UNIX: folyamatok kezelése

Mészáros Tamás
http://www.mit.bme.hu/~meszaros/

Budapesti Műszaki és Gazdaságtudományi Egyetem
Méréstechnika és Információs Rendszerek Tanszék

Problémafelvetés

- „Lassú a rendszer”
  - Mi történik a rendszerben?
  - Ki, mit csinál?
  - Miért reagálnak lassan az alkalmazások?
- „A böngészőm eszi a processzort, pedig nem csinálok semmit”
  - Miért lassú egy alkalmazás?
  - Mit csinál, miközben nem is „piszkáljuk”?
- „Szeretném, ha tovább bírná a laptopom egy feltöltéssel”
  - Mi fut, kell-e futnia?
  - Mi fogyasztja a legtöbb energiát? Szükséges?
- „core dumped”, „kernel panic”
  - Miért fagyott le az alkalmazás? Mit csinált?
  - Miért állt le a gépem kernel hibával? Ki futott? Mi történt?

Áttekintés (két előadás)

- 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

„Mi történik a rendszerben?”

- Listázás (folyamat neve, azonosítója, futtatója, erőforrás adatok, ...)
  - `ps, ps -ef, ps axu, ps -u <felhasználó>, ...`
  - `top, atop, htop` és grafikus társaik (System monitor, gkrellm, procexp, ...)

- Mit látunk a listákban (ps -ef)?
  - UID: a folyamatot futtató felhasználó
    - RUID: real UID, EUID: effective UID
  - PID: a folyamat egyedi numerikus azonosítója
  - PPID: a folyamat szülő folyamatának PID azonosítója (l. később)
  - STIME: mikor indult a folyamat
  - TTY: melyik terminálhoz (felhasználói belépéshez) kapcsolódik
  - TIME: mennyi processzoridőt használt a folyamat
  - CMD: a folyamat programja

- Mit látunk a folyamatmonitorozó programokban a fentieken kívül?
  - Általános rendszerjellemzőket (CPU%, MEM%, DSK%, NET%, stb.)
  - Aggregált statisztikai adatokat

A folyamatok főbb adminisztratív adatai

- PID (Process ID): egyedi, a folyamatot azonosító szám
  - PPID: szülő folyamat azonosítója

    (Házi feladat: TID, TGID?)
- A folyamat állapota
  - fut, alszik, stb. (részletesen később)
- ütemezési információk (prioritás, stb., lásd következő órán)
- 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)

„Hogyan menedzselhetjük a folyamatokat?”

- A folyamatok életciklusa (ismétlés)
  - Létrehozás, futásra kész, fut, várakozik, leáll
  - (UNIX specialitások: kernel fut, felh. fut, zombi – részletesen később)

- Hogyan és mikor jönnek létre a folyamatok?
  - Rendszerinduláskor (rendszerfolyamatok, szolgáltatások)
  - Felhasználó által (elindítja, illetve igény szerinti szolgáltatást kér)
  - Valamilyen (külső) esemény hatására (pl. hálózati kérés kiszolgálása)

- Hogyan vezérelhetjük a folyamatokat?
  - (felhasználói felületükön)
  - jelzésekkel: `CTRL+C, CTRL+Z, kill <JELZÉS> <FOLYAMAT AZONOSÍTÓ>`
  - egyebek: pl. prioritás állítás: `renice`

