(kernel3d vizualizáció: kernel245_graph.mpg)

http://www.pabr.org/kernel3d/kernel3d.html

http://blog.mit.bme.hu/meszaros/node/163

(ml4 unix mérés boot demo)

UNIX: folyamatok kezelése

kiegészítő fóliák az előadásokhoz

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

Budapesti Műszaki 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 tulajdonságai
  - 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
- Hogyan kezeljük a folyamatokat?
  - Listázás (folyamat neve, azonosítója, futtatója, erőforrás adatok, ...)
    - `ps -ef, ps axu, ps -u <felhasználó>, ...`
    - top és grafikus társai
  - Vezérlés
    - jelzésekkel: `kill <SIGNAL> <PID>`
      - leállítás, felfüggesztés
    - egyebek:
      - prioritás állítás: renice

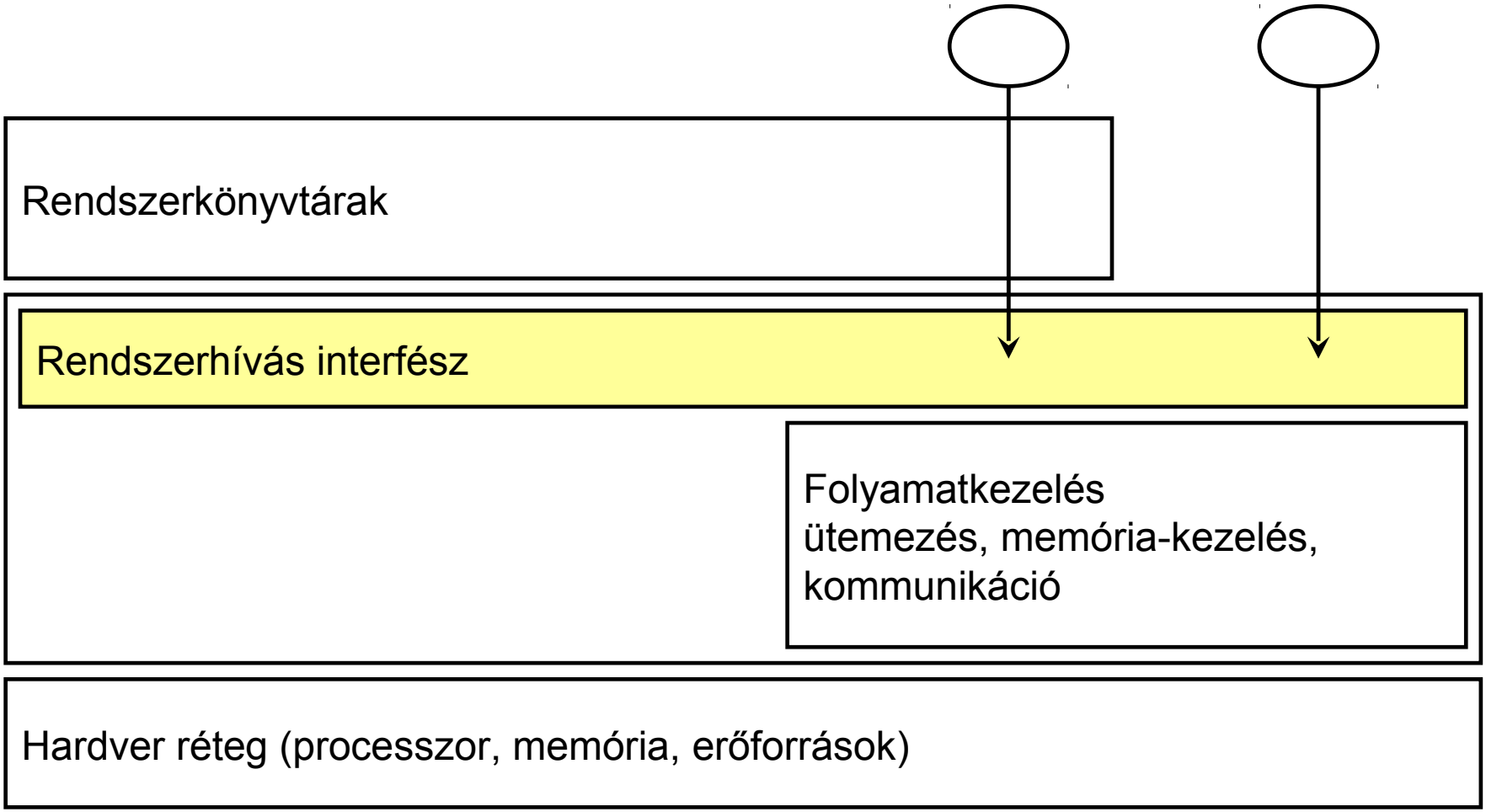
UNIX folyamatok – kernel alapismeretek

- 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
  - Felhasználói („szabad”) mód

