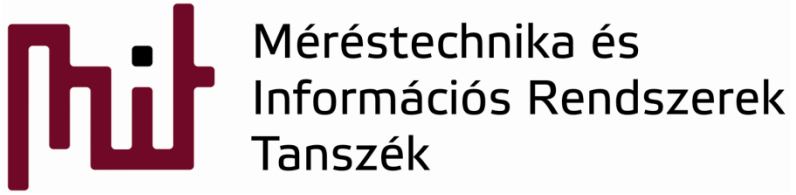
Feladatok (task) kezelése multiprogramozott operációs rendszerekben

dr. Kovácsházy Tamás

2. anyagrész, Ütemezés



Történeti háttér

- Batch rendszerek...
- Spooling...
- Multiprogramozás
  - 1 processzor (végrehajtó egység), M feladat (task)
  - Feladat típusok
    - Rendszerfeladatok
    - Batch feladatok
    - On-line feladatok (felhasználók)
      - Egy felhasználónak akár több független vagy függő feladata lehet
      - Egy feladat akár több párhuzamos részfeladatra is osztható
  - Feladat készlet (Job pool)
  - A feladatok végrehajtásához erőforrások szükségesek

Feladat megfogalmazása

- A multiprogramozás során a cél az, hogy a feladat készletet az rendszer optimálisan hajtsa végre.
- Mit jelent az, hogy optimális?
  - Alkalmazásfüggő szempontok...
  - Egészen más az optimális egy notebook, szerver, vagy kemény valós idejű beágyazott rendszer (ESP, ABS, stb.) esetén.
  - Sokat fogunk erről a kérdésről beszélni...
- Alapesetben a feladatoknak nem szabad tudniuk egymásról:
  - Egy-egy külön „virtuális gépen” futnak.
    - Virtuálisan saját CPU és memória
  - De osztozniuk kell a rendszer erőforrásain, szinkronizálódniuk (együttműködés), kommunikálniuk kell egymással.
  - Ezeket a feladatokat a feladatok nem tudják önmaguk megoldani, az operációs rendszert (annak a szolgáltatásait) kell használniuk.
    - Az OS a „nagy machinátor”, a „diktátor”, stb. a rendszerben.
    - A számítógép államformája a „felvilágosult diktatúra”.

Egy példa más területről 1.

- Kórházi osztály (SOTE-és hallgatók is VOLTAK itt. ☺)
  - Projekt menedzsment...
- Végrehajtó egységek
  - Orvosok, nővérek, kisegítő személyzet (heterogén többprocesszoros rendszer)
  - Egyben ők a rendszer legfontosabb erőforrásai is!
- Erőforrások (memória, perifériák, energia, stb.)
  - Vizsgálók, drága műszerek, stb.
    - Adott számú áll rendelkezésre belőlük.
    - Egy adott feladathoz ezek egy szabályrendszer szerint kell hogy rendelkezésre álljanak.
  - Anyagok és egyéb szükségletek (gyógyszer, energia, stb.)
    - Minél kevesebbet, de eleget kell használnunk.
  - Az erőforrásokhoz történő hozzáférés időbe kerül.

Egy példa más területről 2.

- Feladatok: Betegek és egyéb működéssel kapcsolatos „dolgok” ellátása.
- Ütemezés (scheduling, scheduler):
  - Főorvos, főnővér, stb. kiosztja a munkát.
    - Ki mit és mikor csinál a végrehajtó egységek közül.
  - Mi történik, ha a feladat készlet megváltozik?
    - Pl. feladat befejeződik, egy beteg állapota rosszabbodik, stb.
    - Újraosztják a munkát.
  - A végrehajtó egységek hogyan használják az erőforrásokat?
- A számítógép esetén a feladat egyszerűbb...

Operációs rendszerek

- Alapfogalmak tisztázása.
- Fárasztó lesz, de sajnos elkerülhetetlen...

Esemény (event)

- A rendszer életében lezajló változás vagy történés
- Belső esemény
  - Szoftver megszakítás vagy kivétel
- Külső esemény
  - Hardver megszakítás
- A modern operációs rendszerek megszakítás vezéreltek!
- „Eseményvezéreltnek” is hívják (event driven) ezért az OS-eket...

Feladat (task)

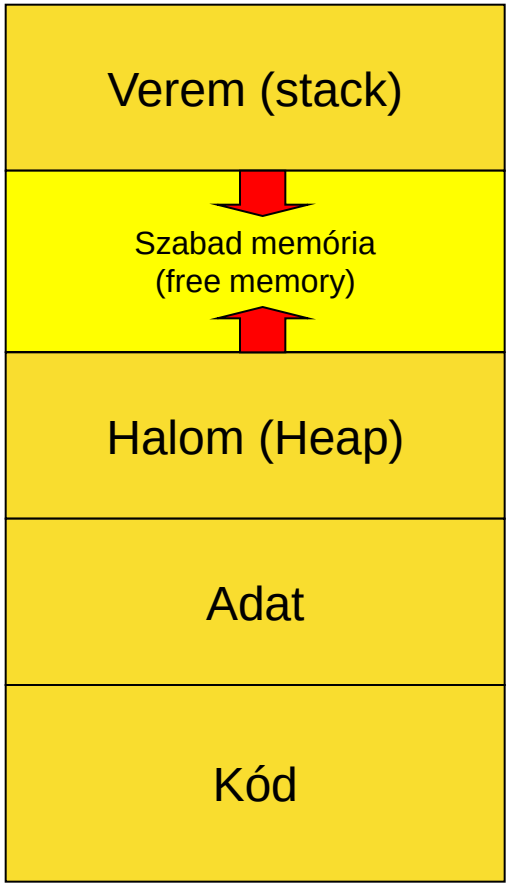
- A feladat (task) fogalmat absztrakt módon használjuk.
  - Később lesznek speciális, a végrehajtás/működés részleteit is megadó fogalmaink (folyamat, szál).
  - Hívják munkának (job) is.
  - Van ahol általánosan is a folyamat (process) szót használják.
  - Ez még az OS API-k szintjén is keveredik.
    - Pl. egyes OS-ekben a CreateTask() nagyon különböző dolgokat csinálhat...
    - Nem az a kérdés, hogyan hívják, hanem hogy mit csinál, vagyis a részletek megismerése nem kerülhető el egy adott OS használata esetén.

Feladat (task) 1.

- A multiprogramozott operációs rendszerek egyik legfontosabb alapfogalma.
- A feladatokat folyamatként, vagy együttműködő folyamatok együtteseként valósíthatjuk meg (implementáció)
- Műveletek meghatározott sorrendben történő végrehajtása.
  - Elkezdődik, szekvenciálisan végrehajt utasításokat, befejeződik.
    - Ezt a definíciót később kibővítjük (lesz egy „szál” fogalom is)
  - A feladat egy végrehajtás alatt álló program:
    - A program betöltése, végrehajtása, és befejezése az operációs rendszer egyik fontos feladata (nem sok szó lesz róla, mindent megcsinál nekünk a fejlesztőrendszer, ha meg nem, akkor megnézzük a részleteket).

Feladat (task) 2.

- A feladat több mint a program:
  - Aktív, nem passzív! (Virtuális CPU)
  - Állapota van (hiszen egy futó program):
    - Minimálisan: Fut (run), vár (waiting), futásra kész (ready).
    - Erősen OS függő állapotok és állapotátmenetek.
    - Részletesen fogunk vele foglalkozni.
  - Adatszerkezetek adattal vannak feltöltve (függ a folyamat előéletétől):
    - Virtuális memória (OS + HW)
    - Adat terület (globális adatok 💣).
    - Verem (stack),
    - Halom (Heap).
    - Erősen egyszerűsített ábra jobbra.



