Operációs rendszerek

Folyamatok ütemezése a UNIX-ban

Folyamatok ütemezése a UNIX-ban

- Ütemező algoritmus megválasztása meghatározza:
  - a rendszer teljesítményét,
  - a hardver kihasználtságát.
- Tradicionális UNIX ütemezési algoritmust használt:
  - a System V. R3, illetve a 3.1 BSD UNIX,
  - de néhány megoldása már idejétmúlt.
- Ezért a mai UNIX rendszerek:
  - az algoritmus valamilyen továbbfejlesztett változatát használják.

Az ütemezési algoritmussal szemben támasztott követelmények

- A UNIX rendszer feltételezett felhasználása:
  - több felhasználós,
  - interaktív programok,
  - batch programok.
- Követelmények:
  - alacsony válaszidő (interaktív folyamatok),
  - nagy átbocsátó képesség,
  - éhezés elkerülése (háttérben futó alacsony prioritású pl. batch folyamatok),
  - a rendszer terhelésének figyelembe vétele,
  - a folyamatok futási esélyeinek a befolyásolása.

A UNIX-ütemezés rövid jellemzése

- Prioritásos ütemezés.
- Felhasználói, illetve kernel módban eltérő algoritmus:
  - **felhasználói** mód:
    preemptív ütemezés, időosztásos, időben változó prioritás, körforgásos FCFS,
  - **kernel** mód:
    nem preemptív ütemezés, fix prioritású folyamatok.

Ütemezés megvalósítása

- Ütemezéshez kapcsolódó tevékenységek (pl. prioritási szám számítása, legnagyobb prioritású folyamat kiválasztása) önálló függvényekben vannak megvalósítva.
- Óra megszakításhoz kötődik az ütemezési rutinok végrehajtása:
  - ütemezési ciklusonként (egy vagy néhány óra megszakításonként).
- Mechanizmus:
  - *call-out* függvényekkel.

Felhasználói módú ütemezés

Felhasználói módban az ütemezés *preemptív.*

- Újraütemezés:
  - ha egy magasabb prioritású folyamat futásra késszé válik,
  - ha több azonos prioritású folyamat van futásra készen és nincs náluk magasabb prioritású futásra kész folyamat: Round-Robin-os FCFS.

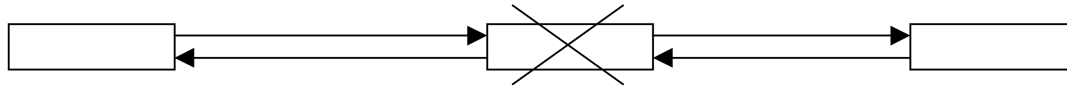
Kernel módú ütemezés

A kernel módban az ütemezés szigorúan nem *preemptív*.

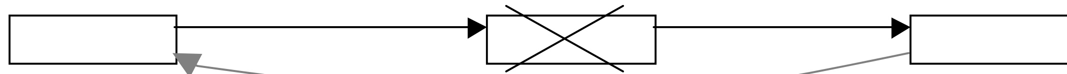
- Kernel kódot végrehajtó folyamatot nem lehet kényszeríteni, hogy lemondjon a CPU használatról egy nagyobb prioritású folyamat javára.
- Átütemezés kernel módban:
  - önként lemond a futás jogáról (*sleep*),
  - folyamat visszatér kernel módból felhasználói módba.

- A kernel kód korlátozottan reentrens:
  - több példányos futás lehetősége,
  - *sleep*, de egy másik folyamat újraindíthatja!

Példa ami indokolja, hogy a kernel nem preemptív



Láncolt lista egy közbülső elem lefűzése előtt



Láncolt lista egy közbülső elem lefűzése közben

Folyamatok ütemezési prioritása

0	legnagyobb prioritás	KERNEL prioritások
.		
.		
.		
49		
50		FELHASZNÁLÓI prioritások
.		
.		
.		
127	legkisebb prioritás	

A fix prioritás meghatározása kernel módban

- Kernel módú prioritás meghatározása:
  - a rendszer milyen ok miatt hajtott végre *sleep* rendszerhívást, azaz
  - **milyen eseményre várakozik,**
  - ezért a kernel prioritást szokták **alvási prioritásnak** is nevezni.

Kernel futás közben nem fontos a prioritás.