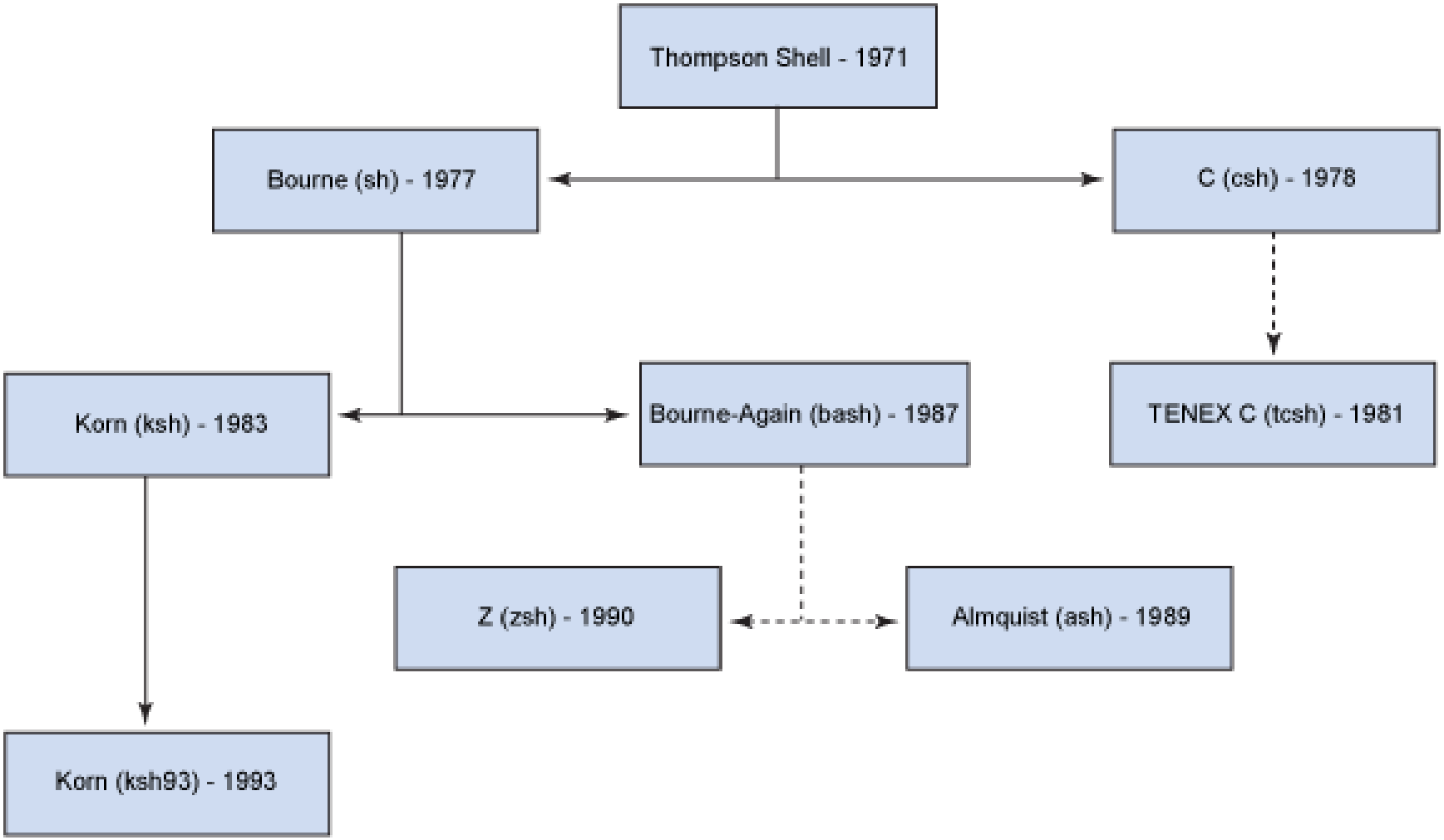
Speciális folyamatok a UNIX rendszerekben

- A kernel folyamatai (*emlékeztető: mikrokernel vs. monolitikus kernel*)
  - A ps parancsban jellemzően [ ] jelek között látszanak.
  - Nem mindegyik teljesen „igazi” folyamat, és nem mindegyik létezik mindig
  - kjournald, kswapd, …
  - Init (PID=1): minden folyamat őse, futási szintek menedzsere (l. később)

- Az alapvető rendszerszolgáltatások folyamatai (*daemon processes)*
  - Példák: számítógép nevének beállítása, időkezelés, fájlrendszerek ellenőrzése és csatolása, hálózati interfészek konfigurálása, tűzfal, ...
  - Jellemzően az `init` indítja őket (bár újabban upstart, systemd) a `/etc/init.d/` alatt található parancsfájlok futtatásával.
  - Az indítási sorrendet a `/etc/rc?.d/` fájlsorrendje határozzák meg.
  - Az adott futási szinten aktív szolgáltatások beállítása: **`ntsysv, bum`**

- Felhasználói parancsértelmező (shell)
  - Lehetővé teszi a felhasználói és az operációs rendszer interakcióját
  - Sok beépített paranccsal rendelkezik, de más programokat is elindíthat.

