR: mutató aktuális pozíójától
- SEEK_END: file végétől





Folyamatok létrehozása

<sys/types.h>

pid_t típus

pid_t fork()

létrehoz egy gyerekfolyamatot, mely csak a

- PID (process identification) és
- PPID (parent process identification)

értékekben tér el a szülőtől.

Értéke:

- a szülő folyamatban: a létrehozott gyerekfolyamat azonosítója, vagy -1 (hiba).
- a gyerek folyamatban: 0





Folyamatok kezelése



pid_t getpid()

a folyamat azonosítójának lekérdezése

pid_t getppid()

a folyamat szülőjének azonosítóját kérdezi le

sleep({másodperc})

usleep({mikroszekundum})

felfüggeszti a folyamat végrehajtását a megadott ideig, vagy egy szignál érkezéséig





Feladat



Hozzunk létre azonos szülőfolyamatból két gyermekfolyamatot. A szülő írja ki a két gyerek PID-jét, a gyerekek pedig a saját és a szülőjük PID-jét.

Tapasztalat: Összekeverednek a sorok!
Szinkronizálni kellene!





Szülő-gyerek PID-ek

- Gyerek:
 - pid változó: 0
 - PID: gyerek PID
 - PPID: szülő PID
- Szülő:
 - pid változó: gyerek PID
 - PID: szülő PID
 - PPID: bash (shell) PID

Szülő nem látja gyerek folyamatot, csak a pid változóból tudhatunk róla.





Folyamatok szinkronizálása

wait(&status)

Felfüggeszti a folyamat végrehajtását, míg bármelyik gyerekfolyamat terminál.





Folyamatok szinkronizálása



waitpid(PID, &status, options)

Felfüggeszti a folyamat végrehajtását, míg {PID} gyerekfolyamat állapota megváltozik.

options:

- WNOHANG – terminált (0)
- WUNTRACED – leállt (stopped)
- WCONTINUED – folytatódott (resumed)





Külső program végrehajtása

execv(parancs, argumentumok)

Lecseréli az aktuális folyamatot a meghívott folyamatra.





Külső program végrehajtása

(*system(parancssor)*)



Létrehoz egy gyerekfolyamatot, és
végrehajtja vele a *parancssor*-t.





Fájl zárolás

<fcntl.h>

struktúra: **flock**

- `l_type` (short) lezárás típusa:
 - `F_RDLCK` – olvasásra,
 - `F_WRLCK` – írásra,
 - `F_UNLCK` – feloldás.
- `l_whence` (short) a lock-olás kezdőpozíciójának viszonyítási pontja:
 - `SEEK_SET` – fájl elejétől,
 - `SEEK_CUR` – aktuális pozíciótól,
 - `SEEK_END` – fájl végétől.
- `l_start` (off_t) a lock-olás kezdőpozíciója.
- `l_len` (off_t) a lock-olt bájtok száma, 0: teljes fájl.
- `l_pid` (pid_t) a lock-oló folyamat PID-je.
- ...

Fájl zárolás

fcntl(fileazonosító, mód, &flock)

mód:

- **F_SETLKW** – beállítja a zárolást (flock szerint), vár, ha már zárolva van.
- **F_SETLK** – beállítja a zárolást, nem vár.





Véletlenszám generálás



`<stdlib.h>`

rand()

Értéke egy [0,RAND_MAX] intervallumba eső véletlenszám.

srand(kezdőérték)

Véletlenszámgenerátor kezdőértékének megadása.





Idő

`<time.h>`

`time_t time(NULL)`

Értéke az aktuális rendszeridő.

`localtime(&{time_t})`

Átkonvertálja `time_t` típusú időt `tm` típusúra.



Idő

struct tm

- **tm_sec** int másodperc (0-61)
- **tm_min** int perc (0-59)
- **tm_hour** int óra (0-23)
- **tm_mday** int nap (1-31)
- **tm_mon** int január óta eltelt hónap (0-11)
- **tm_year** int 1900 óta eltelt év
- **tm_wday** int vasárnap óta eltelt nap (0-6)
- **tm_yday** int Január 1 óta eltelt napok száma (0-365)
- **tm_isdst** int nyári időszámítás flag (1: nyári, 0: téli, -1 ismeretlen DST állapot)





Jelzések (signal)

`<signal.h>`

pause()

felfüggeszti a folyamatot, míg egy szignál nem érkezik hozzá. Csak az utána érkező szignálokra érzékeny.

kill(PID, signal)

Szignált küld a PID által meghatározott megadott folyamatnak.





