Az operációs rendszerek belső működése

Feladatkezelés

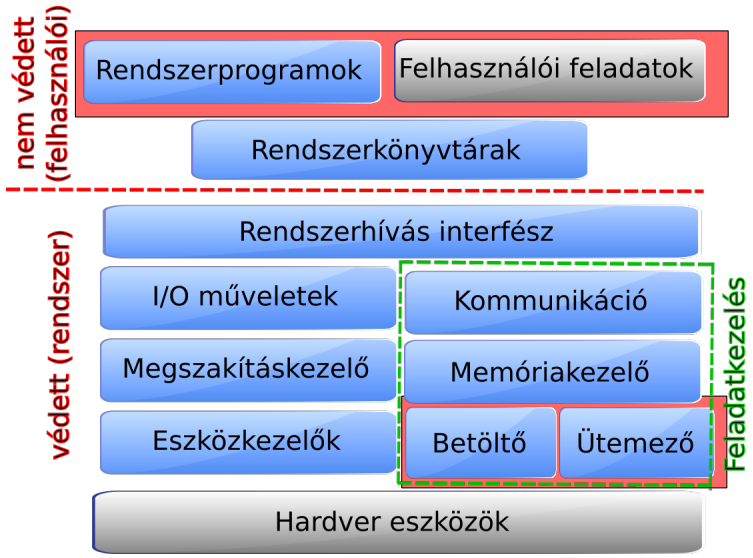
Mészáros Tamás
http://www.mit.bme.hu/~meszaros/

Budapesti Műszaki és Gazdaságtudományi Egyetem
Méréstechnika és Információs Rendszerek Tanszék

Az eddigiekben történt...

- Az operációs rendszer
  - feladatok végrehajtása
  - **vezérlőprogram**
  - **erőforrás-allokátor**

- Elvárások
  - feladatok egyidejű kiszolgálása
  - megbízható működés
  - esetenként valósidejűség



Az OS felépítése

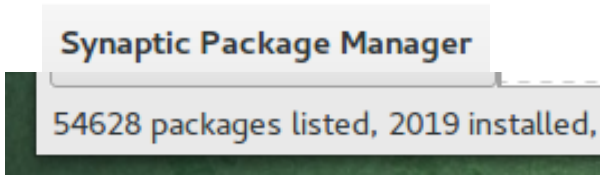
- Kialakulása
  - kötegelt rendszerek
  - **multiprogramozott**
  - **időosztásos**
  - beágyazott

Hogyan kezeljük a feladatokat?

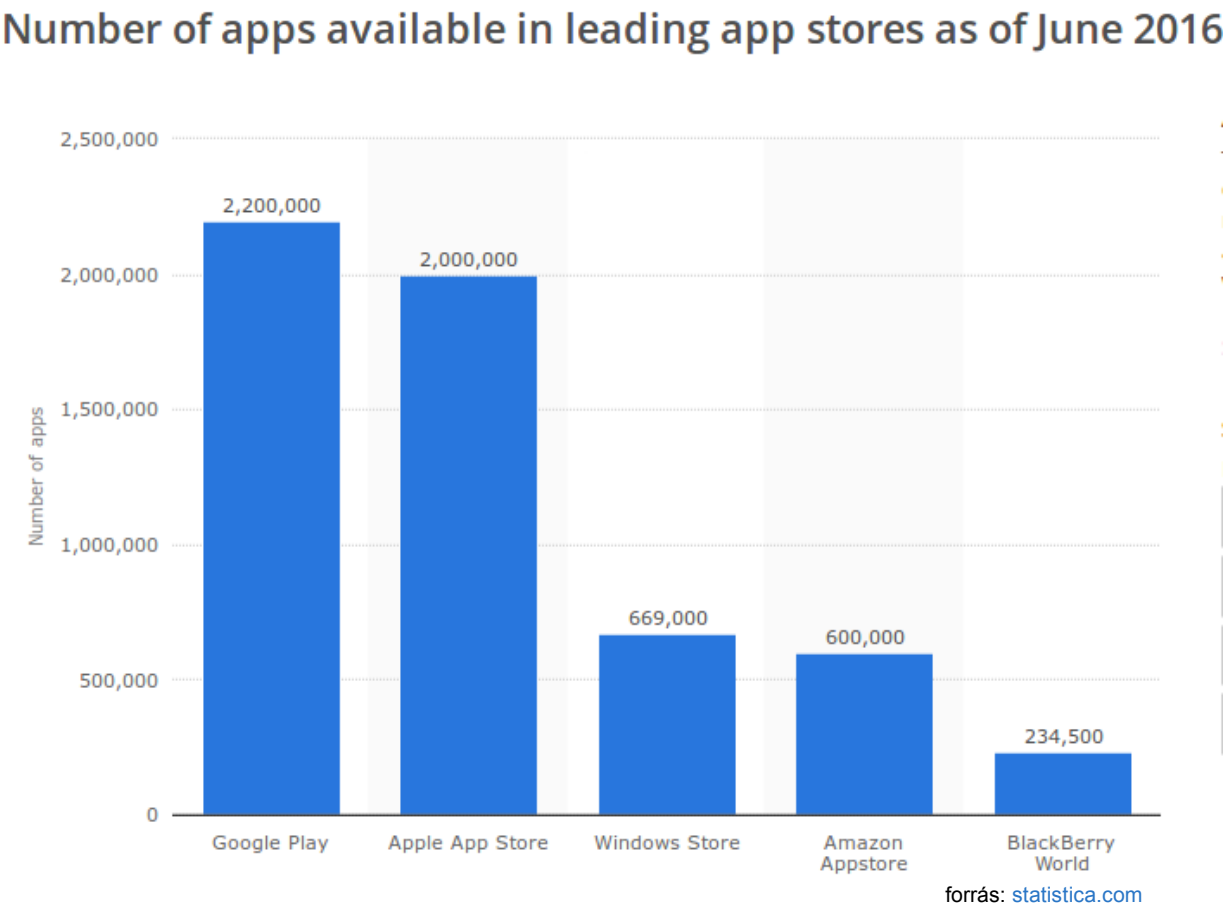
Milyen feladatokat futtatunk?