Shell (parancsértelmező, parancshéj)



Forrás: http://www.ibm.com/developerworks/

A rendszer futási szintje (runlevel)

- A rendszerindulás után az **`init`** az első folyamat.
  - A kernel indítja el (nincs szülő folyamata, lásd később)
  - Meghatározza és elindítja a további folyamatokat (azért nem mindegyiket)
  - Ezt a rendszer futási szintjének megfelelően teszi meg.

- A rendszer futási szintje
  - Egy számmal (időnként egybetűs rövidítéssel) azonosított
  - Meghatározza a rendszer működési módját, az aktív szolgáltatások körét
  - A rendszergazda kiválaszthatja és beállíthatja.
  - A **`/etc/inittab`** sorolja fel a futási szinteket
  - UNIX változatonként eltérő mennyiségű és értelmezésű, de jellemzően...
    - 0: teljes leállás
    - 1 vagy S: single-user: egyfelhasználós (adminisztrátori) mód
    - 2-5 környékén: többfelhasználós üzemmódok, GUI-val illetve nélküle
    - Jellemzően az 5. az alapértelmezett, amelyik a teljes GUI-t jelenti
    - 6: újraindítás
  - Váltás: **`telinit, shutdown`** (`halt, reboot`)

Folyamatok életciklusa (állapotátmenetek)

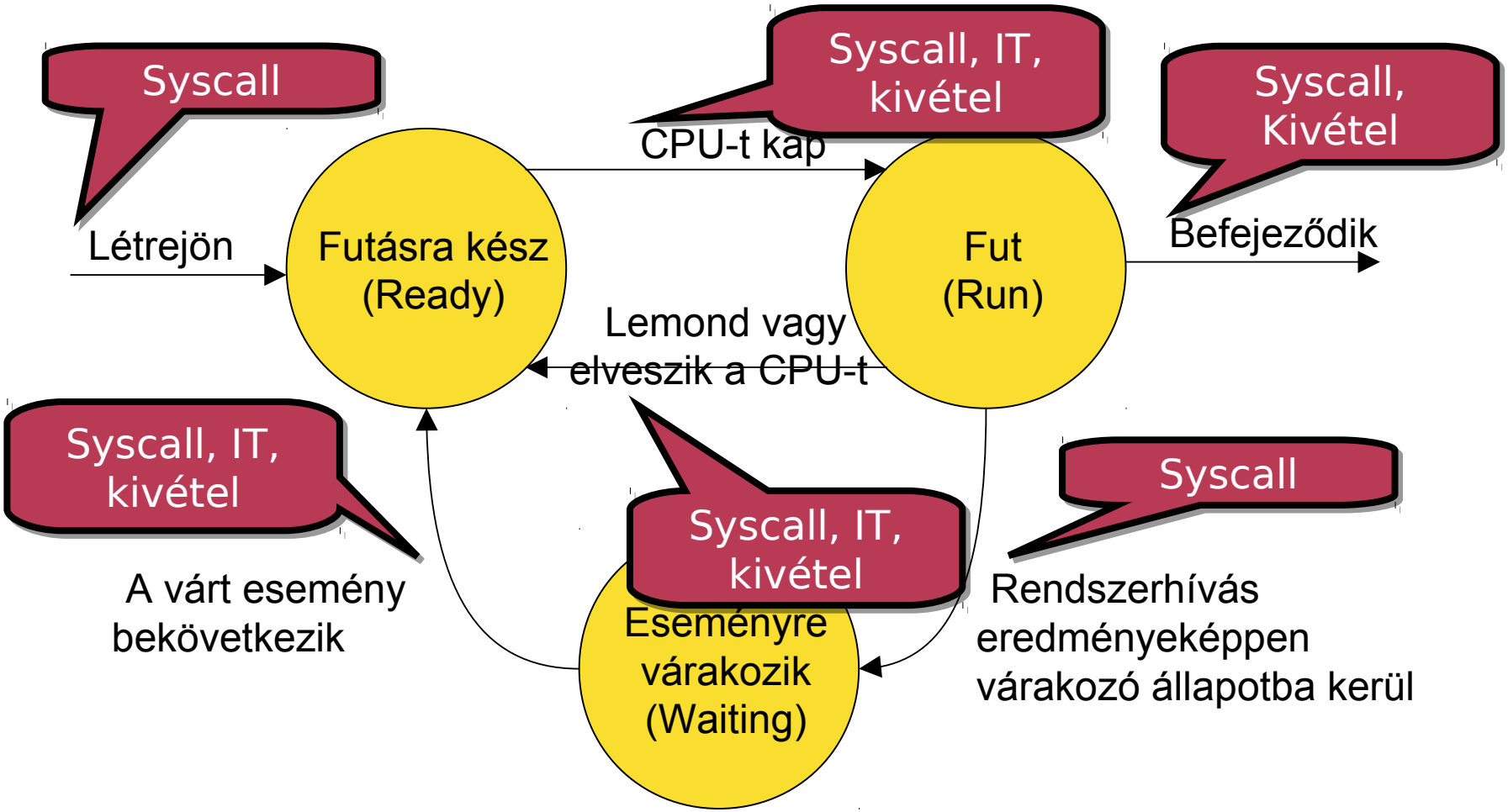
- Létrehozás
  - Új folyamat létrehozása mindig meglevő folyamat duplázásával (fork)
    - Ebből következik, hogy a folyamatok családfába rendezhetők.
  - Már futó folyamat címterébe új programkódot tölthetünk be (exec)