Handler

Alap handler függvény:

```
void handler(int signum)
```

```
{
```

```
...
```

```
}
```





Handler

<signal.h>

signal(signal, handler)

A szignálhoz rendeli a kezelő eljárást.

handler:

- a kezelő eljárás belépési pontja, vagy
- SIG_IGN: figyelmen kívül hagyja, vagy
- SIG_DFL: alapértelmezett tevékenységet kapcsolja be.

psignal(signal, saját hibaüzenet)

Kiírja {saját hibaüzenet} szövegét, majd kettőspont és szóköz után a szignál leírását.





Jelzések blokkolása (maszkolás)

<signal.h>

- **sigset_t**: szignálkészlet típus.
- **sigemptyset(&sigset)**
inicializálja és üressé teszi a szignálkészletet.
- **sigfullset(&sigset)**
inicializálja és hozzáadja az összes szignált a szignálkészlethez.
- **sigaddset(&sigset, {szignál})**
hozzáadja {szignál}-t a szignálkészlethez.
- **sigdelset(&sigset), {szignál})**
eltávolítja {szignál}-t a szignálkészletből.



Jelzések blokkolása (maszkolás)

- **sigprocmask({művelet}, &sigset, &oldset)** szignálok blokkolásának beállítása.
{művelet}:
 - SIG_BLOCK: Hozzáadja a sigset-ben található szignálokat a blokkolt szignálokokhoz
 - SIG_UNBLOCK: Eltávolítja a sigset-ben található szignálokat a blokkolt szignálokok közül
 - SIG_SETMASK: A sigset-ben található szignálok lesznek a blokkolt szignálok





Mi generálta a szignált?

<signal.h>:

- **struct sigaction:**
eseményleíró struktúra.
 - sa_handler: kezelő eljárás belépési pontjának mutatója, vagy SIG_DFL, SIG_IGN valamelyike.
 - sa_mask: sigset_t típusú, a kezelő eljárás végrehajtása közben *további* blokkolandó szignálok készlete.
 - sa_flags: jelzések, melyek a szignál viselkedését határozzák meg
 - 0: alap viselkedés
 - SA_SIGINFO





Mi generálta a szignált?

SA_SIGINFO:

handler paraméterezésének szabályozása

- ha nincs beállítva:

void handler(int signo)

- ha be van állítva:

void handler(int signo, siginfo_t *info, void *context)





Mi generálta a szignált?

- **sigaction({signum}, &{act}, &{oldact})** megváltoztatja a {signum} szignál által kiváltott folyamatot.



Várakozás szignálra

- **sigsuspend(&{mask})**
a hívó folyamat szignál maszkját átmenetileg lecseréli a paraméterként átadott {mask}-ban megadottra, majd felfüggeszti a folyamat végrehajtását amíg egy olyan nem maszkolt szignál érkezik, mely meghív egy handlert vagy terminálja a folyamatot.
 - ha a szignál terminálja a folyamatot, a `sigsuspend` nem tér vissza (értelemszerűen, hiszen a folyamat terminált).
 - ha a szignál kezelése megtörtént (handler végzett), akkor a folyamat szignál maszkja visszakapja `sigsuspend` előtti értékét, és a folyamat végrehajtása folytatódik.





Valós idejű (real-time) szignálok

Ha egy blokkolt szignál többször is küldésre kerül a blokkolás alatt, akkor a blokkolás feloldásakor:

- ha nem real-time szignál, akkor csak egyszer érkezik meg,
- ha real-time szignál, akkor sorban áll, és minden megérkezik.

A real-time szignálok SIGRTMIN és SIGRTMAX közé eső szignálok.





rt library használata

A real-time funkciókat tartalmazó programok fordításakor meg kell adni, hogy linkeljen az rt library-vel. Ezt a fordítónak a -lrt kapcsoló megadásával jelezhetjük.

Pl. gcc valami.c -lrt





raise

<signal.h>

- **raise({signal})**

A raise függvényel a folyamat önmagának küld szignált.



Időzítés

Időzítő (interval timer): szignált küld amikor az időzítő lejár, de a folyamat végrehajtását nem függeszti fel.

Egyszeri időzítés:

- **alarm({másodperc})**

A megadott idő leteltekor küld egy SIGALARM szignált.



