

UNIX: folyamatok kezelése

kiegészítő fóliák az előadásokhoz

Mészáros Tamás

<http://home.mit.bme.hu/~meszaros/>

*Budapesti Műszaki és Gazdaságtudományi Egyetem
Méréstechnika és Információs Rendszerek Tanszék*

Áttekintés

- Alapismeretek
 - Mi a folyamat? Hogy indul? Hol látszik? Viszonya a kernellel?
 - Kontextusok és végrehajtási módok
- Folyamatok
 - alapadatok
 - állapotok és állapotátmenetek
 - életciklus: létrehozás, futás (ütemezés), leállítás
- Rendszerhívások
 - új folyamat létrehozása
 - új programkód betöltése
- Klasszikus és modern UNIX ütemezők
 - prioritás, ütemezési szintek és működés, preemptivitás
 - modularitás, real-time, interaktív és multimédia folyamatok igényei
 - többprocesszoros rendszerek sajátosságai

UNIX folyamatok – felhasználói ismeretek

- Mi az a folyamat?
 - Feladataink megoldására programokat írunk
 - Egy program futás alatt álló példánya a folyamat (def.)

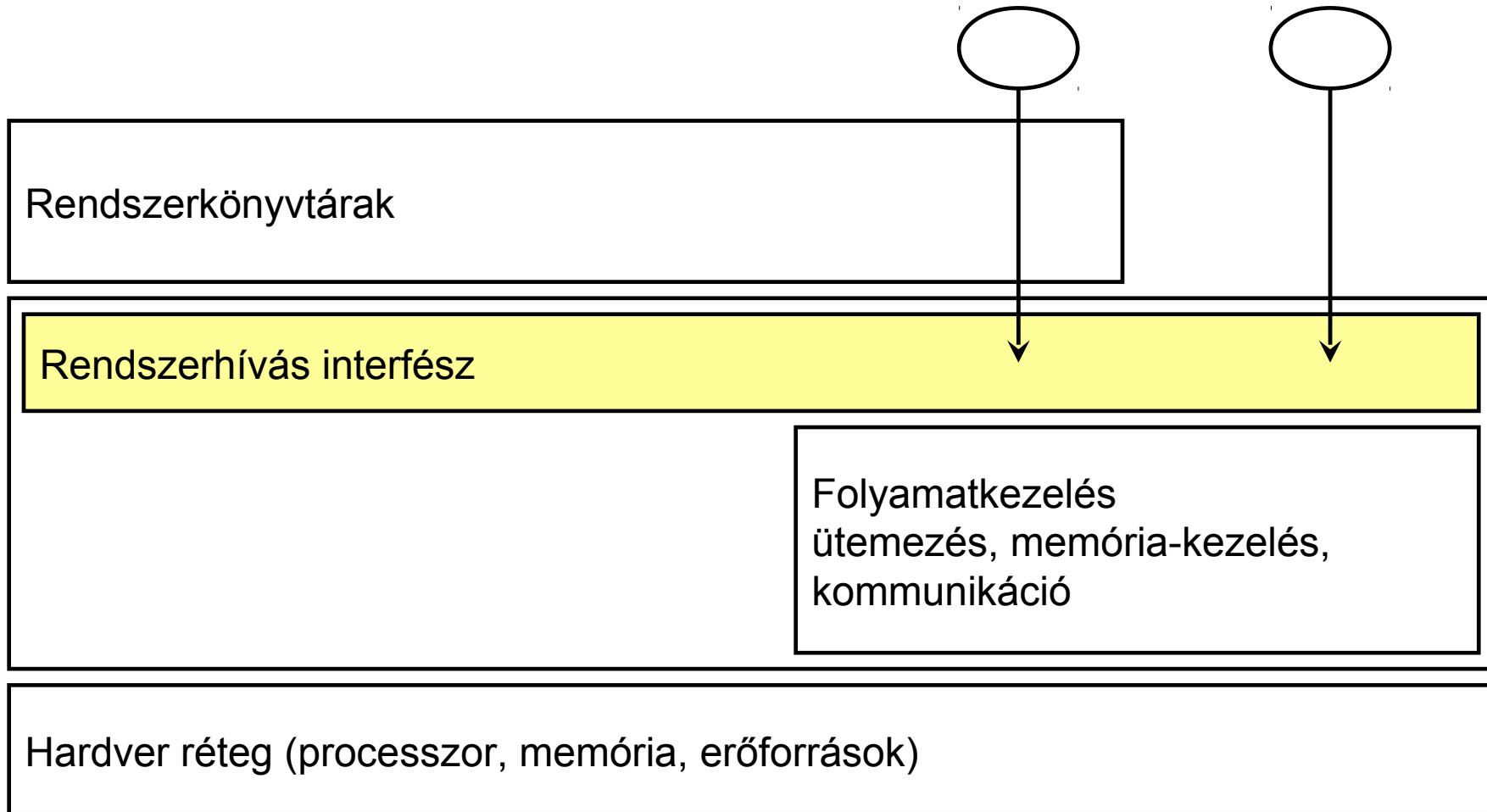
- Hogyan indulnak a folyamatok?
 - Rendszerinduláskor (rendszerfolyamatok, szolgáltatások)
 - Felhasználó által (elindítja, illetve igény szerinti szolgáltatást kér)