Feladatok állapota 1.

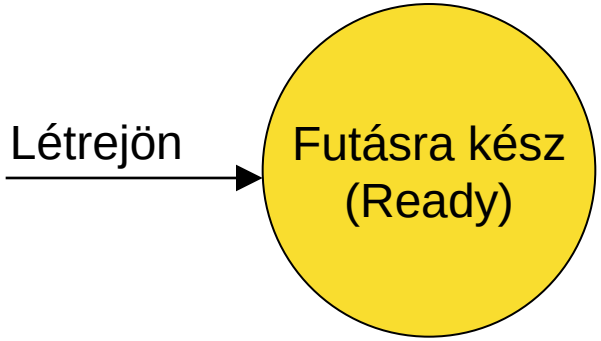
- Egyszerűsített állapot-átmeneti ábrát fogunk vizsgálni.
- Ennél több, OS specifikus állapot és állapotátmenet szokott lenni.

Feladat állapota 2.

- Minden feladat először létrejön...

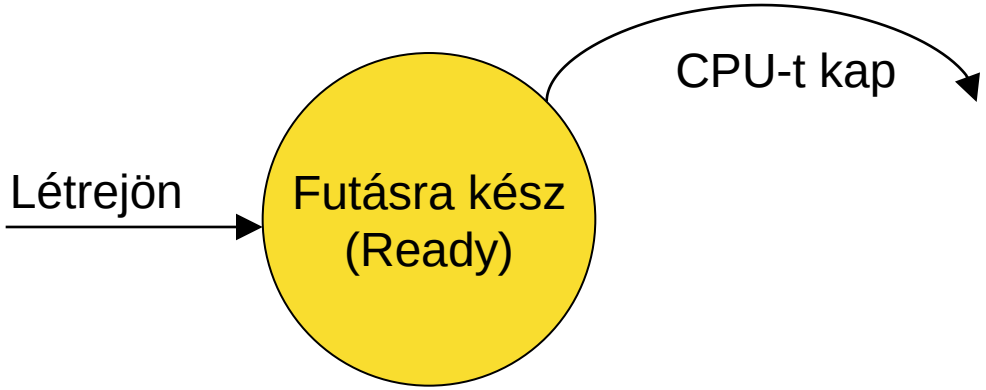
Feladatok állapota 3.

- Ekkor többnyire „Futásra kész” (Ready) állapotba kerül.
- Az ilyen feladatnak minden erőforrás rendelkezésére áll, kivéve a CPU.



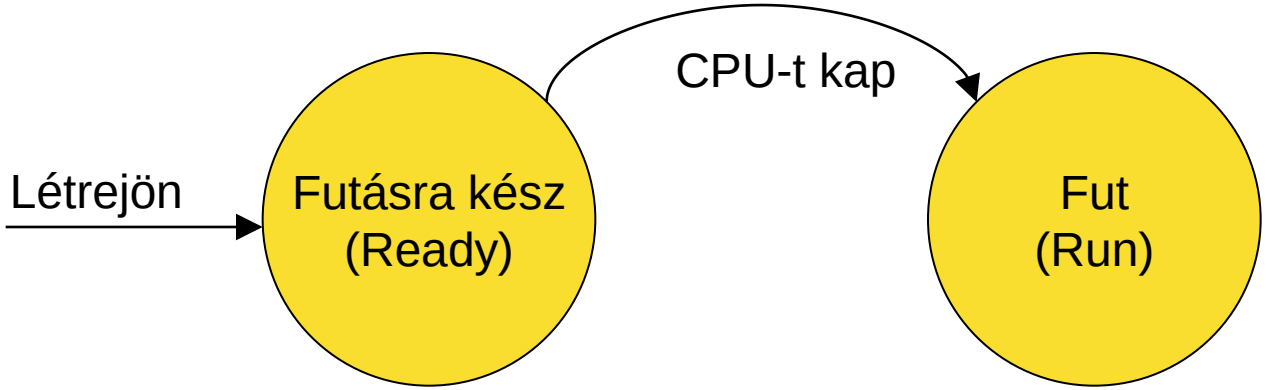
Feladatok állapota 4.

- Ha a CPU felszabadul, akkor egy futásra kész feladat futó állapotba kerülhet.
- Milyen algoritmus dönti el, hogy a futásra kész feladatok közül melyik kap CPU-t? A CPU ütemezés (CPU scheduling).



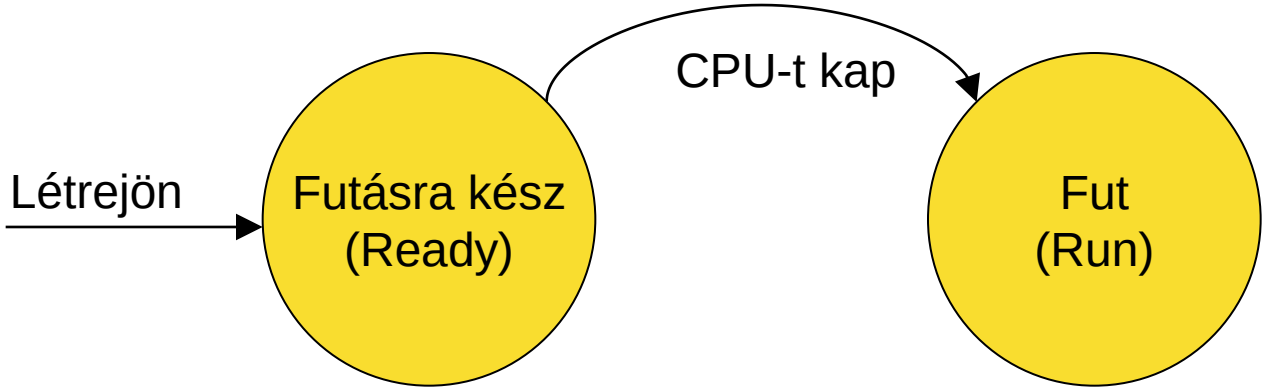
Feladatok állapota 5.

- A feladat „Futó” (Run) állapotban van (övé a processzor).



Feladatok állapota 5.

- A feladat „Futó” (Run) állapotban van (övé a processzor).

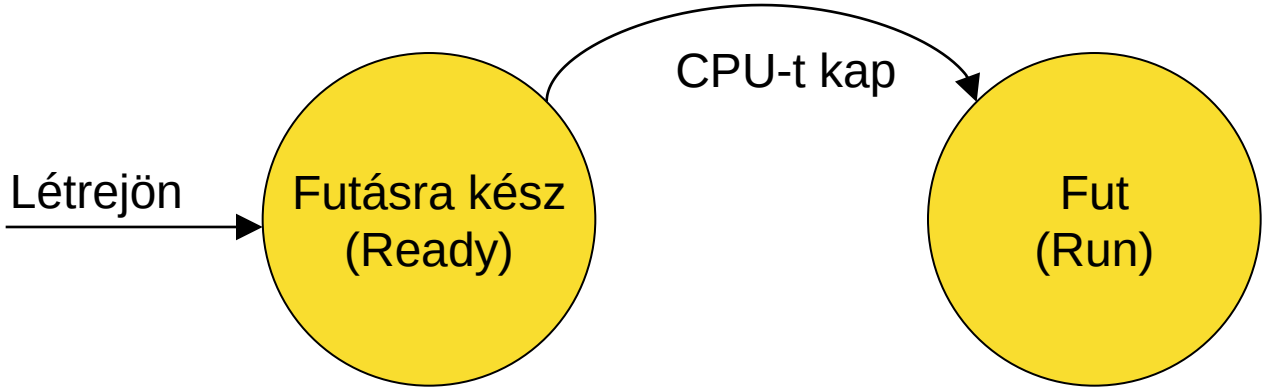


Mi fut, ha nem fut semmi?
Valaminek futnia kell egy CPU-un, ha az be van kapcsolva!