- Végrehajtási környezetek:
  - Kernel (rendszer) kontextus
    - A kernel saját feladatainak ellátásához szükséges
  - Folyamat kontextus (virtuális memória-kezelés!)
    - Folyamat adatok (programszöveg, adatok, verem, stb.)
    - „**u-terület**”: adminisztratív adatok a folyamat címterében
    - **processz leíró**: adminisztratív adatok a kernel címterében

A folyamatok és a kernel (emlékeztető)



Folyamatok futtatása: végrehajtási mód és kontextus

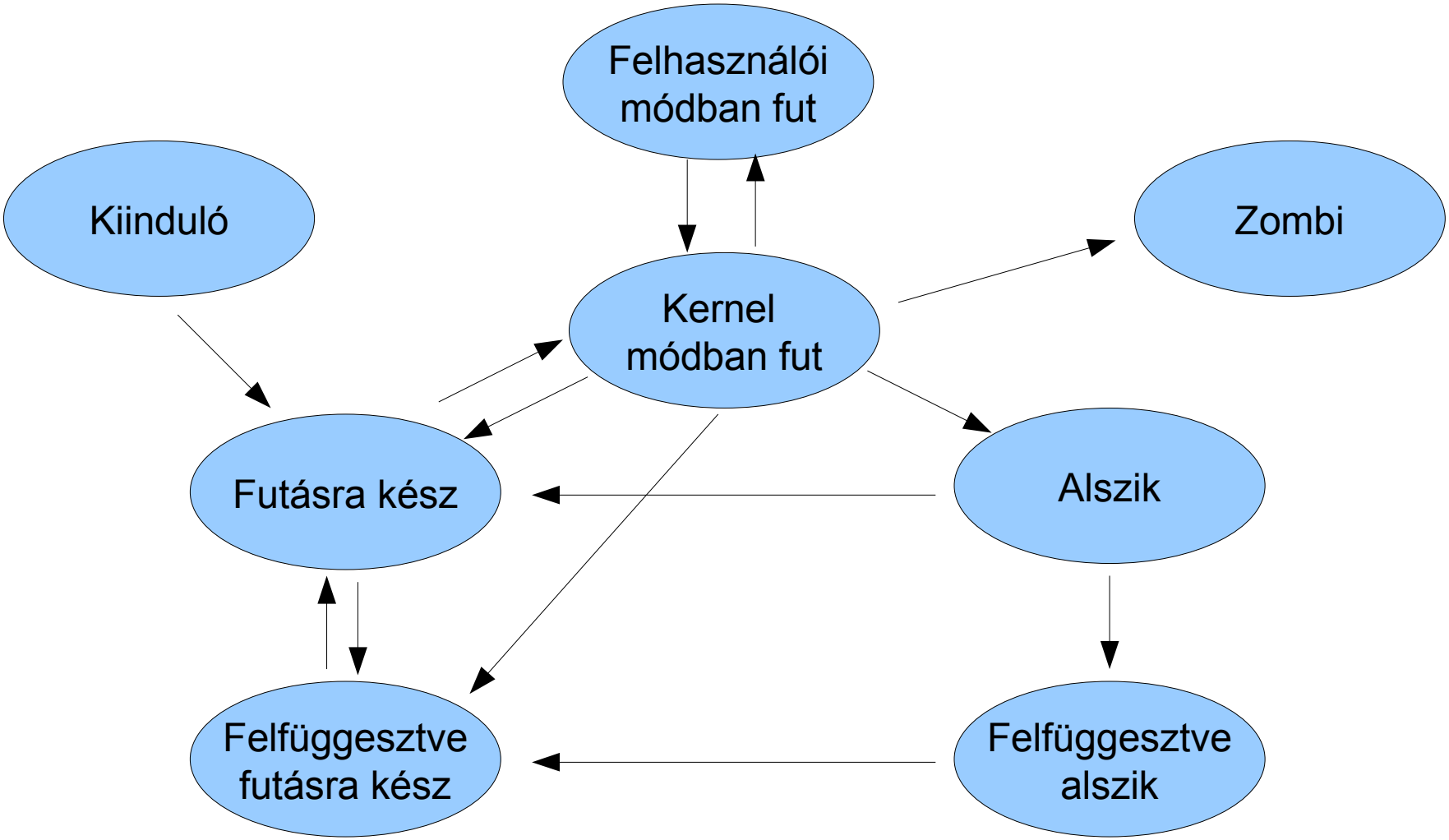
	felhasználói mód	kernel mód
folyamat kontextus	alkalmazás	rendszerhívás
kernel kontextus	(üres)	megszakítások, rendszer feladatok

Folyamatok adatai

- Saját adatok
  - programkód, verem, allokált memória, stb.