- A felhasználói feladatok sokszínűsége
- Az operációs rendszerek osztályozása (kliens, szerver, beágyazott…)

- Alkalmazások




```
> yum list all | wc -l
22747
```



forrás: statistica.com

A feladatok jellege

- I/O-intenzív feladatok
  - idejük nagy részét várakozással töltik (adatbetöltés, adatkiírás)
  - kevés processzoridőre van szükségük
  - pl.: fájlszerver, webszerver, email kliens és szerver stb.

- CPU-intenzív feladatok
  - idejük nagy részét a processzoron szeretnék tölteni
  - ehhez képest (relatíve) kevés I/O műveletre van szükségük
  - pl.: titkosítási és matematikai műveletek, összetett adatfeldolgozás stb.

- Memória-intenzív feladatok
  - egy időben nagy mennyiségű adat elérésére van szükségük
  - ha van elég, akkor CPU-intenzívek, ha nincs, akkor I/O-intenzív feladattá válnak
  - pl. nagy mátrixok szorzása, keresési indexek építése és használata stb.

- Speciális igények
  - valósidejű működés
  - filmnézés
  - …

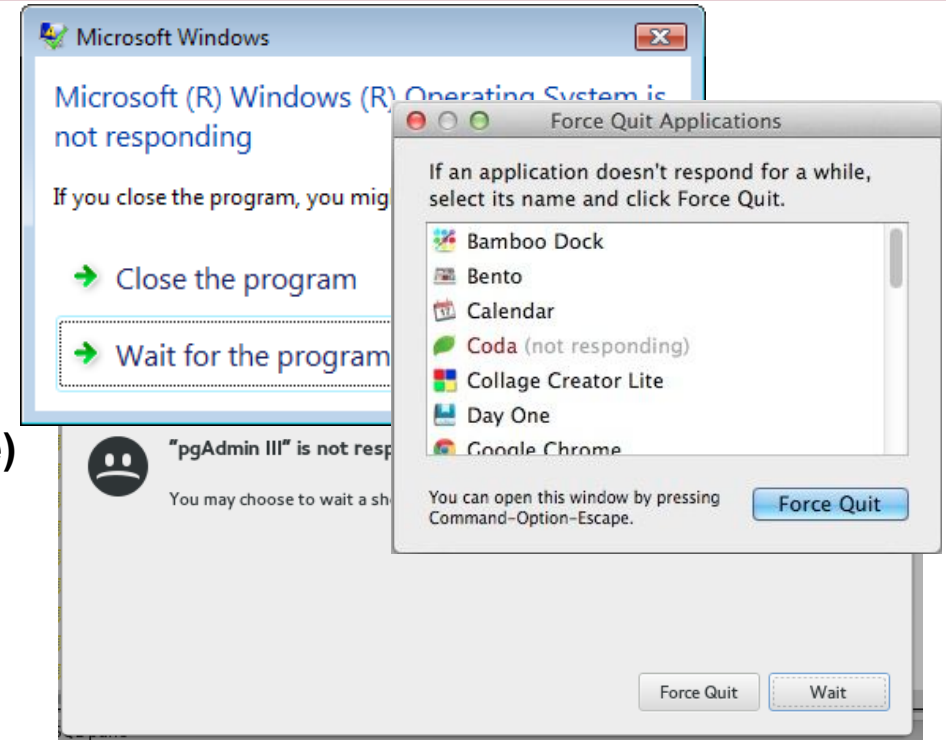
Elvárásaink

- Kevés várakozás

  várakozási idő (waiting time)

  körülfordulási idő (turnaround time)

  válaszidő (response time)