- Idle feladat
  - Alacsony prioritású háttérfeladatok
  - Power menedzsment

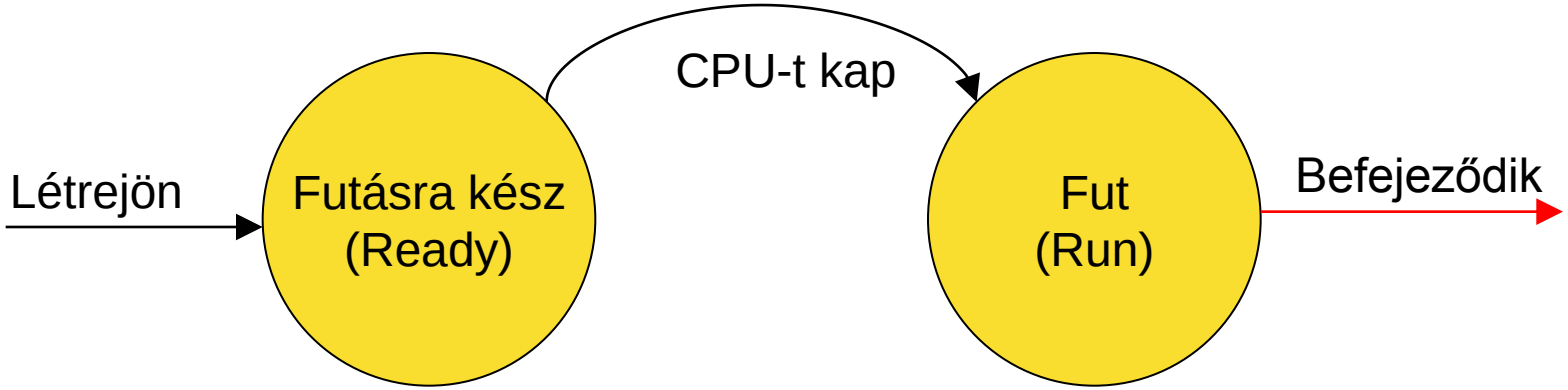
Feladatok állapota 5.

- A feladat „Futó” (Run) állapotban van (övé a processzor).
- Mi veheti el tőle a CPU-t? Vizsgáljuk meg a lehetőségeket.



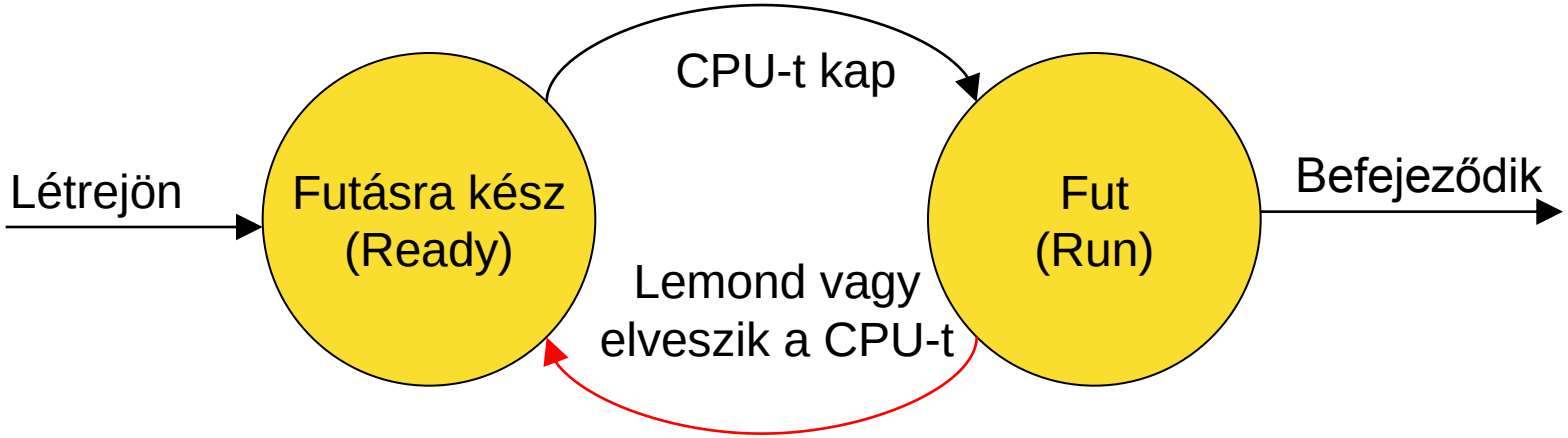
Feladat állapota 6a.

- A feladat befejeződik (kilép, terminál, stb.).
- Ezt a feladat akarja így (exit() rendszerhívást hívja).
- Hibakezelés esetén további állapotok szükségesek többnyire.



Feladatok állapota 6b.

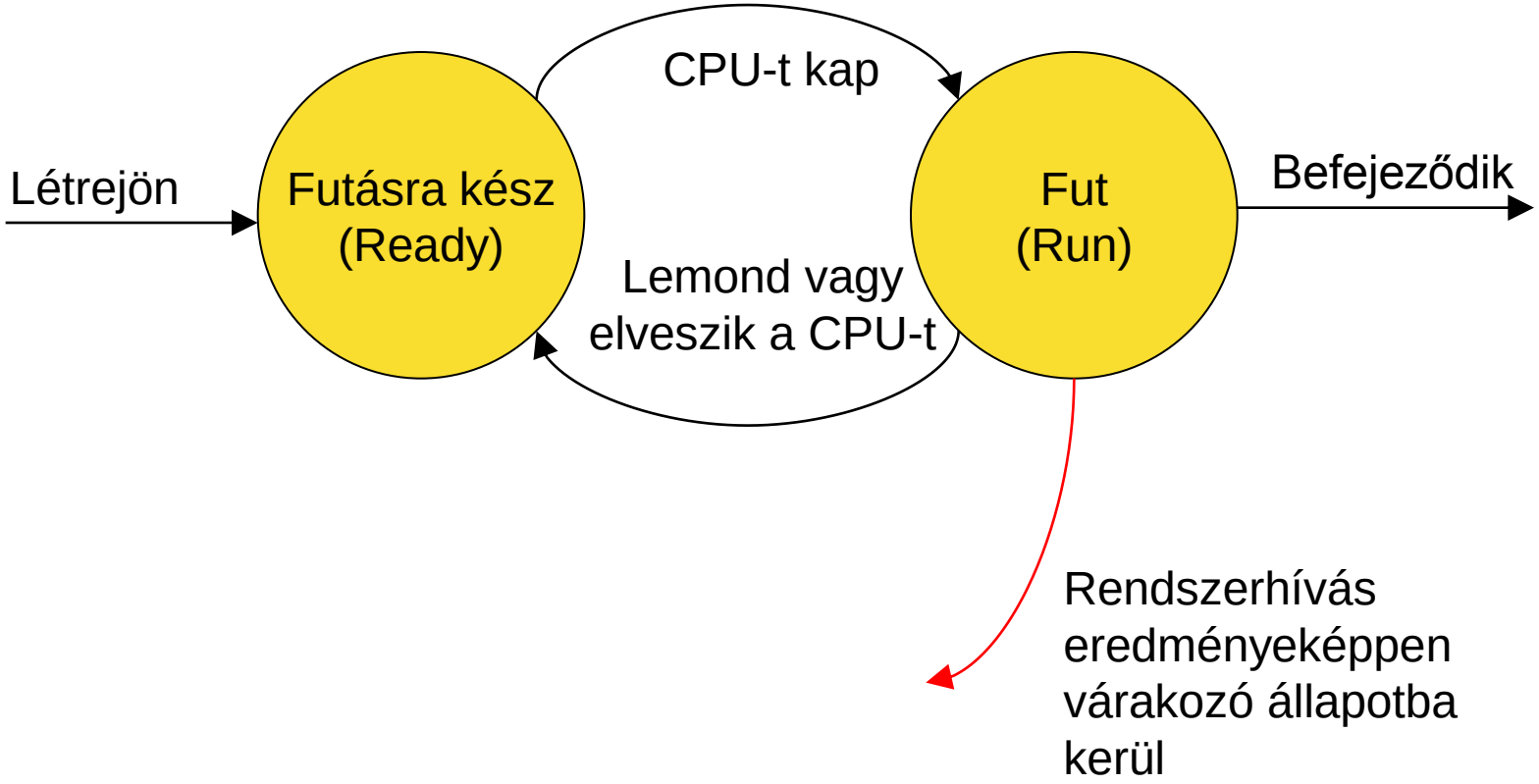
- A feladat lemond vagy elveszik tőle a CPU-t.
- Lemond a CPU-ról. Pl. yield() rendszerhívás.
- Elveszik a CPU-t. Preemptív CPU ütemezés, lesz róla szó...