- Adminisztratív adatok
  - PID (Process ID): egyedi, a folyamatot azonosító szám
    - PPID: szülő folyamat azonosítója
  - A folyamat állapota
    - fut, alszik, stb. (l. következő fólia)
    - ütemezési információk (prioritás, CPU használat, nice érték)
  - Hitelesítők
    - UID, GID: a kapcsolódó felhasználó adatai
  - Memória-kezelési adatok
    - címleképezési térkép
  - Kommunikációs adatok
    - fájlleírók, jelzés információk
  - Statisztikák
    - erőforrás használat (számlázáshoz)

Folyamatok futtatása: állapotátmeneti gráf



Folyamatok életciklusa (állapotátmenetek)

- Létrehozás
  - fork(): új folyamat létrehozása (copy-on-write technika)
  - exec(): új programkód betöltése egy folyamat címterébe
- Futtatás (ütemezés)
  - futó és futásra kész állapotok közötti átmenetek vezérlése
- Futás megállítása
  - várakozás eseményre (önként)
  - felfüggesztés (felhasználó)
  - leállás (önként vagy a kernel)
- Leállás
  - exit() rendszerhívással
  - zombi állapot
  - szülő értesítése
  - gyerekek adoptálása

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 (init, idle, swapper)
  - minden folyamat őse
  - a rendszer leállásáig fut
  - örökli az árva folyamatokat
  - figyeli bizonyos rendszerfolyamatok meglétét (futását)
- A család fontos
  - a szülő értesítést kap a gyerek folyamat leállásáról (nyugtáznia kell)

UNIX folyamatok ütemezése (tradicionális UNIX)

(Korábbi előadásokon példával együtt szerepelt, itt csak ismételjük)

- Főbb tulajdonságok
  - prioritásos
  - preemptív
  - időosztásos
- A prioritás számításának összetevői
  - korábbi cpu használat
  - futásra kész folyamatok száma (p_cpu „öregítésével”)
  - nice érték (`nice` és `renice` parancsok)
- Algoritmus
  - több szinten, több időléptékben zajlik
  - óraütésenként a prioritási sorok ellenőrzése
  - 10 ütésenként round-robin körforgó ütemezés egy soron belül
  - 100 lépésenként a prioritások újraszámítása