- Hatékonyság

  CPU-kihasználtság (CPU utilization)

  átbocsájtó képesség (throughput)

  rezsiköltség (overhead)

- Jósolhatóság, determinisztikusság

Az optimális feladat-végrehajtó rendszer

- A naiv felhasználó elvárásai
  - biztosítja feladatai végrehajtását
  - minimalizálja a várakozási és válaszidőt
  - az erőforrásokat (CPU, I/O) maximális kihasználja
  - minél kisebb rezsiköltséggel dolgozik

- Mit tapasztal a rendszer használata során?
  - egyes programok „lassan” futnak
  - mások ok nélkül „lefagynak”
  - „feleslegesen” erőforrásokat foglalnak
  - akadozik a filmnézés
  - gyorsan merül az akkumulátor
  - néha mintha az egész rendszer leállna
  - nem tudja fogadni a hívást
  - ...

Mi okozza a nehézségeket?

- Az OS nem lát a jövőbe
  - milyen feladatok jönnek
  - milyen jellegűek
- Sok a feladat
  - különböző elvárások
  - más az optimalitási kritérium
  - néha túl sok, „vergődik” a rendszer
- A feladatok hatással vannak egymásra
  - együttműködnek
  - versenyeznek
- Hibák
  - programozói
  - hardver

A taszk

- A feladatainkat programok hajtják végre
  elindulnak, működnek és befejeződnek

*A **taszk (task)** egy végrehajtás alatt álló program*

- Dinamikus entitás
  - a háttértáron tárolt program egy statikus program- és adathalmaz
  - a taszk „élő”: van működési állapota és életciklusa

állapot: adminisztratív jellemzők összessége egy adott pillanatban
létrejön, végrehajtás alatt áll a processzoron, várakozik valamire, befejeződik stb.

életciklus: a létrehozástól a befejeződésig terjedő állapotváltozások
az életciklus kezelése az operációs rendszer feladata

A feladat – taszk összerendelés

- Egy feladat – egy taszk

```
ps -ef
```

- Egy feladat – több taszk

```
ps -ef | wc -l
```

  - néha így gyorsabban megoldható a feladat
  - feladat dekompozíciója
  - nagyobb teljesítmény
  - jobb erőforrás-kihasználtság

- A taszkok kommunikálhatnak és együttműködhetnek
  - adatokat cserélhetnek egymással
  - szétoszthatnak részfeladatokat
  - egyesíthetnek részeredményeket
  - közös vezérlési szerkezeteket, kooperációs sémákat alakíthatnak ki

Taszkok szeparációja: az absztrakt virtuális gép

- Ideális esetben minden taszk teljesen önállóan fut
  - mintha saját gépen (erőforrásokon) futnának
- A valóságban osztoznak az erőforrásokon

absztrakt virtuális gép:
a kernel által biztosított erőforrások számítógépként elképzelt együttese.
virtuális CPU + virtuális memória

ismétlés: multiprogramozott rendszer
- M db processzor, N db taszk (N >> M)
- N db absztrakt virtuális gép leképezése a fizikai erőforrásokra

- A feladat-taszk összerendelés tovább bonyolítja a helyzetet
  - az absztrakt virtuális gép erős szeparációt jelent
  - ez nehezíti együttműködő taszkok kialakítását
    - adatcsere
    - vezérlési információk átadása