Elveszik a CPU-t, miért, hogyan?
- A feladat fut, mi tudja elvenni a CPU-t?
- Az OS tudja csak elvenni → Futnia kell valahogy, de a folyamat fut...
- Megszakítás esetén tud futni az OS...
- Kell egy megszakítás (HW, SW, vagy kivétel)

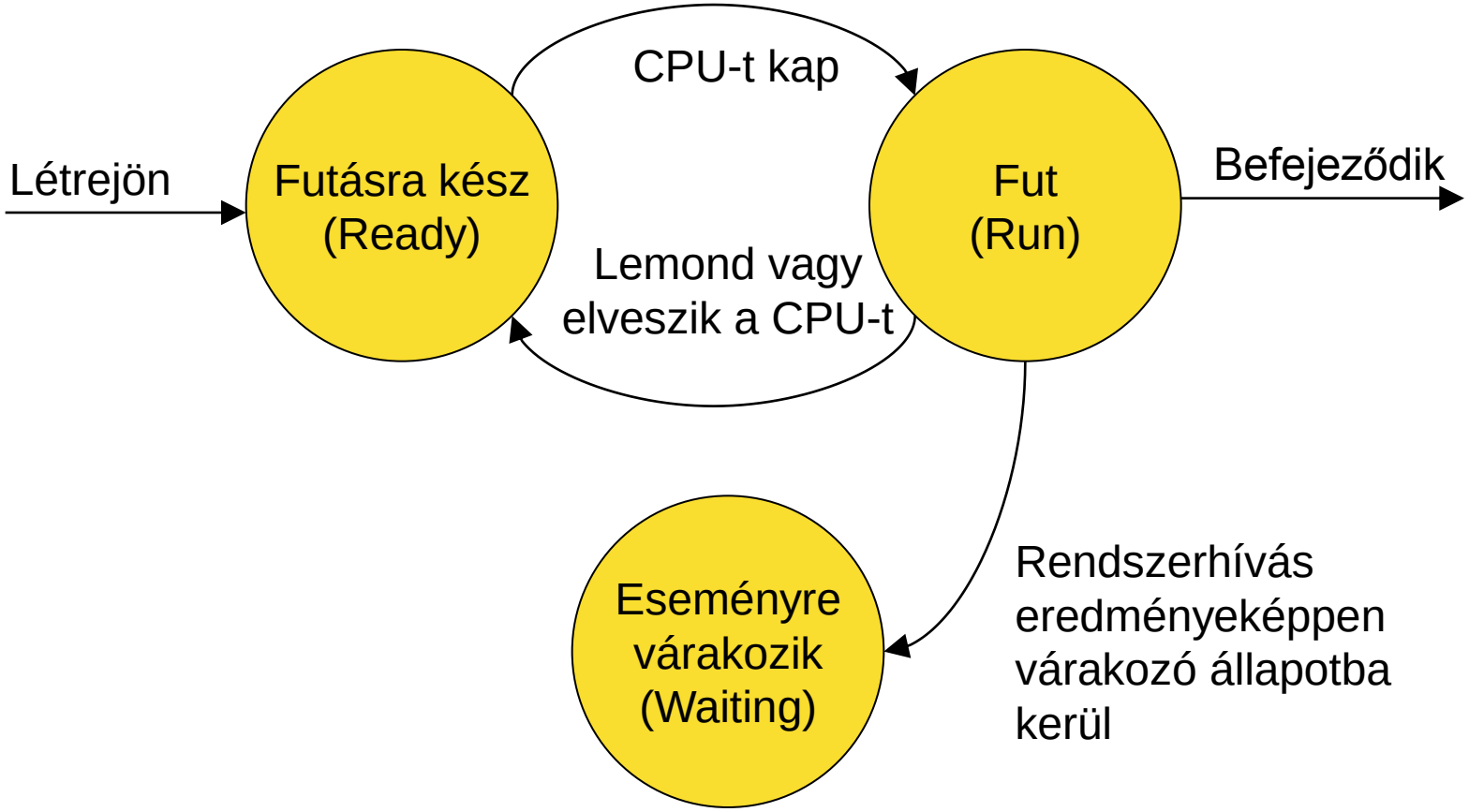
Feladatok állapota 6c.

- A feladat rendszerhívást hajt végre, de nem kapja vissza a CPU-t.
- Az OS úgy dönt, hogy a feladat által küldött kérés kiszolgálásához időre van szüksége, és addig a CPU-t más használja.



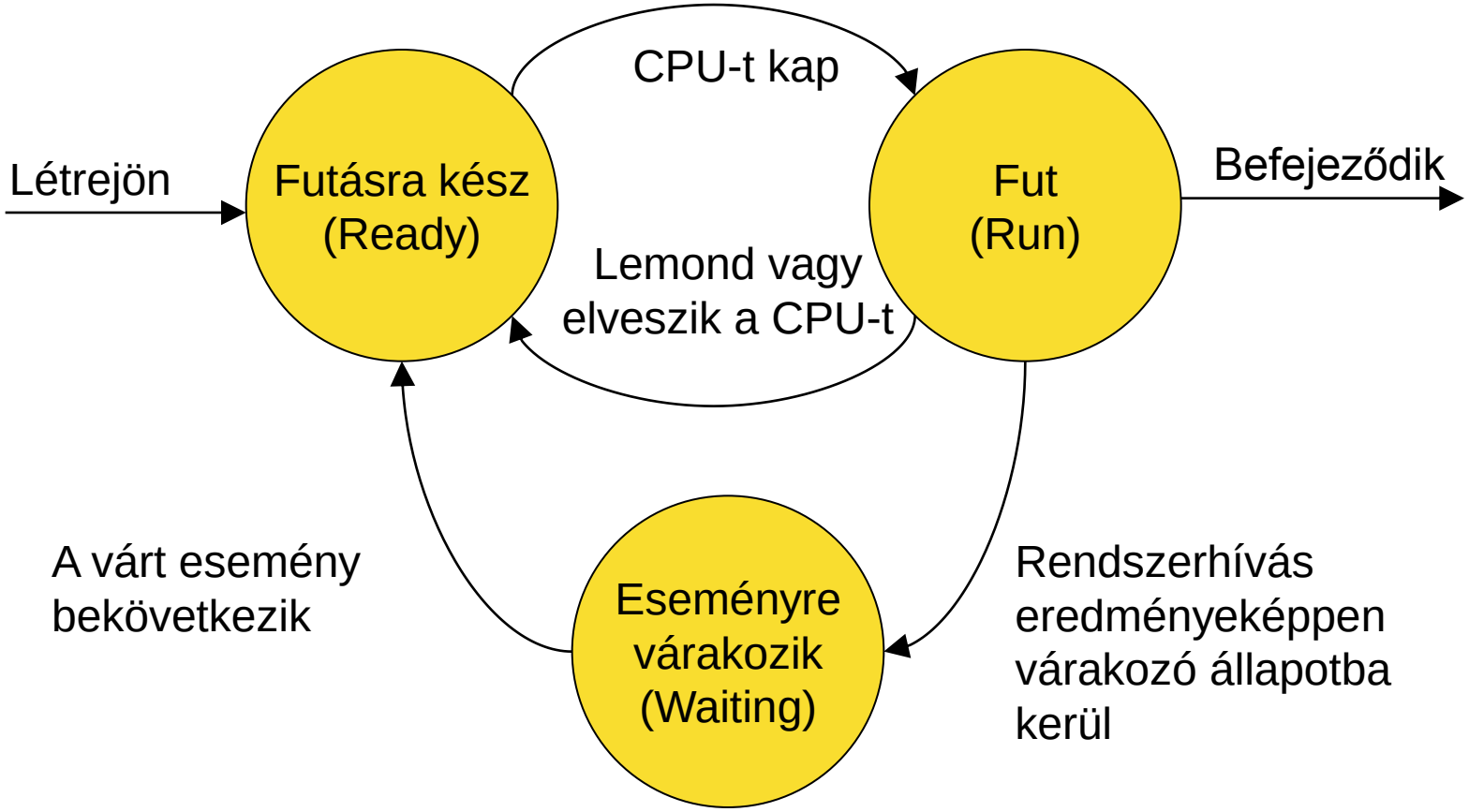
Feladatok állapota 7.