A folyamat kernel módba kerülése

- Rendszerhívással:
  - *user* futás után, ahol a prioritás adott és mentett,
  - a *kernel* módú még ismeretlen, de nem is kell.
- Külső megszakítással:
  - nincs ütemezés,
  - a megszakítási rutin a folyamat környezetében hajtódik végre (mintha a folyamat futna tovább),
  - a megszakítási rutin után:
    - *kernel* futás, ha a megszakítás előtt is így futott (nincs prioritás),
    - *user* futás, az áttérés előtti mentett prioritás fog élni.

A változó prioritást meghatározó tényezők felhasználói módban

- Két fő tényező:
  - *nice* szám (kedvezési szám):
    - a felhasználó beavatkozása a prioritás számításába,
  - korábbi CPU használat:
    - öregítés,
    - egyenletes CPU használat.

Prioritás számításához használt paraméterek

- A prioritás számításához négy paramétert használ a UNIX:
  - a **p_pri**-t: az aktuális ütemezési prioritást,
  - a **p_usrpri**-t: a felhasználói módban érvényes prioritást,
  - a **p_cpu**-t: a CPU használatra vonatkozó számot,
  - a **p_nice**-t:**:** a felhasználó által adott kedvezési számot.
- A paramétereket minden folyamat esetén külön-külön számon tartja a UNIX.

Prioritás számításához használt konstansok

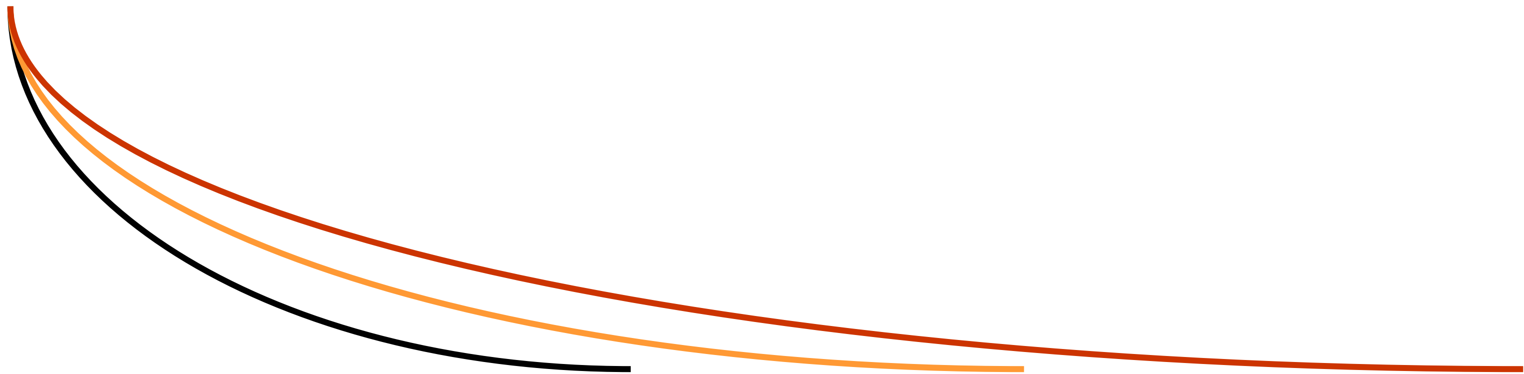
- P_USER konstans:
  - a legmagasabb *user* módú prioritás azaz: 50.
- Korrekciós faktor (KF):
  - a rendszer terheltségének a figyelembevétele,
  - sok folyamat: lassú öregítés,
  - kevés folyamat: gyors öregítés,
  - a várakozó folyamatok számának a növekedésével, az értéke egyre inkább tart az 1-hez.

Prioritás számítása felhasználói módban

	futó folyamat	**minden folyamat**
minden óra-interrupt	Induláskor: 0. p_cpu := p_cpu +1	Megvizsgálja, van-e a futónál magasabb prioritású folyamat. Ha van, átütemez.
minden 10. óra-interrupt	Round-Robin algoritmus: ha több azonos prioritású folyamat van a legmagasabb prioritású pozícióban, 10 óra-interrupt-onként váltja a futó folyamatot, FCFS módon.	
minden 100. óra-interrupt		1. KF=2*futásra kész folyamatok/(2* futásra kész folyamatok+1), 2. p_cpu = p_cpu * KF, 3. p_usrpri = P_USER+p_cpu/4+2*p_nice.