Időzítés

Ismétlődő időzítés:

<sys/time.h>

- struct **timeval**
 - tv_sec: másodperc,
 - tv_usec: milliomod másodperc.
- struct **itimerval**
 - it_interval: ismételt időzítés értéke,
 - it_value: lejáratig hátralévő idő (=első időzítés értéke).





Időzítés



- **setitimer**({időzítő},&{új timer},&{régi timer})
Beállítja és elindítja az ismétlődő időzítőt.
{időzítő}:
 - ITIMER_REAL: a ténylegesen eltelt időt méri, és lejáratkor SIGALARM szignált küld,
 - ITIMER_VIRTUAL: az időzítőt indító folyamat végrehajtásának idejét méri, és lejáratkor SIGVTALARM szignált küld.
 - ...
- **getitimer**({időzítő},&{timer})
Lekérdezi az {időzítő} által meghatározott időzítőt.



unió típus

A union egy speciális típus, mely lehetővé teszi különböző típusú adatok tárolását a memória ugyanazon helyén.

Deklaráció:

```
union azonosító {  
    típus1 adattag1;  
    típus2 adattag2;  
    ...  
}
```

Az unió típusú változó memóriafoglalása akkora, amekkora a legnagyobb méretű adattag memóriaigénye.





Szignállal küldött információ fogadása

```
void handler(int signo, siginfo_t *info, void *context)  
struct siginfo_t *info:
```

- info->si_code: miért lett küldve a szignál?
 - SI_USER: kill küldte,
 - SI_QUEUE: sigqueue küldte (ld. később),
 - SI_TIMER: timer küldte,
 - SI_MESGQ: üzenetsor küldte (ld. később),
 - ...
- info->si_pid: a küldő PID-je.
- info->si_value: a szignállal küldött információ union sival_t típusú (int vagy pointer):
 - info->si_value.sival_int: egy int típusú érték,
 - info->si_value.sival_ptr: egy mutató.





Információ küldése szignállal

<signal.h>

- **sigqueue({pid}, {signal}, {value})**
{signal} szignált küld {pid} folyamatnak,
és a szignállal küldi {value} union sigval
típusú értéket is.





Tetszőleges szignál küldése időzítővel

<time.h>

- **timer_create({clock id}, &{signal event}, &{timer id})**
Létrehoz egy új időzítőt.

{clock id}:

- CLOCK_REALTIME: beállítható valós idejű óra
- ...

{signal event} sigevent típusú struktúra:

- sigev_notify: értesítési mód (int):
 - SIGEV_NONE: nem történik semmi az esemény bekövetkezéskor
 - SIGEV_SIGNAL: szignált küld az esemény bekövetkezéskor
 - ...
- sigev_signo: szignál, amit küld (int),
- sigev_value: a szignállal küldött union sigval tip. információ
- ...

{timer id}: a létrehozott timer azonosítója (timer_t)





Tetszőleges szignál küldése időzítővel



- **timer_gettime({timer id}, {flags}, &{új timer}, &{régi timer})**
Beállítja és elindítja a {timer id} azonosítójú időzítőt.
- **timer_gettime({timer id}, &{aktuális érték})**
Lekérdezi {timer id} azonosítójú időzítőt.
- **timer_delete({timer id})**
Törli a {timer id} azonosítójú időzítőt.





Névtelen cső

- **pipe(int pipefd[2])**

Létrehoz egy csövet és a fájlazonosítóit elhelyezi a paraméterként átadott tömbben.

A tömb

- [0] eleme az olvasásra, az
- [1] eleme pedig az írásra használható fájlazonosítót kapja.



Névtelen cső

- **read(...)**

Vár, míg megjön a várt adatmennyiség vagy hiba történik (pl. eof).

- **write(...)**

Vár, míg ki tudja írni a teljes adatmennyiséget (pl. ha tele a fifo), vagy hiba történik.

- **close(...)**

A cső végeit (R/W) külön-külön kell lezárni.





Névtelen cső hátrányai

- Mivel a csövet olyan közös kódban kell megnyitni, mely minden a csövet használó folyamat közös kódja, ezért esetlegesen olyan folyamatokban is nyitva lesz, melyekben nem kívánjuk használni.
- Csak olyan folyamatok tudnak rajta keresztül kommunikálni, melyek azonos kódból indultak.





Nevesített cső (named pipe)

- **mkfifo({fájlnév}, {engedélyek})**
Létrehoz egy cső fájlt a megadott
hozzáférési engedélyekkel.
{engedélyek}: S_I[R|W][USR|GRP|OTH])
- **unlink({fájlnév})**
Törli a cső fájlt.