- Hogyan kezeljük a folyamatokat?
 - Listázás (folyamat neve, azonosítója, futtatója, erőforrás adatok, ...)
 - `ps`, `ps -ef`, `ps axu`, `ps -u <felhasználó>`, ...
 - `top`, `atop` és grafikus társaik
 - Vezérlés (leállítás)
 - (felhasználói felületükön)
 - jelzésekkel: `CTRL+C`, `CTRL+Z`, `kill <JELZÉS> <FOLYAMAT AZONOSÍTÓ>`
 - egyebek: pl. prioritás állítás: `renice`

UNIX folyamatok – kernel alapismeretek

- Folyamatok elkülönítése a kerneltől
 - Végrehajtási mód: különbség a kernel és a folyamat programkódja között
 - Kontextus: kernel és folyamat adatok közötti különbség
- Végrehajtási (futási) módok:
 - Kernel („privilegizált, védett”) mód
 - védett (kernel) tevékenységek végrehajtása
 - Felhasználói („szabad”) mód
 - a folyamat programkódjának végrehajtása
- Végrehajtási környezetek:
 - Kernel (rendszer vagy megszakítás) kontextus
 - a kernel saját feladatainak ellátásához szükséges adatok
 - Folyamat kontextus (virtuális memória-kezelés!)
 - a folyamat futásának adatai (programszöveg, adatok, verem, stb.)
 - a folyamatok kezeléséhez, vezérléséhez szükséges adatok

A folyamatok és a kernel (emlékeztető)



Folyamatok futtatása: végrehajtási mód és kontextus



A végrehajtási mód váltása

- A felhasználói mód és a kernel mód között átmenet lebonyolítása
- Jellemzően rendszerhívások meghívása esetén zajlik:
 - a folyamat védett módban végrehajtható tevékenységet szeretne végezni (pl. fájl írása, olvasása, eszközök kezelése, pontos idő lekérdezése, stb.)
 - meghívja a megfelelő rendszerhívást (pl. `open()`, `read()`, `write()`, stb.)
 - Ez a programozók számára klasszikus függvényhívásnak tűnik, de valójában egy speciális szoftver megszakítás, amit a függvényhívást implementáló libc rendszerkönyvtár bonyolít le.
 - a `libc` kiadja a `SYSCALL` utasítást (megszakítást generál)
 - A `SYSCALL` neve hardver függő, pl. `trap`, `syscall`, `sysenter`
 - a CPU kernel (védett) módba vált a megszakítás hatására
 - a kernel `SYSCALL` kezelője előkészíti a rendszerhívás végrehajtását
 - végrehajtódik a kernel módú eljárás (a rendszerhívás utasításai)
 - a kernel visszatér a megszakításból (`iret`, `sysexit`)
 - a CPU felhasználói módba vált a visszatérési utasítás hatására
 - a `libc` visszatér a folyamat által meghívott (rendszerhívás) függvényből
- Hardver megszakítások és kivételek (hibák) esetén is kell módváltás

Rendszerhívások nyomkövetése (gyakorlat)

- Nézzük meg egy folyamat rendszerhívásait!
 - nyomkövetés (Linux alatt): `strace`
 - bővebb információ és példák: `man strace`
 - A Solaris DTrace megoldásáról később részletesen szó lesz.

- Milyen rendszerhívásokat hajt végre a `ps` parancs?

```
strace -c ps
```

```
strace -e open ps
```

- Milyen rendszerhívásokkal dolgozik a firefox böngésző?

RHEL 5, Firefox 3.0.12

```
ps -ef | grep firefox
```

```
strace -c -p <Firefox_PID>
```


Virtuális rendszerhívások

(Nem a klasszikus UNIX része.)

