Linux Programozás

Ismerkedés a Linux Kernellel

A Kernel jellemzői

- Monolitikus (egy nagy program az egész)
  - A kezdeti fejlesztése könnyebb volt.
  - Optimalizálható
  - Kevésbé flexibilis
  - Az egyes részek nem fejleszthetőek egymástól függetlenül.
- Betölthető modulok támogatása
  - Pótolja a monolitikus kernelből adódó flexibilitás hiányt.
  - Tárgykódú állomány, amely menet közben linkelődik a kernelhez.
  - A monolitikus kernel miatt erősen verzió függő.

A boot folyamata

- A HW boot programja (BIOS, Bootstrap)
- Bootmanager (LILO, Grub, U-Boot)
- Linux kernel:
  - Betöltődik és kitömörödik
  - HW inicializálás
  - start_kernel() C függvény
  - Kernel inicializálás
  - init kernel szál elindulása
  - A swap elindítása
  - A rendszer konzol kezelése
  - A gyökér állományrendszer felcsatolása
- Az init program elindítása (/bin/init, /sbin/init, /etc/init)
  - Az /etc/inittab (/etc/init/) alapján a rendszer inicializációs scriptek elindítása az /etc/rc.d könyvtárból.

A Kernel feladatai

A Kernel uralkodik az erőforrások felett, ő osztja ki a versenyző folyamatoknak. A legfőbb erőforrások:

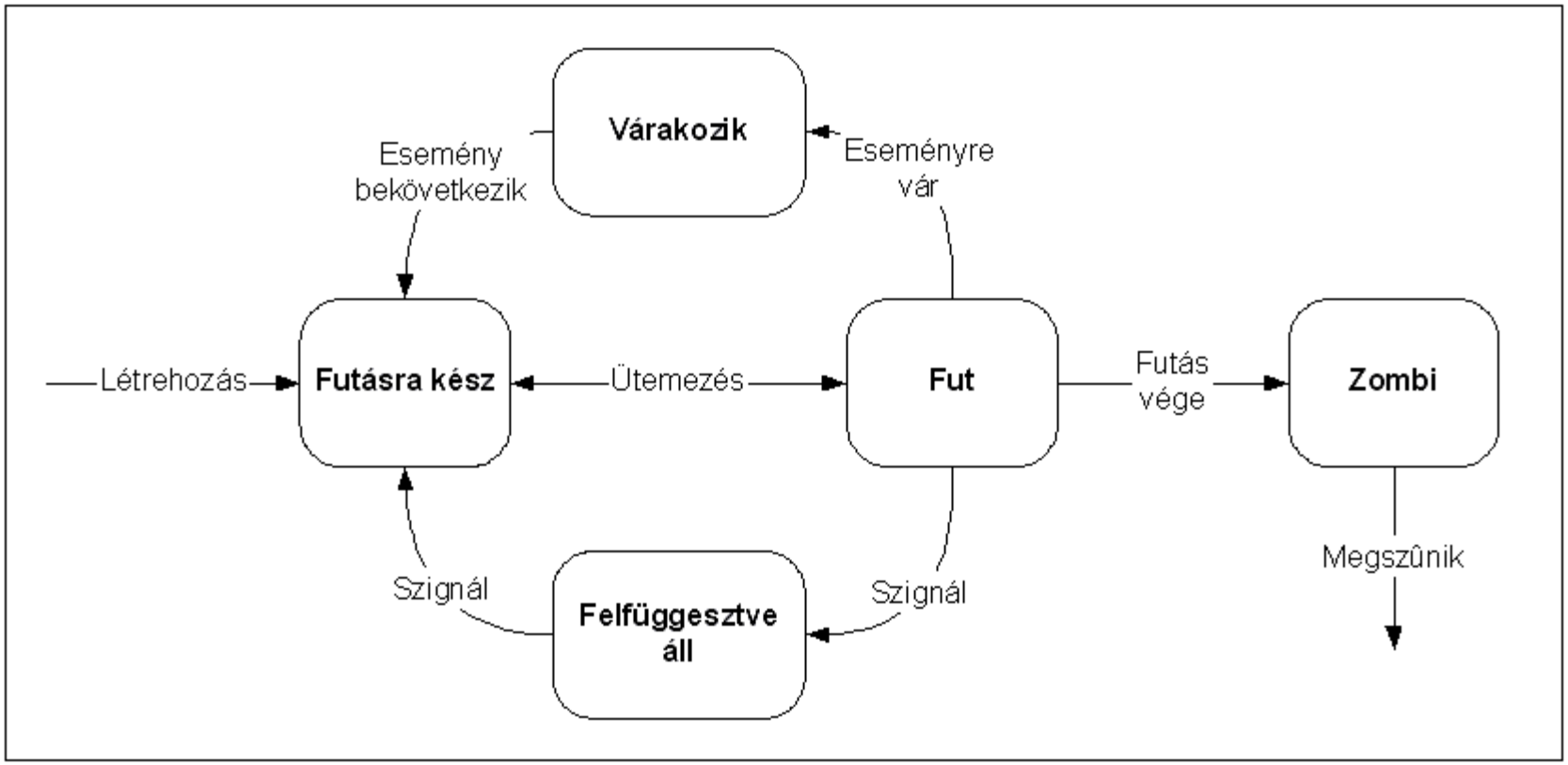
- CPU – Ütemezés
- Memória
- Állományrendszer
- Eszközök

Ütemezés

A Linux folyamataihoz kapcsolódó információk

- Állapot
- Azonosítók (pid, uid, gid)
- Kapcsolatok (ppid, pgid)
- Ütemezési információk
- Memória információk
- Fájlrendszer (home könyvtár, munka könyvtár)
- Folyamatok közötti kommunikáció (IPC)
- Idő és időzítők
- Processzor specifikus adatok

A folyamatok állapotai



Klasszikus ütemezés (1)

A klasszikus ütemezés a 2.4.x kernelig volt jelen.