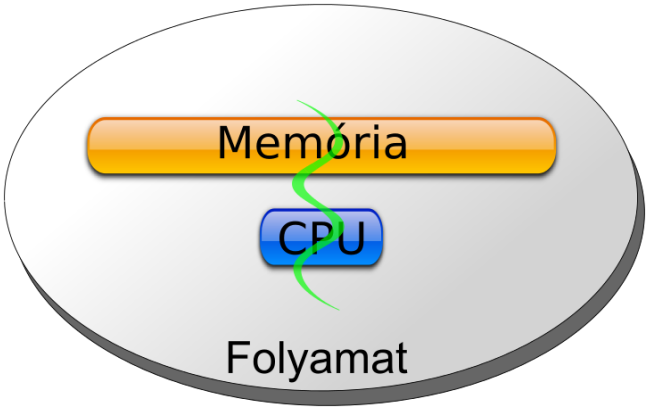
Taszk megvalósítások

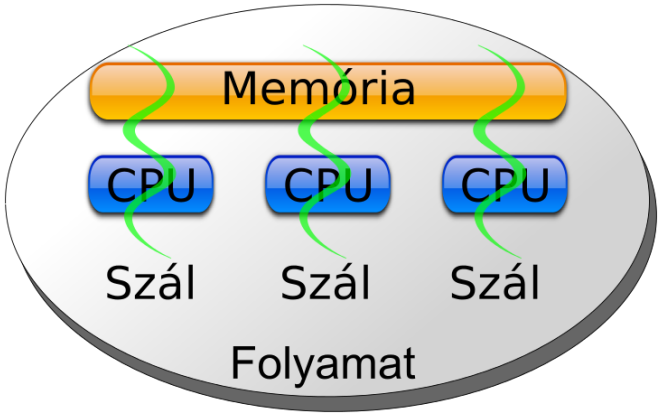
Folyamat **védelmi egység**

önálló memóriatartománnyal rendelkező taszk, amely szálakat tartalmazhat

Szál **végrehajtási egység**

szekvenciális működésű taszk, amely más szálakkal közös memóriát használhat





Folyamatok és szálak viszonya

- A szálak
  - önmagukban szekvenciálisan működnek → saját vermük van
  - **egy folyamaton belül**
    - egymással párhuzamosan működnek
    - közös memóriatartományt használnak
    - tudnak egymással adatot cserélni és kooperálni
    - nincs közöttük memóriaszeparáció
  - együttműködő taszkok megvalósítására alkalmas

- A folyamatok
  - önmagukban párhuzamosan is működhetnek
  - saját memóriatartományuk van
  - nem látják más folyamatok memóriatartományát (OS védelem)
  - ezért egymással nehezebben tudnak együttműködni

- Demo: folyamatok és szálak listázása

Folyamatot vagy szálakat használjak?

- Feladat – taszk vs. folyamat – szál
  - Megkívánja a feladatom a multiprogramozást?
  - Hány párhuzamos végrehajtóegységre van szükségem?
  - Milyen gyakran?
  - Elérhető a szál/folyamat az adott rendszeren?

- Szálak előnyei és hátrányai
  - kisebb az erőforrásigény
  - egyszerűbb létrehozás
  - egyszerű kommunikáció, ha egy folyamaton belül vannak
  - nem mindenütt érhető el
  - gondosabb programozást igényel (lásd később)

- Folyamatok előnyei és hátrányai
  - kernelszintű védelem
  - elterjedtebb
  - nagyobb erőforrásigény
  - nehézkesebb kommunikáció

Folyamatok és szálak teljesítménye (demo)

- Apache Multi-Processing Modules (MPM)
  - folyamatalapú: „Prefork”
    - PHP modul
  - folyamat + szálak: „Worker”
    - gondban lehet, ha a szálakban hiba következik be
  - szálak: „Event”

- Teszteljük a webszerver teljesítményét és a rendszer terhelését
  - Apache Benchmarking: ab

```
ab -n 5000 -c 500 <URL>
```