- A feladat „Eseményre várakozik” (Waiting) állapotban van.
- Az OS vagy más feladatok fogják azt előállítani. A feladat passzívan várakozik, nem használ CPU időt.



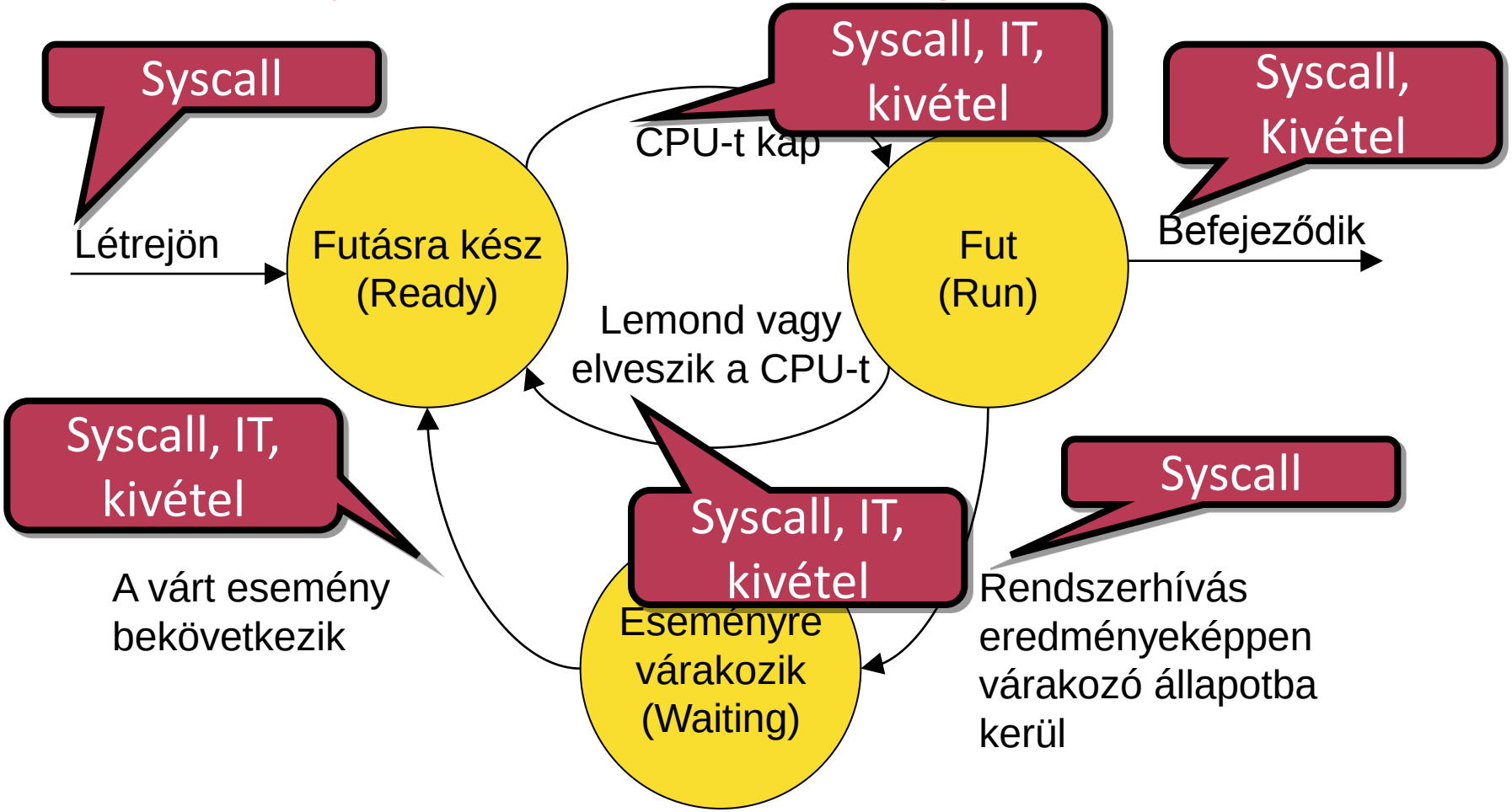
Feladat állapota 8.

- A várt esemény bekövetkezik.
- A feladat ismét futásra kész állapotba kerülhet, hiszen minden feltétel megvan a futásához, kivéve a CPU.



Feladat állapota 9.

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



Feladat leíró (Task control block, TCB)

- Adatstruktúra a feladattal kapcsolatos adatok ***operációs rendszeren*** belüli tárolására.
- Ez alapján végzi az OS a feladatok kezelését.
  - A feladat bizonyos részéhez hozzáférhet rendszerhívással (többnyire lekérdezheti).
- Mit tartalmaz?
  - Task ID (Process ID vagy Thread ID az implementációkban)
  - Állapot (Futásra kész, Fut, Eseményre vár, stb., OS specifikus)
  - A feladat legutolsó kontextusa:
    - Utasítás számláló (Program Counter, mentés/visszaállítás)
    - CPU regiszterek (Registers, mentés/visszaállítás)
    - CPU ütemezéssel kapcsolatos információk
    - Memória kezeléssel kapcsolatos információk (Virtuális memória)
      - » MMU állapot (mentés/visszaállítás)
  - Jogosultságokkal kapcsolatos információk (tulajdonos, ACL)
  - I/O státusz információ (használt I/O erőforrások, és azok állapota)

Kontextus váltás (context switch)

- Amikor egy feladat futni kezd helyre kell állítani a kontextusát, amiben korábban futott.
  - Utasítás számláló, egyéb CPU regiszterek, MMU állapot, erősen HW függő részletek...
- Ehhez el kell menteni az előtte futó feladat kontextusát.
- Ezt csak az OS tudja megtenni, neki is van kontextusa...
- Tiszta CPU idő veszteség (overhead)
  - Egyes CPU-k speciális utasításokat tartalmaznak erre
  - Egyes CPU-k többszörös regiszter készletet tartalmaznak
  - Hyperthreading (marketing név, semmi köze a thread-ekhez)
    - Látszólag több processzor, de valójában csak egy
    - Architektúrális állapot van többszörözve (regiszterek, stb.)
    - Ha van utasítás szintű párhuzamosság, akkor már akár párhuzamosan futhatnak (más jellegű aritmetikai műveleteket végeznek)