Nevesített cső (named pipe)

A cső csak úgy nyitható meg, ha minden végét megnyitjuk. Az open vár, míg a másik véget is megnyitja egy folyamat.

- Mialatt meg van nyitva írásra, akárhány szálban nyitható olvasásra.
- Mialatt meg van nyitva olvasásra, akárhány szálban nyitható írásra.
- Ha nincs megnyitva írásra de van benne adat, az olvasásra nyitás vár, míg írásra nyitás is érkezik, majd elsőként a már bennlévő adatot olvassa ki (nem vesz el a bennmaradt adat).
- Külön programokban elhelyezett folyamatok is kommunikálhatnak rajta keresztül.

eof akkor következik be, ha minden írásra nyitó folyamat lezárta és kiürült.



Cső állapotának lekérdezése



<poll.h>

- **pollfd** struktúra
 - fd: file azonosító (int)
 - events: figyelt esemény (short)
 - revents: bekövetkezett esemény (short)
- events és revents:
 - POLLIN: van olvasható adat a fájlban,
 - POLLOUT: lehet írni adatot a fájlba,
 - ...



poll

- int **poll**({figyelendők tömbje}, {figyelendők száma},
 {időtúllépés})

Blokkolja a folyamatot, míg

- egyik figyelt fájlleíróban a figyelt esemény bekövetkezik,
- szignál érkezik,
- időtúllépés történik.

Értéke:

- pozitív: Figyelt esemény következett be az egyik figyelt fájlban. Ekkor a {visszatérési érték} a figyelt fájl sorszáma (1-től számozva), és a bekövetkezett esemény a {figyelendők tömbje}[{visszatérési érték} -1].revents adattagba kerül.
- 0: letelt az {időtúllépés} által meghatározott idő.
- negatív: hiba



ppoll

- int **ppoll**({figyelendők tömbje},
 {figyelendők száma}, {időtúllépés},
 {szignálmaszk})

Blokkolja a folyamatot, míg

- egyik figyelt fájlleíróban a figyelt esemény bekövetkezik,
- szignál érkezik,
- időtúllépés történik.



select

- **fd_set**: fájlleíró készlet típus
- **FD_ZERO({készlet})**:
{készlet} kiürítése
- **FD_SET({fájlazonosító}, {készlet})**:
hozzáadja a {fájlazonosító}-t a {készlet}-hez
- **FD_CLEAR({fájlazonosító}, {készlet})**:
eltávolítja a {fájlazonosító}-t a {készlet}-ből
- **FD_ISSET({fájlazonosító}, {készlet})**:
értéke 0, ha nem tartalmazza a {készlet} a {fájlazonosító}-t



select

- **select({ndfs}, {readfds}, {writefds}, {exceptfds}, {időtúllépés})**

Figyeli a fájlleírókkal megadott fájlokat, és felfüggeszti a folyamat működését, míg

- valamelyikük készen áll a figyelt műveletre,
- vagy eltelik az {időtúllépés}-ben megadott idő,
- vagy szignál érkezése szakítja meg a folyamatot.

Értéke:

>0: Az egyes fájlleíró készletekbe azon fájlazonosítók kerülnek, melyek a figyelt funkcióra készenállnak. A visszatérési érték ezen azonosítók együttes száma.

0: Időtúllépés.

-1: Hiba.



IPC

System V InterProcess Communication:

- message queues (üzenetsorok),
- semaphore sets (szemafor készletek),
- shared memory segments (osztott memória szegmensek).

IPC üzenetsor parancssori műveletek:

- **ipcs**: kilistázza az IPC-ket.
- **ipcrm -q** {üzenetsor azonosító}: eltávolítja az üzenetsort (pl. beragadt).
- **ipcmk -Q**: új üzenetsort hoz létre.





System V üzenetsorok

<sys/ipc.h>

- **ftok({path}, {project id})**
IPC kulcsot generál.
- **msgget({kulcs}, {jogosultságok})**
Üzenetsort hoz létre.
- **key_t típus:**
 - IPC_PRIVATE: automatikusan generál kulcsot,
 - ftok-kal generált kulcs.
- **msgctl({üzenetsor azonosító}, IPC_RMID,
NULL)**
Törli az üzenetsort.