  - Szerver monitorozás: atop, apachetop

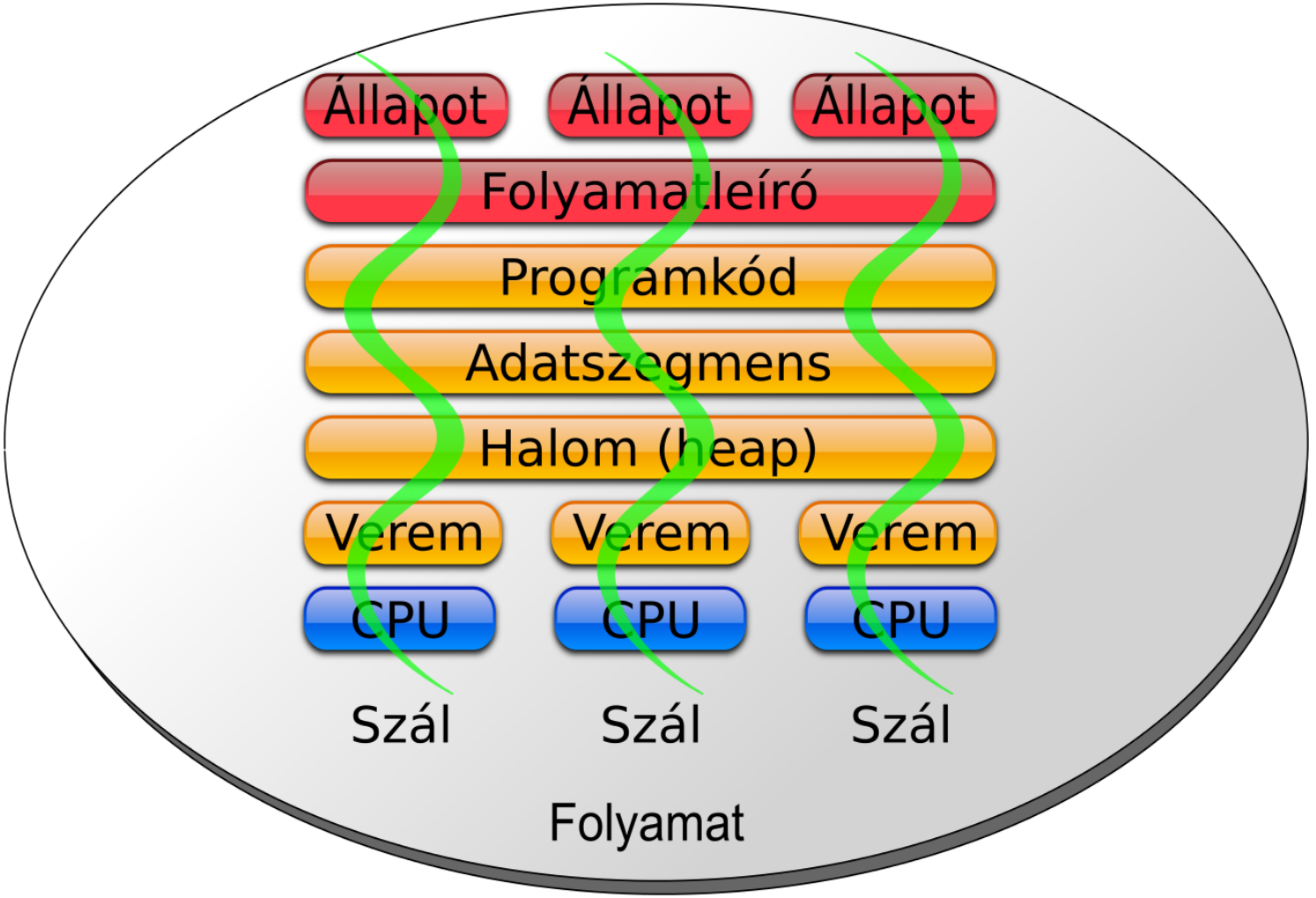
A taszkok adatai

- Saját
  - programkód
  - statikusan allokált adatok
  - verem, átmeneti adattár pl. függvényhívások számára
  - halom, a futásidőben, dinamikusan allokált adattár
    (demó)

- Adminisztratív (kernel)

  taszk- (folyamat-, szál-) leíró
  - egyedi azonosító (**PID**, TID)
  - **állapot** (l. később)
  - a taszk **kontextusa**: a végrehajtási állapot leírója
    - utasításszámláló (PC) és más CPU regiszterei
    - ütemezési információk
    - memóriakezelési adatok (MMU állapot)
  - tulajdonos és jogosultságok
  - I/O állapotinformációk
  - …
    (demó)

Folyamat és szálak – részletesebben



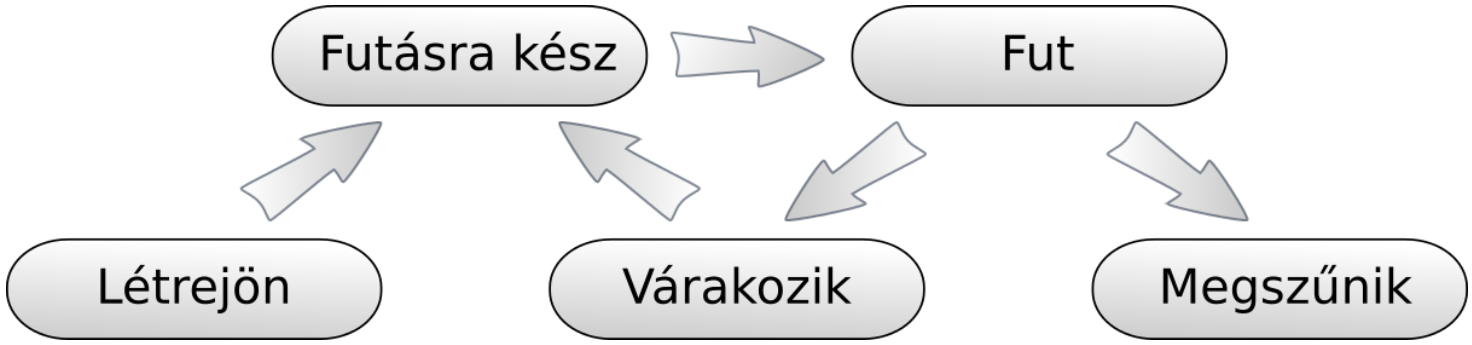
Hol tároljuk a taszkok adminisztratív adatait?