- Probléma: túl sok rendszerhívás, sok interrupt, sok kontextusváltás
 - A Firefox példa: a `gettimeofday()` csak a pontos időre kíváncsi
 - `gettimeofday()` – `libc` – `SYSCALL` – kontextusváltás – stb.
- Az egyszerű esetekben próbáljuk lerövidíteni az utat
 - Bizonyos rendszerhívások esetében próbáljuk elkerülni a módváltást.
 - Ha nincs módváltás, akkor felhasználói címtérben el kell érni kernel területeket.
 - Lényegében bizonyos kernel funkciókat felhasználói címtérben levő eljárásokként teszünk elérhetővé.
- Virtuális rendszerhívások (Linux)
 - minden folyamat címtérében megjelenik egy speciális „kernel” lap
 - biztonságosnak ítélt rendszerhívások érhetők el rajta
 - nincs kontextusváltás, módváltás, nem érik el a kernel címtérét
 - a felhasználók programjai nem látják a különbséget (a `libc` igen)

UNIX folyamatok kontextusa részletesebben

- Folyamat adatok: programszöveg, adatok, veremek, stb.
- Hardver kontextus: cpu, mmu, fpu, stb.
- Adminisztratív adatok (a folyamatok kezeléséhez, vezérléséhez)
 - elsősorban a folyamat futása során szükségesek **u-terület**
 - hozzáférés-szabályozás adatai **A folyamat címtér része**
 - rendszerhívások állapotai és adatai
 - nyitott fájl objektumok
 - számlázási és statisztikai adatok, stb.
 - elsősorban a folyamatok kezeléséhez szükségesek **proc struktúra**
 - azonosítók **A kernel címtér része**
 - futási állapot és ütemezési adatok
 - memóriakezelési adatok, az u-terület címe, stb.
- Környezeti adatok (a folyamat indításakor megörökölt tulajdonságok)
 - megörökli az őt elindító folyamat környezetét
 - *tulajdonság = érték* párok halmaza
 - `set`, `setenv`, `export` parancsokkal a felhasználók is beállíthatják

A folyamatok főbb adminisztratív adatai

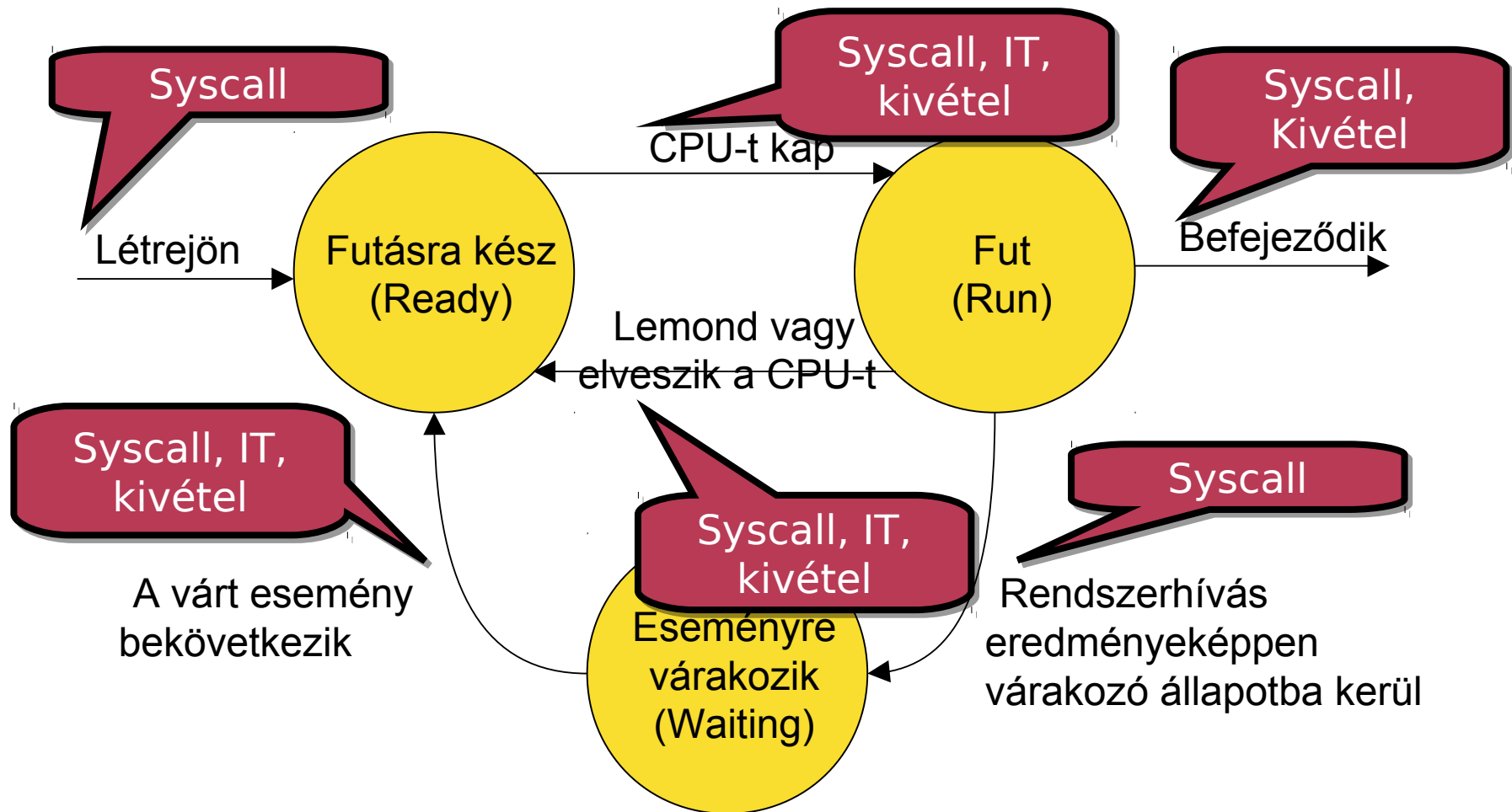
- PID (Process ID): egyedi, a folyamatot azonosító szám
 - PPID: szülő folyamat azonosítója
- A folyamat állapota
 - fut, alszik, stb. (részletesen később)
 - ütemezési információk (prioritás, stb., lásd később)
- Hitelesítők
 - UID: a futtató felhasználó azonosítója (UID=0 a *root* vagy *superuser*)
 - GID: a futtató felhasználó csoport azonosítója
 - valós (real) és effektív (effective) azonosítók (lásd fájlok *setuid* bit)
- Memória-kezelési adatok
 - címleképezési térkép
- Kommunikációs adatok
 - fájlleírók, jelzés információk
- Statisztikák
 - erőforrás használat (számlázáshoz)