Feladatok ütemezése (task scheduling)

- Feladat: Kiválasztani a futásra kész feladatok közül a futót (visszatértünk a feladat megnevezéshez)
- A feladatokat a többnyire feladat sorokban (task queue vagy job queue) tároljuk
  - Ez a legegyszerűbb esetben tényleg egy FIFO, ami TCB típusú objektumokra/struktúrákra mutató pointereket tartalmaz.
  - Pl. futásra kész sor, adott eseményre vár sor, stb.
  - Az ütemezés sorbaállási modellje (Queuing diagram)
  - Az ütemező algoritmus ezeken a struktúrákon dolgozva végzi el a feladatát (elsősorban a futásra kész soron).
  - Keresésről van szó, általános esetben NP teljes a probléma.
  - Viszont kemény valós idejű a feladat még nem valós idejű operációs rendszerben is.
    - Az overhead-et egy minimális szinten kell tartani!

Ütemezés időskálái 1.

- Rövid távú (short-term) v. CPU ütemezés
  - A futásra kész sorból választ egy futó állapotba átmenő feladatot.
  - Minimum 10-20 ms-enként végrehajtásra kerül (időzítő megszakítás), de inkább gyakrabban.
- Hossz távú (long-term) ütemezés batch rendszerekben
  - Sokkal több feladatunk van, mint amennyit hatékonyan párhuzamosan végre tudunk hajtani.
  - Percenként vagy ritkábban fut, ismernie kell a feladatot (az általa okozott terhelést).
  - Többnyire maximum időzíteni lehet feladatokat (UNIX: cron)
  - Pl. 3-4 nagy file másolása Windows alatt egy diszkről egy másikra.
    - 1. lehetőség: Párhuzamos másolás.
    - 2. lehetőség: Szekvenciális másolás.
    - Kérdés a hallgatóknak: Melyik lesz gyorsabb és miért?

Ütemezés időskálái 2.

- Középtávú (medium-term) ütemezés
  - Swapping (Később részletesen fogunk vele foglalkozni)
    - A rendszerben lévő feladatok memóriájának egyes részei kiírhatóak háttértárra.
    - Több feladat fér el a fizikai memóriába (virtuálisan).
    - Egy feladatra több fizikai memória juthat (amikor fut).
  - Mi van, ha a futó vagy futásra kész (ami várhatóan hamarosan futni fog) feladat egyes szükséges részei a háttértáron vannak?
    - Nem tud futni, addig, amíg a szükséges memória részek vissza nem kerülnek a fizikai memóriába.
    - A középtávú ütemezés ezt irányítja...

További alapfogalmak

- Újabb alapfogalmakat és definíciókat kell bevezetnünk...

Preemptív ütemezés (preemptive)

- Tervezési kérdés, hogy a futó feladattól elvehető-e a CPU.
  - Preemptív: A futó feladattól az OS elveheti a CPU-t
    - Ez a tipikus a modern operációs rendszerekben.
    - Bizonyos kernel feladatokra nem feltétlenül, azok nem megszakíthatóak, ennek vannak következményei (pl. valós idejű működés nehezen biztosítható)
  - Nem preemptív vagy kooperatív
    - Pl. Windows 3.x
    - A futó feladatnak le kell mondania a CPU-ról vagy eseményre kell várnia, ahhoz hogy más feladat tudjon futni.
    - Az egyes alkalmazásoktól függ a teljes rendszer működése.
      - Lényegében az alkalmazás programozójától, ami nem jó ötlet…
    - Ha a futó feladat hibás (végtelen ciklus), akkor a teljes rendszer működésképtelenné válik.
      - OS az ilyen OS egyáltalán? ☺

Mértékek 1.

- Mérték (metric)
  - Ezekkel tudjuk az algoritmusokat összehasonlítani.
  - Többnyire több mértéket együtt tekintve kell választanunk.
  - Mindegyiknek van mértékegysége (unit) is!
- CPU kihasználtság (CPU utilization)
  - Mértékegység: %
  - A hasznos munkával töltött idő aránya az összes időhöz képest.
  - $t_{CPU}=t_{CPU,munka}+t_{CPU,admin}+t_{CPU,idle}$, vagyis az adminisztráció és henyélés veszteség
  - Jellegzetesen egy 40-90 % körüli érték tekinthető jónak.

$$\frac{\sum t_{CPU,munka}}{\sum t_{CPU}} * 100[\%]$$

Mértékek 2.

Átbocsátó képesség (throughput)

- Mértékegység: munka/s, vagy 1/s
- Adott időegység alatt elvégzett feladatok száma.
- A rendszerfeladatokat nem számoljuk.
  - Lényegében csökkentik az átbocsátó képességet, elveszik a CPU időt.
- A tipikus érték erősen függ a feladat jellegétől
  - Standard benchmarkokkal mérik.
  - Pl. TPC-X benchmarkok a Transaction Processing Performance Council-tól (Lásd: http://www.tpc.org/)

$$\frac{\text{Elvégzett munkák száma}}{\text{Idő}} [1/s]$$

Mértékek 3.

Várakozási idő (waiting time)

- Mértékegység: s
- Az összes idő, amit a feladat várakozással töltött.
- Igazán ez függ az ütemező algoritmustól...
- Statisztikai ingadozás a munka jellegéből és a végrehajtás részleteiből következően:
  - Ennek megfelelően átlagos várakozási időről, annak a szórásáról, stb. beszélhetünk inkább.

$$t_{waiting} = t_{ready} + t_{other,non-running} [s]$$

Mértékek 4.

Körbefordulási idő (turnaround time)

- Mértékegység: s
- Egy feladatra vonatkozóan a rendszerbe helyezéstől a teljesítésig eltelt idő.
- Statisztikai ingadozás a munka jellegéből és a végrehajtás részleteiből következően:
  - Ennek megfelelően átlagos körbefordulási időről, annak a szórásáról, stb. beszélhetünk inkább.
  - Ez fontos a felhasználó számára, hiszen az minél hamarabb szeretné látni a teljes eredményt.
  - A szabványos benchmarkok itt is fontos szerepet játszanak.

$$t_{CPU,végrehajtás} + t_{várakozás}$$

Mértékek 5.

Válaszidő (response time)

- Mértékegység: s
- On-line, interaktív feladatok esetén.
- A feladat megkezdésétől az első kimenetek produkálásáig eltelt idő.
- Statisztikai ingadozás a feladat jellegéből és a végrehajtás részleteiből következően:
  - Ennek megfelelően átlagos válaszidőről, annak a szórásáról, stb. beszélhetünk inkább.
  - Ez fontos a felhasználó számára, hiszen az minél hamarabb szeretné látni az első eredményeket (el tud kezdeni dolgozni, nem henyél tovább).
  - A szabványos benchmarkok itt is fontos szerepet játszanak.

Mértékek 6.

Felhasznált energia jellegű mértékek (energy metrics)

- Mértékegység: pl. Ws/task (pl. TPC-Energy)
  - Mennyi energia szükséges egy standard feladat (benchmark) elvégzéséhez?
- Az energia fogyasztás egyre jobban a középpontba kerül.
- Erősen átlapolódik a többi mértékkel.
  - Pl. Számítási teljesítmény, ár és energia fogyasztás?
  - Pl. Intel ATOM (vagy akár egy ARM) összehasonlítása egy C2D-vel? Ki a jobb?
- Statisztikai ingadozás a feladat jellegéből és a végrehajtás részleteiből következően:
  - Ennek megfelelően átlagos energiafogyasztásról, annak a szórásáról, stb. beszélhetünk inkább.
  - Energiatudatos ütemezés, benchmarkok kidolgozás alatt.