- Futás és várakozás
  - a futás elindítása (a processzor folyamathoz rendelése)
  - a futás megállítása (a processzor elvétele)
    - várakozás eseményre (önként)
    - felfüggesztés (felhasználó)
    - leállás (önként vagy a kernel)
    - átütemezés (másik, fontosabb folyamat érkezett)

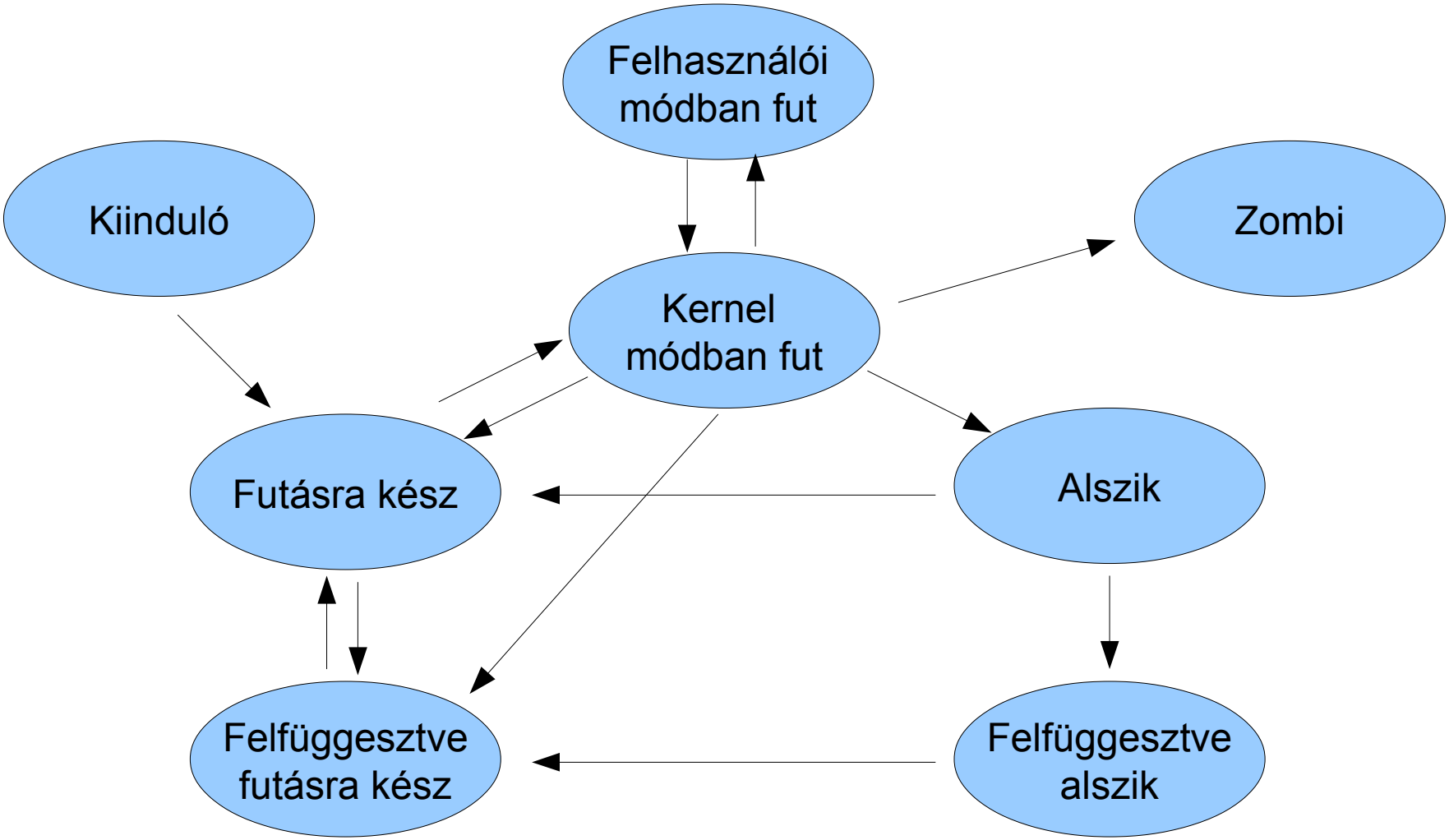
- Leállás (exit)
  - Először zombi állapotba lép a folyamat.
  - A szülő értesül a gyerekfolyamat leállásáról
  - A leálló folyamat gyerekeit az init adoptálja.