Folyamatok életrajza (állapotátmenetek)

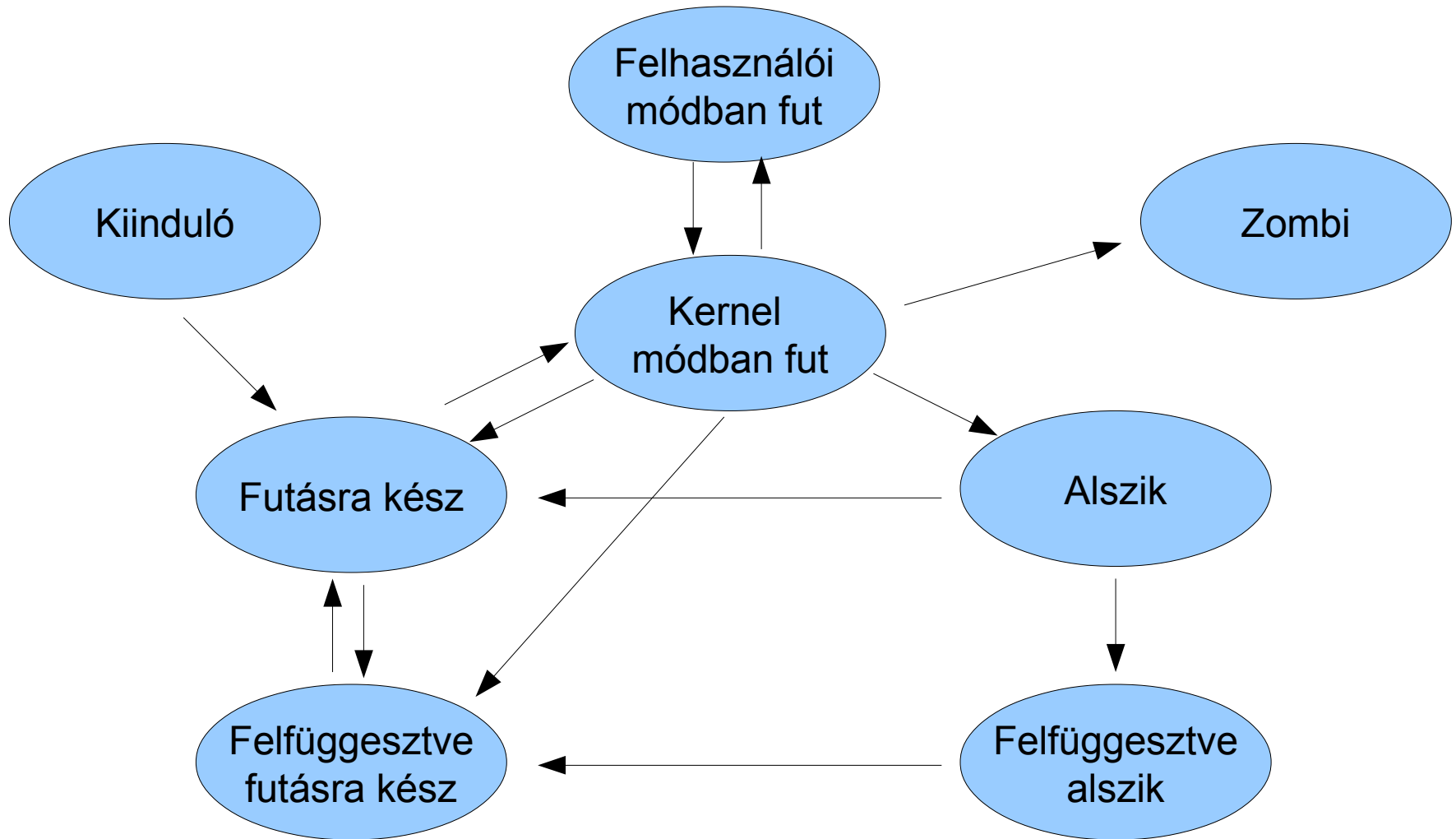
- Létrehozás
 - fork(): új folyamat létrehozása
 - exec(): új programkód betöltése egy folyamat címtérébe
- Futás és várakozás
 - a futás elindítása (a processzor folyamathoz rendelése)
 - a futás megállítása (a processzor elvétele)
 - várakozás eseményre (önként)
 - felfüggesztés (felhasználó)
 - leállítás (önként vagy a kernel)
 - átütemezés (másik, fontosabb folyamat érkezett)
- Leállítás
 - exit() rendszerhívással
 - zombi állapot
 - szülő értesítése
 - gyerekek adoptálása