Követelmények

- Az ütemező algoritmus valós idejű működése
  - Alacsony overhead…
- Célfüggvény szerint legyen optimális
  - A célfüggvény a különböző mértékekből származtat egy összehasonlításra használható számot.
    - Pl. mértékek súlyozott lineáris kombinációja.
- Matematikai modell, szimuláció vagy mérések alapján vizsgálódnak.
    - Reprodukálható, jellemző terhelés (benchmark)?
    - Statisztikai ingadozás a rendszer működési bizonytalanságai miatt (spekulatív végrehajtás, utasítás átrendezés, cache, stb.).

Kvalitatív jellemzők

- Elvárt tulajdonságok (nem vagy nehezen számszerűsíthetők):
  - Korrektség (fairness)
  - Kiéheztetés elkerülése (no starvation)
  - Jósolható viselkedés (predictability, determinism)
  - Alacsony adminisztratív veszteségek (low overhead)
  - Maximális átbocsátó képesség, minimális várakozási idők
  - Erőforrás használat figyelembe vétele
    - Népszerű erőforrásokat használók futtatása (haladjon a rendszer)
    - Ritkán használt erőforrásokat használók előtérbe helyezése (nem fog feltartani senkit)

Követelmények 2.

- Egyéb tulajdonságok
  - Valós idejű ütemezés (hard or soft real-time)
    - Lehessen feladatokat garantált körbefordulási idővel ütemezni, ha azok végrehajtási idejére adható felső korlát.
- Prioritás
  - A feladatokhoz fontosságot (prioritást) rendelünk
  - Prioritás ≠ Lágy valós idejű (soft real-time) (tipikus hiba)
  - Részletesen fogunk vele foglalkozni!
- Fokozatos leromlás/összeomlás? (Graceful degradation)
  - Ha a rendszer terhelése eléri az u.n. könyökkapacitást, akkor utána viselkedése megváltozik, a tovább növekvő terhelésre már egyre rosszabb működéssel reagál (overhead).
  - Elvárható, hogy ezt fokozatosan tegye (ne omoljon össze hirtelen)

Egyéb szempontok

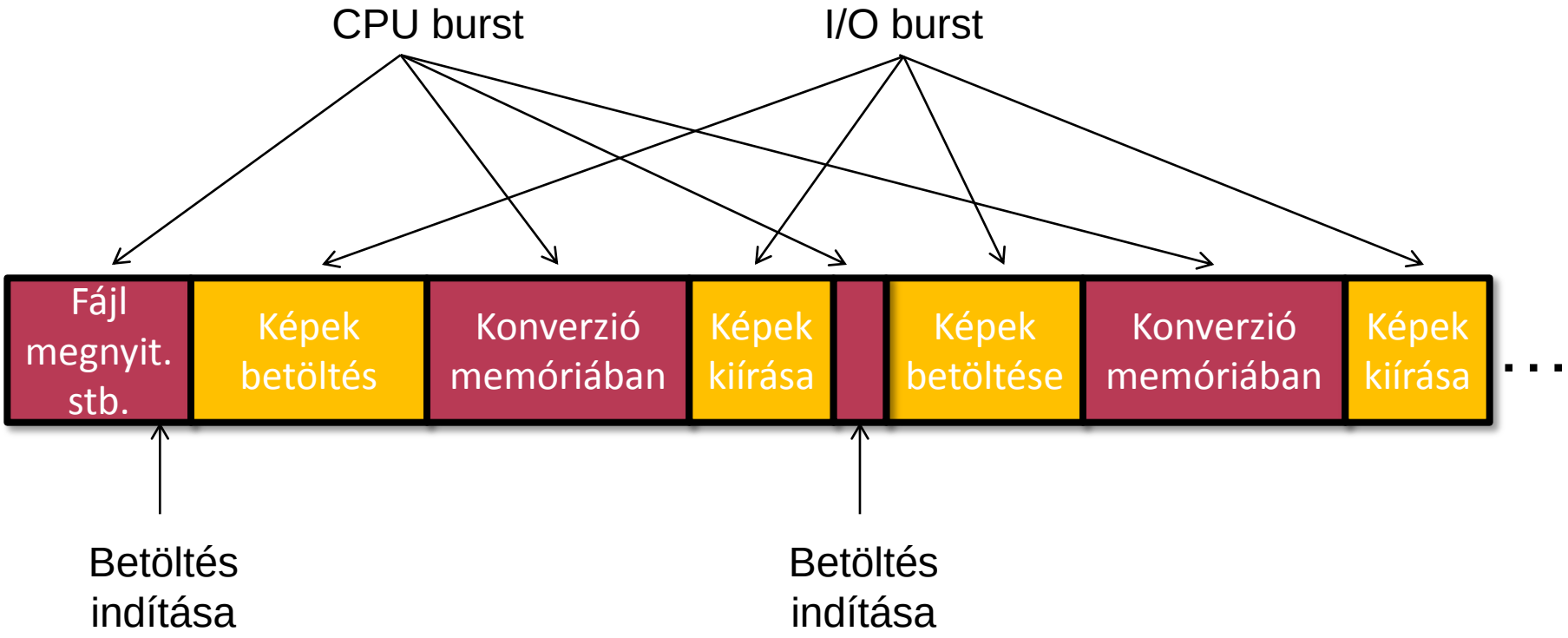
- Statikus v. dinamikus ütemezési algoritmusok
  - Statikus:
    - Tervezési időben teljesen meghatározott, hogy milyen feladatok és mikor futnak.
    - Legrosszabb esetre tervezés.
    - Nem foglalkozunk vele, nagyon speciális, többnyire biztonságkritikus beágyazott rendszerekben alkalmazzák.
  - Dinamikus: Futási időben dől melyik feladat és mikor fut (Ezekkel foglalkozunk).
    - A gyakorlatban használt algoritmusok ilyenek.
    - Dinamikus erőforrás kihasználás.
    - Tervezési időben nehezen vizsgálhatók.

CPU ütemezési algoritmusok (1 CPU)

- Feltételezzük az alábbiakat:
  - A rendszerben egy végrehajtó egység van.
    - Több processzor esetén a probléma sokkal összetettebb.
  - Egy időben egy feladat tud futni.
  - A futásra kész feladatokat valamilyen várakozási sorban tároljuk (task/job queue)
    - Ez speciális esetben lehet FIFO, de általában nem az...
  - A feladataink CPU és I/O löketekből (burst) állnak
    - Ez tapasztalat, mérések is vannak.
    - Pl. a CPU löket jellegzetesen kisebb 10 ms-nél a mérések szerint.
    - Ezek között I/O löketek vannak, ekkor a feladat passzívan vár az I/O lezajlására.

CPU és I/O löket példa

- Video fájl konverziója egy processzoron kevés rendelkezésre álló memóriával…



Legrégebben várakozó (FIFO, FCFS)

- A legegyszerűbb algoritmus:
  - Feladat leíróra mutató referenciákat tároló FIFO (First Input First Output) sor az implementáció.
    - put()-tal bekerül a folyamat a várakozó sorba.
    - get()-tel levesszük a futásra kerülőt.
    - FCFS (First Come First Serve)
- Nem preemptív definíciószerűen
  - I/O-ra várhat!
- Az átlagos várakozási idő nagy lehet, és erősen függ a CPU és I/O löket (burst) nagyságoktól.
  - Ez egyben átlagosan nagy válaszidőt is jelent, on-line felhasználókat is kiszolgáló rendszerek számára nem alkalmas.
- Kis adminisztrációs overhead.