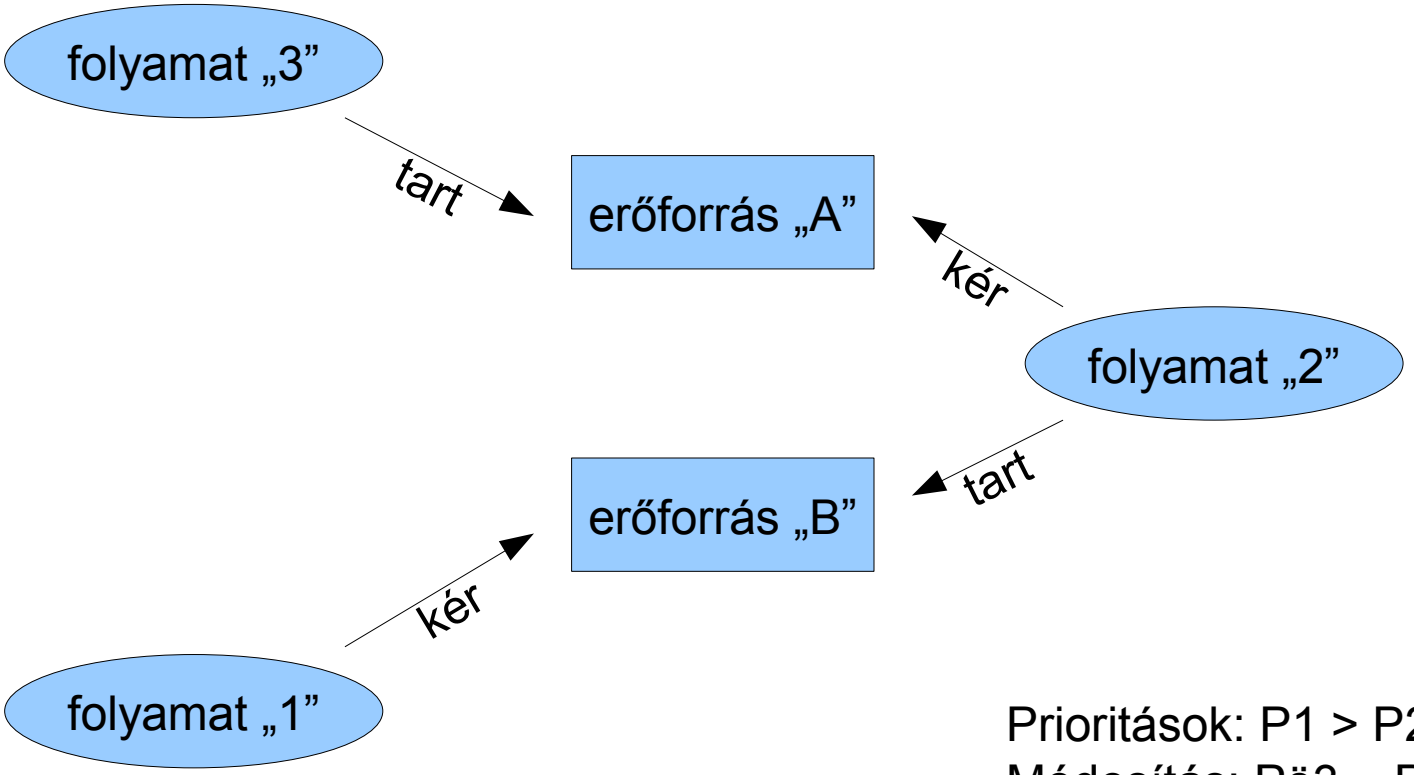
Modern UNIX ütemezők szempontjai

- Új ütemezési osztályok
  - speciális alkalmazási igények: multimédia, beágyazott, stb.
  - „fair share”: az erőforrások tervezhető elosztása
  - a kernelek is egyre több saját folyamatot (szálat) futtatnak
  - batch, real-time, kernel és interaktív folyamatok keveréke fut egy időben
- Moduláris ütemező rendszer
  - sokféle osztály – egyféle keretrendszer
  - egyszerűbb a speciális ütemezési megoldások megvalósítása
  - bővíthető
- Az ütemező erőforrásigénye (teljesítménye)
  - az ütemezés egyre összetettebb feladat
  - nem lenne jó az ütemezésre pazarolni a processzoridőt
  - az ütemezési algoritmus komplexitása lineáris, de inkább *O(1)* legyen
- Szálak kezelése (folyamatokat vagy szálakat ütemezünk?)
  - szálak számának növekedésével nő az ütemező terhelése (Java VM)
  - a szálak kerültek előtérbe – ennek megfelelő ütemező kell

Modern UNIX ütemezők szempontjai (folyt.)

- Kernel megszakíthatóság
  - a tradicionális UNIX kernel nem preemptív (szűk keresztmetszet)
  - a többprocesszoros (többmagos) ütemezéshez elengedhetetlen
- többprocesszoros, többmagos ütemezés és terheléselosztás
  - régóta tipikus a 4-8-16 processzoros rendszer (szervereken)
  - folyamatok (csoportok) processzorokhoz (csoportokhoz) kötése
  - garantált processzor erőforrások egyes folyamatok (csoportok) számára
- Lokalitás, processzor halmazok, hardver sajátosságok
  - multithreading, multi-core, multi-cpu – nem mindegy, milyen a hardver
  - processzorok – cache – memória viszonyok figyelembe vétele
  - pl. próbáljuk ismét ugyanarra a processzorra ütemezni a folyamatot
- Elosztott ütemezés
  - egy ütemezési sor? több? processzoronként? magonként?

Örökölt prioritások (Solaris)



Prioritások: P1 > P2 > P3
Módosítás: Pö3 = Pö2 = P1

Összefoglalás

- Folyamatokkal kapcsolatos alapismeretek
  - metaadataik (u-terület, processz leíró), PID
  - felhasználói kezelésük: `ps, kill, nice`
  - kernel kezelésük: kétféle futási mód, életciklus, ütemezés
- Folyamatok életciklusa
  - létrehozás: klasszikusan a `fork()` rendszerhívással
  - állapotok (speciális: kétféle futó állapot, felfüggesztett állapotok)
  - vezérlés alapvetően jelzésekkelű
- Folyamatok családfája
  - a `fork()` fát épít, az ős folyamat az `init`
- Ütemezés
  - klasszikus: prioritásos, időosztásos, preemptív
  - modern: moduláris, több ütemezési osztály, többprocesszoros