- Az időt szeletekre osztja (jiffy).
- A szelet végén az ütemező dönt a folyamat váltásról.
- A folyamatok rendelkeznek statikus és dinamikus prioritással is.
  - Statikus: felhasználó állítja be
  - Dinamikus: a rendszer állítja az igazságossághoz
- A folyamat lemondhat az időszelet maradékáról, vagy várakozó állapotba kerülhet.
- Más processzek nem vehetik el a futás jogot. (Csak a kernel megszakítás kezelő rutinjai.)

Klasszikus ütemezés (2)

A folyamatokhoz választható ütemezési stratégiák:

- Normál ütemezés
- Valósidejű ütemezés
  - FIFO
  - Round-Robin

Ha valósidejű folyamat futásra kész, akkor az ütemező a következő váltásnál a legnagyobb prioritású valósidejű folyamatot választja.

Klasszikus ütemezés (3)

- Váltás csak az időszeletek végén történhet, vagy ha az éppen futó folyamat lemond az időszeletről.
- Az ütemező algoritmus futási ideje arányos a folyamatok számával. O(n)

Következés képen nem kiszámítható a reakció idő, csak bizonyos korlátok adhatóak. -> soft-realtime

O(1) ütemezés

- A 2.6-os kernelben jelent meg. (Molnár Ingo alkotása.)
- Az ütemező algoritmus futási ideje konstans, független a folyamatok számától.
- A Java virtuális gépek miatt volt rá szükség, a sok szál miatt.

Completely Fair Scheduled (CFS)

- A 2.6.23-as kerneltől az alapértelmezett ütemező. (Molnár Ingo alkotása ez is.)
- Nincsenek időszeletek (dinamikusan változnak).
- A CPU-t igazságosan osztja el a folyamatok között. (nanosec felbontással)
- A várakozási listát piros-fekete bináris fákban tárolja.
- Moduláris a felépítése:
  - Tartalmazza a korábbi normál és valós ütemezési stratégiákat.
  - Új ütemezési stratégia a „batch” a szerverek számára.
  - Szabadon bővíthető.

CFS (2)

- Az ütemezés O(log n) komplexitású, azonban…
  - A taszk váltás konstans idő.
  - A leváltott taszk beillesztése O(log n) idő.
  - Bár O(log n) normális esetben kisebb, mint volt.

CFS (3)

- A valósidejű folyamatok továbbra is megelőzik a többi folyamatot a választás során.
- Nincsenek időszeletek így a rendszer azonnal válthat.
- A taszk váltás a gyorsítótárazás miatt rövid idő alatt lezajlik.

Megszakítás kezelés (1)

Két megszakítás típust különböztetünk meg:

- Gyors: Letiltja a megszakításokat.
- Lassú: Nem tiltja le a megszakításokat.

(A „gyors” megszakítás kezelőt gyorsra kell megírni.)

Megszakítás kezelés (2)

A tipikus megszakítás kezelő implementáció:

„Top half”:

- A tényleges megszakítás kezelő.
- Csak a szükséges részeket tartalmazza.

„Bottom half”:

- Tasklet vagy workqueue
- Nem megszakítás időben fut.
- A kernel a „lehető leghamarabb” végrehajtja.

A Linux és a valósidő

Az eredeti kernel:

- Az eredeti 2.4-es kernelnél csak soft realtime-ról beszélhetünk.
- A kernel módosításával (patch-ek) a rendszer átalakítható hard realtime rendszerré.
- A kernel fejlődésével és a valósidejű módosítások integrálásával az aktuális (>2.6.23) verzió már közel valósidejű.

Memória kezelés

Memória kezelés a Linux rendszerben

- Virtuális memória kezelés lapozással
- A memóriát két részre osztja a rendszer:
  - Kernel space
    - Kernel férhet hozzá
    - Nincs lapcsere
    - Korlátos méret
    - A virtuális címek a fizikai címtől csak egy eltolással különböznek
  - User space
    - A felhasználói folyamatok és a Kernel is használja
    - Van lapcsere
    - Virtuális címek

Igény szerinti lapozás

- Csak akkor tölti be a lapot, amikor szükség van rá.
  - Pl. adatbázisnál nem tölti be a teljes adatbázist.
  - Kód futtatásnál nem tölti be a teljes program kódot.
  - Csak leképezi őket a memóriába, és a laphibák révén töltődnek be darabonként.

Lapcsere

- Ha nincs hely a fizikai memóriában lapokat kell eltávolítani:
  - Ha nem módosított – kidobjuk
  - Ha módosított – lapcsere állományba elmentjük
- Dirty flag
- **L**east **R**ecently **U**sed lapozási technika

Megosztott virtuális memória

- Egy lapot több folyamat címterébe képezünk le
- A leggyorsabb kommunikáció két folyamat között

Copy on write

- Új folyamat létrehozásakor nem másolódik le a teljes memória terület, csak leképeződik
- A lapok csak olvashatóak
- Ha írjuk a lapot megtörténik a másolás

Lap attribútumok

- A lap bent van-e a fizikai memóriában?
- Olvasható
- Írható
- Futtatható
- A kernel space/user space része.
- A lapot módosították-e (dirty)?
- A laphoz hozzáfértek-e?
- További, a gyorsító tárakkal kapcsolatos beállítások.