Feladat állapota 9. (ism.)

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



UNIX folyamatok állapotai



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 leállásáig fut
  - örökli az árva folyamatokat
  - figyeli / vezényli bizonyos rendszerfolyamatok meglétét / futását
- A család fontos
  - a szülő értesítést kap a gyerek folyamat leállásáról (nyugtáznia kell)

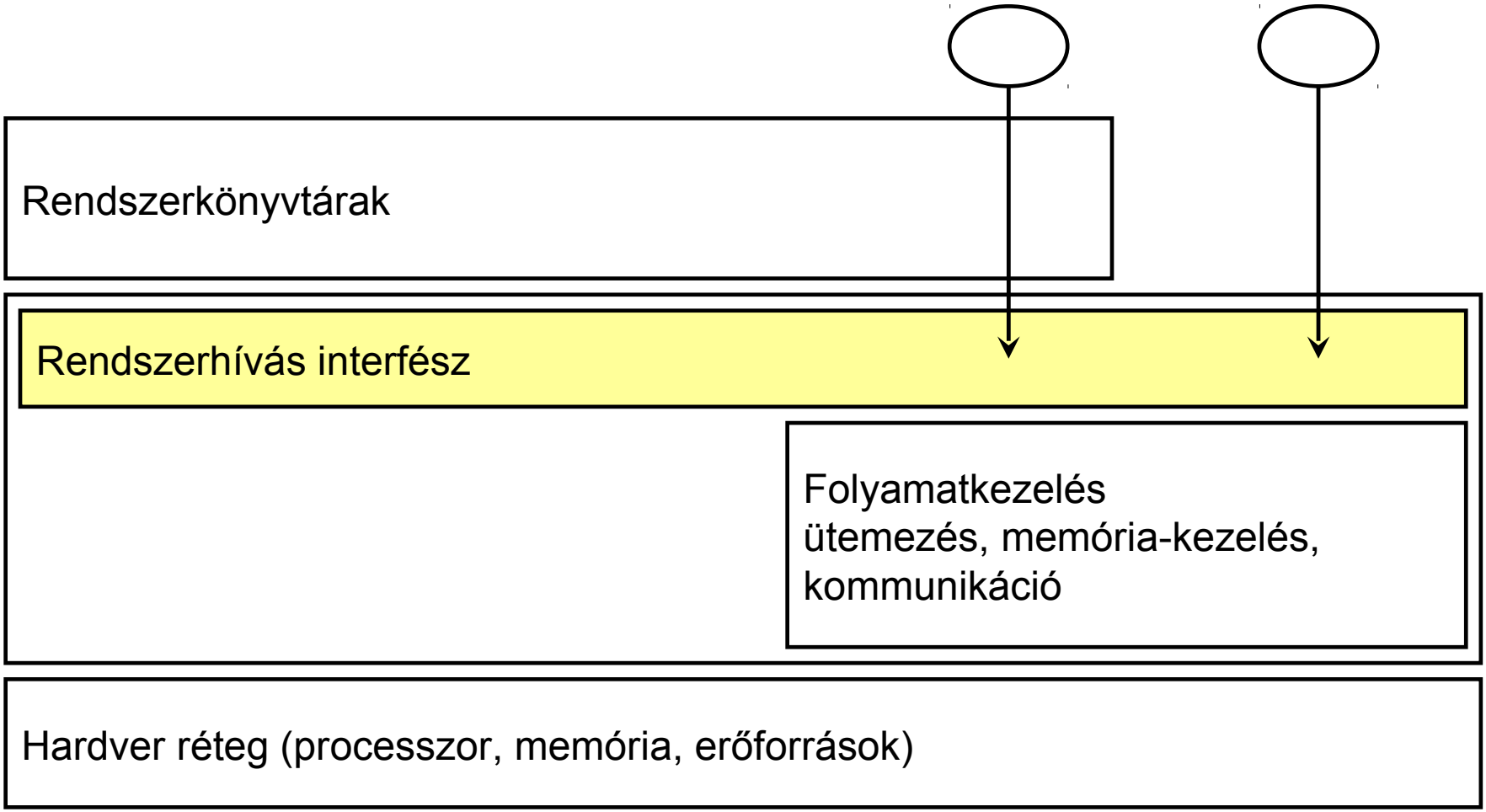
UNIX folyamatok – kernel alapismeretek

- 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

	felhasználói mód	kernel mód
folyamat kontextus	alkalmazás	rendszerhívás
kernel kontextus	(üres)	megszakítások, rendszer feladatok

UNIX folyamatok kontextusa részletesebben

- Folyamat adatok: programszöveg, adatok, vermek, 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* = *értek* párok halmaza
  - `set, setenv, export` parancsokkal a felhasználók is beállíthatják

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

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()`, ...

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 -cp <Firefox_PID>
```

A /proc fájlrendszer

- A folyamatok (és sok más) kernel adat elérhető fájlokon keresztül
  - `/proc`
  - `man proc`
  - Ahogy a korábbi demókból láttuk minden folyamat kap egy könyvtárat
  - A folyamatokat listázó programok jellemzően innen olvasnak