- A kernel memóriatartományában?
- A folyamat címterében?
- Mikor van szükség az adatokra?
  - „gyakran”, a kernel működése során → kerüljenek a kernel címterébe
  - a folyamat működése során, ritkábban → kerüljenek a folyamat címterébe
- Az adminisztratív adatok csoportosítása

	UNIX példa
elsősorban a folyamat futása során szükségesek	**u-terület**
• hozzáférés-szabályozás adatai	**folyamat címtér**
• rendszerhívások állapotai és adatai	
• IO műveletek adatai	
• számlázási és statisztikai adatok, stb.	
elsősorban a folyamatok kezeléséhez szükségesek	**proc struktúra**
• azonosítók	**kernel címtér**
• futási állapot és ütemezési adatok	
• memóriakezelési adatok	

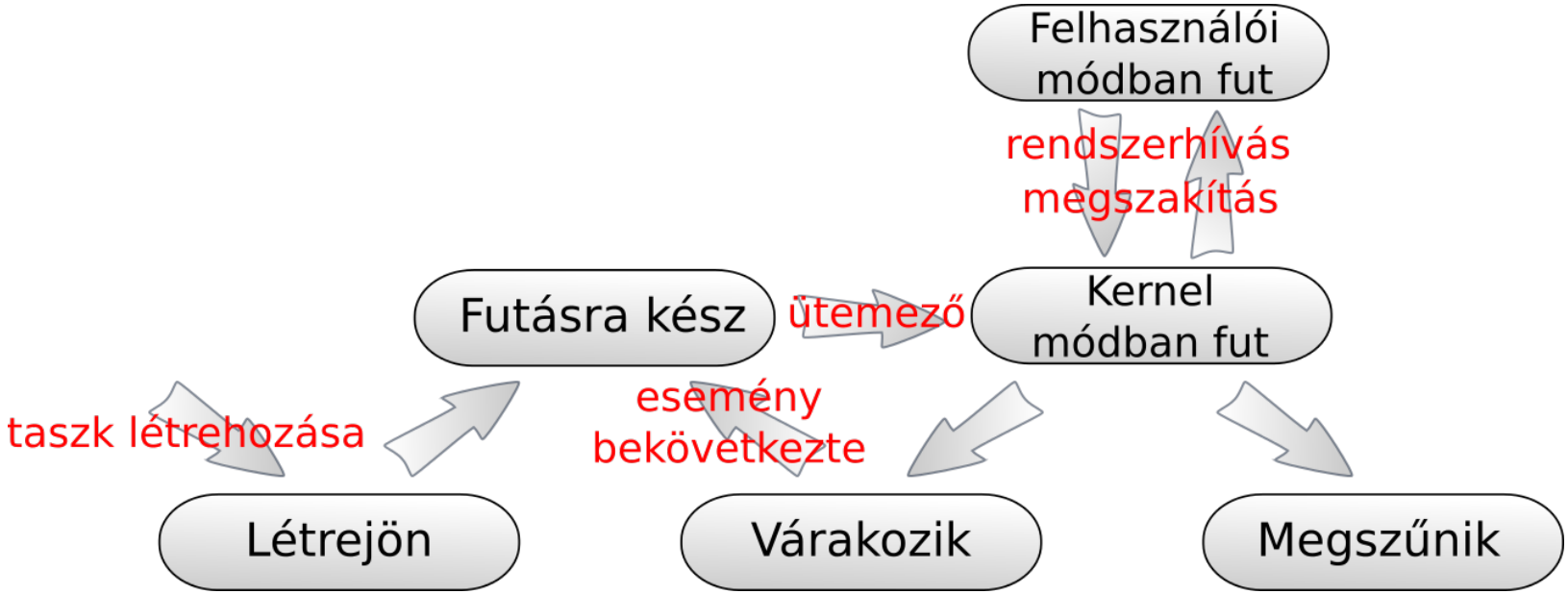
A taszkok állapotai és életciklusa

- **Létrejön** (created)
  - betöltődik és elindul a taszk programja
  - a kernel létrehozza a szükséges adatstruktúrákat és bejegyzéseket
  - a taszk futásra kész állapotba lép
- Működése során
  - **Futásra kész** (ready to run)
  - **Fut** (running)
  - **Várakozik** (waiting) avagy **blokkolt** (blocked)
- **Megszűnik** (exit, terminated)
  - önszántából vagy végzetes hiba hatására



A taszkok állapotátmenetei

- Állapotváltozás rendszerhívások és megszakítások hatására
  - a rendszerhívás is megszakítást eredményez
    - → az állapotváltozások megszakítások hatására következnek be
    - → **a kernelek megszakítás-, azaz eseményvezéreltek**



Hogyan jön létre a taszk?