FIFO algoritmus tulajdonságai

- Várakozási idő kicsit részletesebb vizsgálata:
- Tiszta CPU használat:
  - A később beérkező feladatoknak ki kell várniuk a korábban beérkező feladatokat.
  - Hosszú feladat sokáig feltartja az utána következőket.
- CPU és I/O löketek:
  - Ha van olyan feladat, aminek hosszú a CPU lökete, akkor az fel fogja tartani a kis CPU lökettel rendelkező feladatokat.
    - A kis CPU lökettel rendelkező feladatok a gyakori I/O löket miatt gyorsan a sor végére kerülnek.
    - A nagy CPU lökettel rendelkező még a sor végéről is hamar előre kerül, aztán feltartja a többit.
    - Convoy hatás...

Körforgó (Round-robin)

- Időosztásos rendszerek számára találták ki az egyszerű FIFO ütemezés problémáinak kijavítására.
- Kedvezőbb az on-line felhasználók számára.
  - Jobb az átlagos válaszideje a FIFO-nál.
  - Adott időnként garantáltan vált, függetlenül a feladattól.
- Preemptív:
  - Lényegében a FIFO kiegészítése egy egyszeri óra megszakítással.
    - Quantum vagy time slice (időszelet).
    - 100-1 ms, tipikusan 10-20 ms.
    - Elsősorban elfogadható on-line válaszidő biztosítására.

Egyszeri óra megszakítás

- A feladat futtatása előtt az órát elindítjuk, az egy adott idő (időszelet) lejárta után megszakítást kér (fut az OS)
  - Ha a feladat befejeződik vagy I/O-t hajt végre az óra megszakítás előtt (fut az ütemező):
    - Újraütemezünk, és az órát újraindítjuk.
  - Ha a feladat nem fejeződik be:
    - A megszakítás beérkezik, és fut az ütemező.
    - A feladat futásra kész állapotba kerül, újraütemezünk és az órát újraindítjuk.
- A gyakorlatban sok esetben az egyszerűség kedvéért periodikus óra megszakítást alkalmaznak.
  - Így egyszerűbb az algoritmus, de a matematikai analízis bonyolultabb, és a tulajdonságok is romolhatnak.
  - A periodikus óra IT használható a rendszeróra és SW időzítők kezelésére is.

RR algoritmus tulajdonságai

- Tulajdonságai az időszelet nagyságától, CPU löketek eloszlásának statisztikai jellemzőitől, és a kontextus váltás időtartamának a viszonyától függ.
- Időszelet > átlagos CPU löket → Átmegy FIFO-ba.
  - Lényegében a CPU löket nagyságától függően az I/O műveletek eredményeképpen fut az ütemező.
- Időszelet ≈ átlagos CPU löket → Normális működés.
  - A feladatok időosztásos módon osztoznak a CPU-n.
- Ökölszabály: A CPU löketek 80%-a kisebb az időszeletnél.
- CPU löket > Időszelet, ami viszont összemérhető a kontextus váltás időtartamával:
  - Nagy adminisztratív terhelés.
  - Mivel 10-20 ms (1 ms) az időszelet, ami nagyságrendekkel nagyobb a kontextus váltás idejénél, ezért ez ma nem probléma (régen volt).

Prioritásos ütemezők

- Ütemező család...
- Prioritás = fontosság (0 a legkisebb v. a legnagyobb?)
- A feladatokhoz prioritást rendelünk:
  - Külső/Belső prioritás:
    - Külső prioritás: Operátor vagy a feladat maga állítja be.
    - Belső prioritás: A rendszer adja.
  - Statikus/Dinamikus prioritás:
    - Statikus prioritás: Tervezési időben dől el, állandó a futás során.
    - Dinamikus prioritás: Futási időben dől el, változik a rendszerben lezajló változások hatására.
  - A gyakorlatban lehet ezek kombinációja is...

Prioritás megállapítása

- Prioritás belső megadása mérhető mértékek alapján:
  - Felhasználó adhat egy kiinduló prioritást is
  - Az operációs rendszernek kell kiszámolnia, pl. az alábbiak alapján:
    - Időkorlátok (válaszidő, stb.),
    - Memória igények (méret, elérhetőség, stb.),
    - Használt erőforrások és azok száma,
    - CPU és I/O löket,
    - Futási idő, tervezett futási idő,
    - stb.

Prioritásos ütemezők tulajdonságai

- Általános tulajdonságok:
- Jellegzetesen preemptív, de lehet nem preemptív is.
- Nem korrekt (fair), nem is akarjuk, hogy az legyen...
- Kiéheztetés előfordulhat (Előny vagy hátrány?):
  - Nem pontosan a fontosság figyelembe vétele miatt használjuk?
  - Ha előfordul, akkor az alapvetően vagy túlterhelés miatt, vagy tervezési hiba miatt történik.
  - A magas prioritású feladatok egy időben a feladat készlet egy kis, biztosan futtatható részét tehetik ki.
    - Hagynak CPU-t az alacsony prioritású feladatoknak is.
  - Nem lehet minden feladat egyformán fontos!
  - A feladatok öregítése segíthet (régóta vár, egyre fontosabb).

Prioritásos ütemezők 1.

- Egyszerű prioritásos ütemező:
  - Egy prioritási szinten egy feladat.
  - Egyszerű beágyazott rendszerekben használják.
- Egy prioritási szinten tetszőleges számú feladat:
  - Módosításokkal széles körben alkalmazásra kerül.
    - UNIX, Windows, stb., szinte minden operációs rendszerben ilyet találunk.
  - Módosítások:
    - Egy szinten belül a futásra kész feladat kiválasztására használt algoritmust meg kell adni, tipikusan RR.
    - Dinamikus, belső prioritás meghatározás.
- Alapértelmezett prioritásos ütemező súlyos hibája:
  - Prioritás inverzió (priority inversion), később tárgyaljuk.

Prioritásos ütemezők 2.

- A legrövidebb löketidejű (Shortest Job First, SJF):
  - Nem preemptív, a legrövidebb (becsült) löketidejűt választja futásra.
    - Optimális az átlagos várakozási és körülfordulási idő szempontjából (ha a löketidők ismertek, de nem azok).
    - Hogyan becsüljük a löketidőt?
    - Korábbi löketidők (exponenciális/felejtő átlag) vagy a felhasználó által megadott értékek alapján.
    - A felhasználók ismerik az ütemező algoritmust! ☺

Prioritásos ütemezők 3.

- A legrövidebb hátralévő idő (Shortest Remaining Time First, SRTF):
  - Az SJF preemptív változata.
  - Ha új folyamat válik futásra késszé, akkor vizsgálja meg, hogy melyik folyamat löketideje a kisebb.
  - A legrövidebbet indítja el.
  - A kontextus váltást is figyelembe kell venni (megéri-e a váltás).
  - Hogyan becsüljük a löketidőt?
    - Korábbi löketidők vagy a felhasználó által megadott értékek alapján.
    - Ugyanazok a problémák, mint SJF-nél...

Prioritásos ütemezők 4.

- A legjobb válaszarány (Highest Response Ratio, HRR):
  - A kiéheztetést próbálja megoldani.
  - Az SJF-ből indul ki.
  - A prioritás képzésébe a várakozási idő is beleszámít.
  - Minél többet vár, annál valószínűbb, hogy ütemezve lesz...
  - A k-t meg kell választani, mennyire vegyük figyelembe a feladat korát (várakozási időt)

$$\frac{\text{Löketidő} + k * \text{Várakozási idő}}{\text{Löketidő}}$$