  - Több információt érhetünk el itt, mint a folyamatkezelő alkalmazásokban

- Folyamatok adatai a /proc fájlrendszerben
  - a végrehajtott program és paraméterei (cmd, cmdline)
  - munkakönyvtár (cwd) és környezet (environ)
  - fájljelírók (fd, fdinfo, részletesen későbbi előadáson)
  - memóriakezelési információk (maps, statm)
  - a folyamat futási állapota (stat, nehezen értelmezhető, jobb a ps)
  - aktuális rendszerhívás (wchan)
  - az adott UNIX változattól erősen függ a pontos tartalom és értelmezés
    Linux: http://www.lindevdoc.org/wiki//proc/pid/status

Virtuális rendszerhívások

- Probléma: 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 címterében megjelenik egy speciális „kernel” lap
  - biztonságosnak ítélt rendszerhívások érhetők el rajta
  - nincs kontextusváltás, módváltás, nem érik el a kernel címterét
  - a felhasználók programjai nem látják a különbséget (a libc igen)

Összefoglalás

- Folyamatokkal kapcsolatos alapismeretek
  - felhasználói kezelésük: `ps, kill, nice`
  - folyamatok adatai (u-terület, processz leíró)
  - a rendszer futási szintje, jellemző szolgáltatások folyamatai
  - a folyamat futási módja és kontextusa
  - a rendszerhívások (kontextusváltás) működése

- 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 leállásáról