- Az init illetve a Wininit services.exe elindítja a szolgáltatásokat
- A felhasználó bejelentkezik (Logon) és elindít programokat

- Példa: Windows szálkezelés (Szoftvertechnikák)
- Példa: Unix folyamatok létrehozása: fork() és exec()

```
if ((res = fork()) == 0) {      // gyerek ága

   exec(...);                   // programkód betöltése

} else if ( res < 0 ) {         // szülő ága, hibaellenőrzés
   ...
}

// res = CHILD_PID (>0), szülő kódja fut tovább
```

Szálak létrehozása (példa)

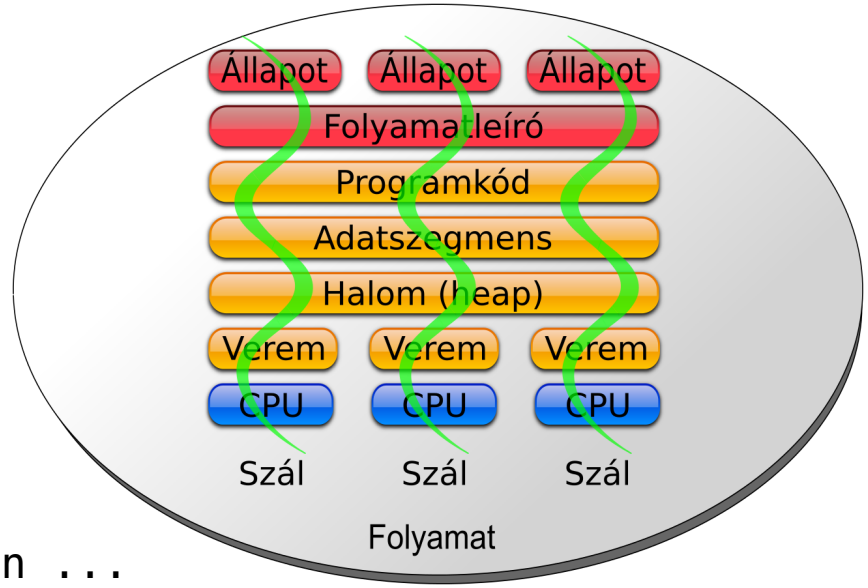
Párhuzamos adatfeldolgozás szálakkal

```
struct data_type data[N];

pthread_t *tid;

for (i=0; i < N; ++i) {
  // szálak indítása
  pthread_create(&tid[i], NULL, compute, &data[i]);
}

for (i=0; i < N; ++i) {
  // megvárjuk, míg elkészülnek
  pthread_join(tid[i], NULL);
}
```



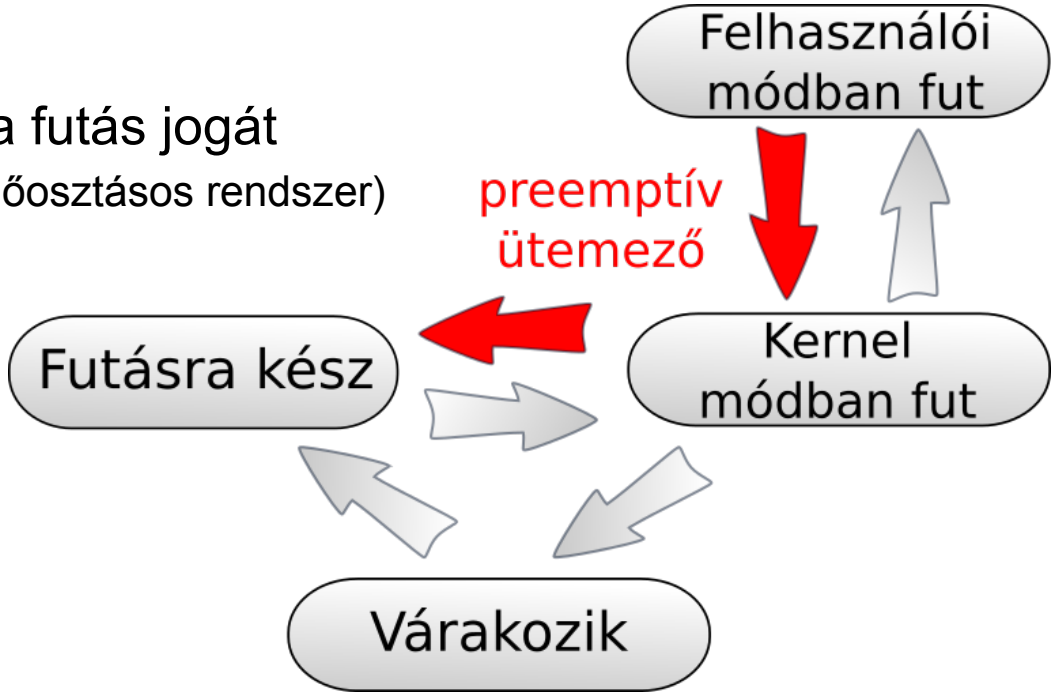

```
compute (void *part) {
  … műveletek a data[] *part részén ...
}
```