System V üzenetsorok

- **msgsnd**({üzenetsor azonosító},
&{üzenet struktúra}, {üzenet mérete},
{flag})

Elküldi a megadott azonosítójú üzenetsorba az üzenetet.

Ha nincs elég hely a sorban az üzenet számára:

- {flag}=0: blokkolja a folyamatot, míg lesz hely,
- {flag}=IPC_NOWAIT: nem blokkolja a folyamatot, hanem hibát ad vissza.





System V üzenetsorok

- **msgrcv({üzenetsor azonosító}, &{üzenet struktúra}, {puffer mérete}, {üzenet kategória}, {flag})**

Kiolvas egy üzenetet a megadott azonosítójú üzenetsorból.

Ha

- {üzenet kategória}=0: a soron következő üzenetet olvassa ki,
- {üzenet kategória}<>0: a soron következő {üzenet kategória}-ban megadott kategóriájú üzenetet olvassa ki.

Ha nincs a sorban a kategóriának megfelelő üzenet:

- {flag}=0: blokkolja a folyamatot, míg megérkezik a várt üzenet,
- {flag}=IPC_NOWAIT: nem blokkolja a folyamatot, hanem hibát ad vissza.



Prioritásos POSIX üzenetsorok

<mqueue.h>

- **mq_open({üzenetsor név}, {mód}, {engedélyek}, &{attribútumok})**
Létrehoz egy új POSIX üzenetsort {üzenetsor név} néven.
- **mq_open({üzenetsor név}, {mód})**
Csatlakozik egy már létező, {üzenetsor név} nevű POSIX üzenetsorhoz.

Értéke **mqd_t** típusú

- üzenetsor leíró vagy,
- -1 ha hiba történik.





Prioritásos POSIX üzenetsorok



- **mq_open({üzenetsor név}, {mód})** vagy
 - **mq_open({üzenetsor név}, {mód},
{engedélyek}, &{attribútumok})**
- {mód}:
- O_RDONLY,
 - O_WRONLY,
 - O_RDWR,
 - O_CREAT,
 - O_EXCL,
 - O_NONBLOCK.





/proc/sys/fs/mqueue/

- msg_default: (10) üzenetek maximális számának alapértelmezett értéke,
- msg_max: (10) üzenetek maximális száma az adott környezetben,
- msgsize_default: (8192) üzenetek méretének alapértelmezett értéke,
- msgsize_max: (8192) üzenetek maximális mérete az adott környezetben,
- queues_max: (256) független üzenetsorok maximális száma az adott környezetben.





Prioritásos POSIX üzenetsorok

- **mq_send**({üzenetsor leíró}, &{üzenet},
 {üzenet mérete}, &{prioritás})
 Hozzáadja az üzenetet az üzenetsorhoz.
- **mq_receive**({üzenetsor leíró},
 &{üzenetpuffer}, {maximális üzenetméret},
 &{prioritás})
 Kiolvassa a soron következő üzenetet az
 üzenetsorból.





Prioritásos POSIX üzenetsorok

- **mq_close({üzenetsor leíró})**
Lezárja az üzenetsorhoz való kapcsolódást.
- **mq_unlink({üzenetsor leíró})**
Törli az üzenetsort.
- **mq_notify({üzenetsor leíró}),
&{signal event})**
Szignált küld a feliratkozó folyamatnak,
amikor egy új üzenet érkezik az *üres*
üzenetsorba.



Osztott memóriaterület létrehozása

<sys/shm.h>

- **shmget({kulcs}, {méret}, {flagek})**

Létrehoz egy legalább {méret} byte méretű osztott memóriaterületet, vagy csatlakozik az azonos kulcsú osztott memóriaterülethez.

{kulcs}

key_t típusú, ftok-kal generált vagy IPC_PRIVATE

{flagek}

- fájlhozzáférési engedélyek (S_IRUSR, S_IWUSR, ...),
- IPC_CREAT,
- IPC_EXCL.

értéke: nem negatív osztottmemória-azonosító (shmid), vagy -1 ha hiba.



Osztott memóriaterület kezelése

- **shmat({shmid}, NULL, 0)**
Kapcsolódik az {shmid} azonosítójú osztott memóriához.
értéke az osztott memória kezdőcíme (shmaddr), vagy -1, ha hiba.
- **shmdt({shmaddr})**
Leválasztja az {shmaddr} memóriacímen kezdődő osztott memóriaterületet.
- **shmctl({shmid}, IPC_RMID, NULL)**
Törli az {shmid} azonosítójú osztott memóriát