Feladat állapota 9. (ism.)

- Minden állapotátmenet megszakításra történik!
- **A modern operációs rendszerek megszakítás vezéreltek!**



UNIX folyamatok állapotai



A `fork()` és az `exec()` rendszerhívások

- A `fork()` eltérő értékkel tér vissza a szülő és a gyerek esetében
- Az `exec()` sikeres végrehajtás esetén nem tér vissza

- Kódminta

```
if ((res = fork()) == 0) {
    // gyerek
    exec(...);
    // ha visszatér, exec hiba történt
} else if ( res < 0 ) {
    // fork hiba történt
}
// res = CHILD_PID (>0), szülő kódja fut tovább
```

- `fork()` variációk (címtér használat, szálak többszörözése)
 - `clone()` (osztott címtér), `vfork()` (`exec`), `fork1()` (egy szál)
- `exec()` variációk (elérés, argumentumok, környezet)
 - `execl()`, `execv()`, `execle()`, `execve()`, `execvp()`, ...

A folyamatok családfája

- Folyamatot csak egy másik folyamat tud létrehozni
 - minden folyamatnak van szülője és lehetnek gyerekei
 - a szülő változhat (így a gyerekek listája is)
- A `fork()` megadja a szülőnek a gyerek azonosítóját (PID)
- Az ős folyamat (PID 1: `init`, `upstart`, `systemd`, ...)
 - minden folyamat őse
 - a rendszer leállításáig fut
 - örökli az árva folyamatokat
 - figyel / vezényli bizonyos rendszerfolyamatok meglétét / futását
- A család fontos
 - a szülő értesítést kap a gyerek folyamat leállításáról (nyugtáznia kell)

Összefoglalás

- Folyamatokkal kapcsolatos alapismeretek
 - felhasználói kezelésük: `ps`, `kill`, `nice`
 - futási mód és kontextus
 - rendszerhívások működése
 - metaadataik (u-terület, processz leíró)
- Folyamatok életciklusa
 - létrehozás: klasszikusan a `fork()` rendszerhívással
 - új végrehajtandó kód betöltése: `exec()`
 - állapotok (speciális: kétféle futó állapot, felfüggesztett állapotok, zombi)
 - vezérlés alapvetően jelzésekkel
- Folyamatok családfája
 - a `fork()` fát épít, az ős folyamat az `init` (PID 1)
 - a szülők értesülnek a gyerekek leállításáról