Unix folyamatok családfája

- Folyamatot csak egy másik folyamat tud létrehozni (`fork()`)
  - szülők – gyerekek – leszármazottak
  - családfa
- A szülő változhat
  - leálló folyamatok gyerekeit az init örökli
- A család fontos
  - a szülő nyilvántarthatja a gyerekfolyamatait
  - értesítést kap a gyerek folyamat leállásáról (nyugtáznia kell)
- pstree demo

Mikor vált taszkot a processzor?

- A futó taszk lemond a futás jogáról
  - exit()
  - rendszerhívás
- A futó taszk elveszíti a futás jogát
  - lejár az időszelete (l. időosztásos rendszer)
  - hibás működés miatt



- Megszakítás, kivétel hatására
  - a CPU megszakítja a normál működését
  - elindul a kernel megszakítás- ill. kivételkezelője

A kontextusváltás

- Kontextus: állapotleíró
  - utasításszámláló (PC), CPU és MMU állapot stb.
- Sok kontextusváltás történik az OS működése során
  - a rezsiköltség minimalizálandó
  - nem mindig menti a teljes kontextust
- Taszkváltás → kontextusváltás (taszk → taszk)
  - jelenlegi taszk kontextusának mentése
  - korábbi taszk kontextusának helyreállítása
- Megszakítás → kontextusváltás (taszk → kernel, üzemmódváltás!)
  - a kontextus egy kis része hardver támogatással elmentődik
  - megszakításkezelés
  - a visszatérés során visszaállítódik a korábbi végrehajtási kontextus

Kontextusváltások megfigyelése (demo)

- A kernel adattábláinak fájlrendszer interfészén keresztül lehetséges
  - a `/proc/stat` fájl `ctxt` mezője
  - a `/proc/<PID>/status` fájlban `ctxt voluntary_ctxt_switches` és `nonvoluntary_ctxt_switches`

- A korábbi Apache terhelésvizsgálatot megismételve
  - figyeljük meg az összes kontextusváltások számát
  - egy `httpd` folyamat kontextusváltásainak számát
  - Miért alacsony a `nonvoluntary_ctxt_switches` értéke?
  - Milyen jellegű folyamat az Apache `httpd`?

- CPU-intenzív folyamat kontextusváltásainak megfigyelése

  pl.: `stress -c 1`
  - az általa indított gyerekfolyamat kontextusváltásait nézzük meg
  - Hogyan változik a `nonvoluntary_ctxt_switches` értéke?
  - Ezt megfigyelve milyen ütemezőt használ az operációs rendszerünk?

- A kísérlet Windows alatt is elvégezhető.

Végrehajtási mód és kontextus

	felhasználói mód	kernel (védett) mód
taszk kontextus	fut a taszk saját programja	a taszk rendszerhívást hajt végre
kernel kontextus	(üres)	megszakítások, rendszerfeladatok kezelése

Összefoglalás

- Sokféle feladat
  - I/O-intenzív, CPU-intenzív, valósidejű, multimédia, …
- Sokrétű elvárások
  - idő: várakozási idő, válaszidő, körülfordulási idő
  - hatékonyság: átbocsájtó képesség, CPU kihasználtság, rezsiköltség
- A feladatkezelés alapjai
  - taszk: végrehajtás alatt álló program
    kontextus, állapot és életciklus (létrejön – fut, FK, vár – megszűnik)
  - absztrakt virtuális gép
  - folyamat
  - szál
- Kontextusváltás
  - megszakítások hatására
  - rendkívül gyakori