

# 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
  - -1 (hiba).
- a gyerek folyamatban: 0



# Folyamatok kezelése

`pid_t getpid()`

a folyamat azonosítójának lekérdezése

`pid_t getppid()`

a folyamat szülőjének azonosítóját kérdezi le



# Feladat

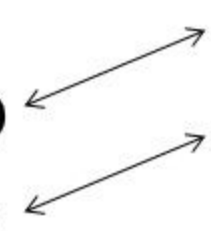


Hozzunk létre azonos szülőfolyamatból két gyermekfolyamatot. A szülő írja ki a két gyerek PID-jét, a gyerekek pedig a saját és a szülőjük PID-jét.

Tapasztalat: Összekeverednek a sorok!  
Szinkronizálni kellene!



# Szülő-gyerek PID-ek

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

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



# Problémák

## 1. A folyamatok szinkronizálása

- Tudjon róla az egyik folyamat, ha a másik terminált.
- Tudja értesíteni az egyik folyamat a másikat arról, hogy végzett valami részfeladattal.

## 2. A folyamatok közötti adatcsere



# Folyamatok szinkronizálása

**sleep**({másodperc})

**usleep**({mikroszekundum})

Felfüggeszti a folyamat végrehajtását a megadott ideig, vagy egy szignál érkezéséig.





# Folyamatok szinkronizálása

pid\_t **wait**(&status)

Felfüggeszti a folyamat végrehajtását, míg bármelyik gyerekfolyamat terminál.

*Értéke:* a megváltozott folyamat PID-je, vagy -1, ha nincs gyerekfolyamat.



# Folyamatok szinkronizálása

`pid_t waitpid(PID, &status, {options})`

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

*Értéke:*

- A megváltozott folyamat PID-je, vagy
- -1, ha
  - nincs több gyerekfolyamat vagy
  - a megadott {PID} nem gyerekfolyamaté.

`<sys/wait.h>`

`{options}`:

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





# Folyamatok szinkronizálása

`<signal.h>`

**pause()**

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



<signal.h>

**kill**({PID}, {signal})

{signal} szignált küld a PID által meghatározott folyamatoknak.

{PID}:

- a folyamat azonosítója, amelyiknek a szignált küldi, vagy
- 0: a küldő folyamattal azonos csoportban lévő összes folyamatnak, vagy
- -1: az összes folyamatnak, melyekre a küldő folyamat rendelkezik szignál küldési engedéllyel

*Értéke:*

- 0: ha sikerült legalább egy szignált küldeni,
- -1: ha nem sikerült



# Külső program végrehajtása

**execv**({parancs}, {argumentumok})

Lecseréli az aktuális folyamatot a meghívott folyamatra.

{parancs}: karaktertömb,

{argumentumok}: karaktertömbök tömbje



# Külső program végrehajtása

**system({parancssor})**

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

{parancssor}:

karaktértömb, tartalma

parancsértelmezőnek kiadandó utasítás



# Véletlenszám generálás

`<stdlib.h>`

**rand()**

Értéke egy `[0,RAND_MAX]` intervallumba eső véletlenszám.

**srand({kezdőérték})**

Véletlenszámgenerátor kezdőértékének megadása.



`<time.h>`

`time_t time(NULL)`

*Értéke:* az aktuális rendszeridő.

`localtime(&{time_t})`

Átkonvertálja `time_t` típusú időt `tm` típusúra.

*Értéke:* egy `tm` típusú struktúra mutatója.





## 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)