Korrekciós faktor

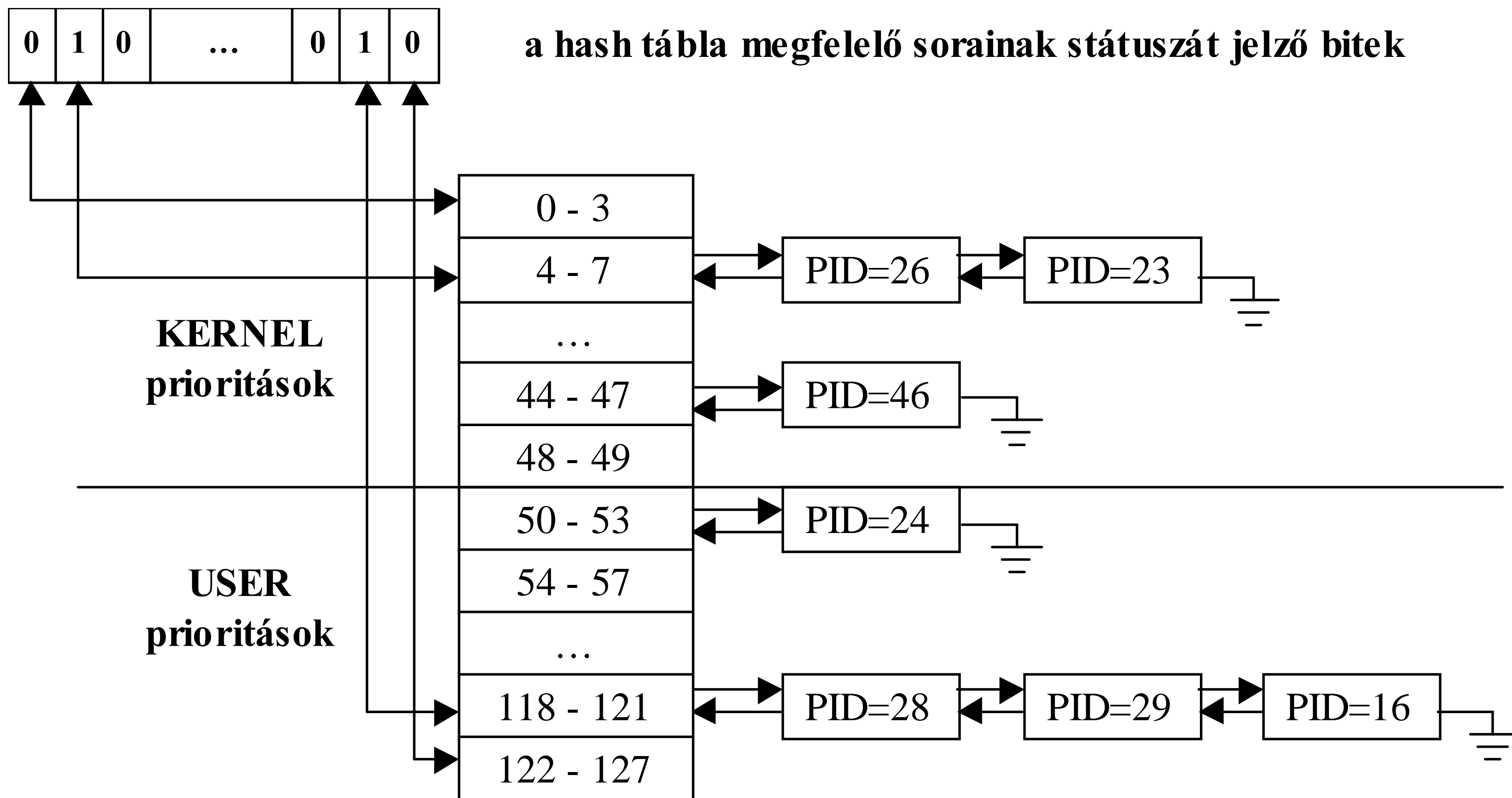
- KF=várakozó folyamatok száma/ várakozó folyamatok száma+1.
- Számos rendszerben konstans az értéke. Pl.: 1, ½, …
- ½, 2/3, 3/4, …



Környezetváltás ütemezéskor

- Nem *preemptív* ütemezés esetén:
  - Várakozik: *sleep* rendszerhívást hajt végre.
  - Befejeződik: *exit* rendszerhívást hajt végre.
- *Preemptív* ütemezés esetén:
  - a 100-adik óraciklusban a prioritások újraszámításakor, ha az egyik folyamat prioritása nagyobb lesz a futóénál,
  - a 10-edik óraciklus esetén a Round-Robin algoritmus (azonos prioritás) alkalmazásakor,
  - felébred (ready to run állapotba jut) egy, az aktuálisan futónál magasabb prioritású folyamat, két óraciklus között.

Adatszerkezetek folyamatok prioritásának tárolása



Példa az ütemezés számolására I.

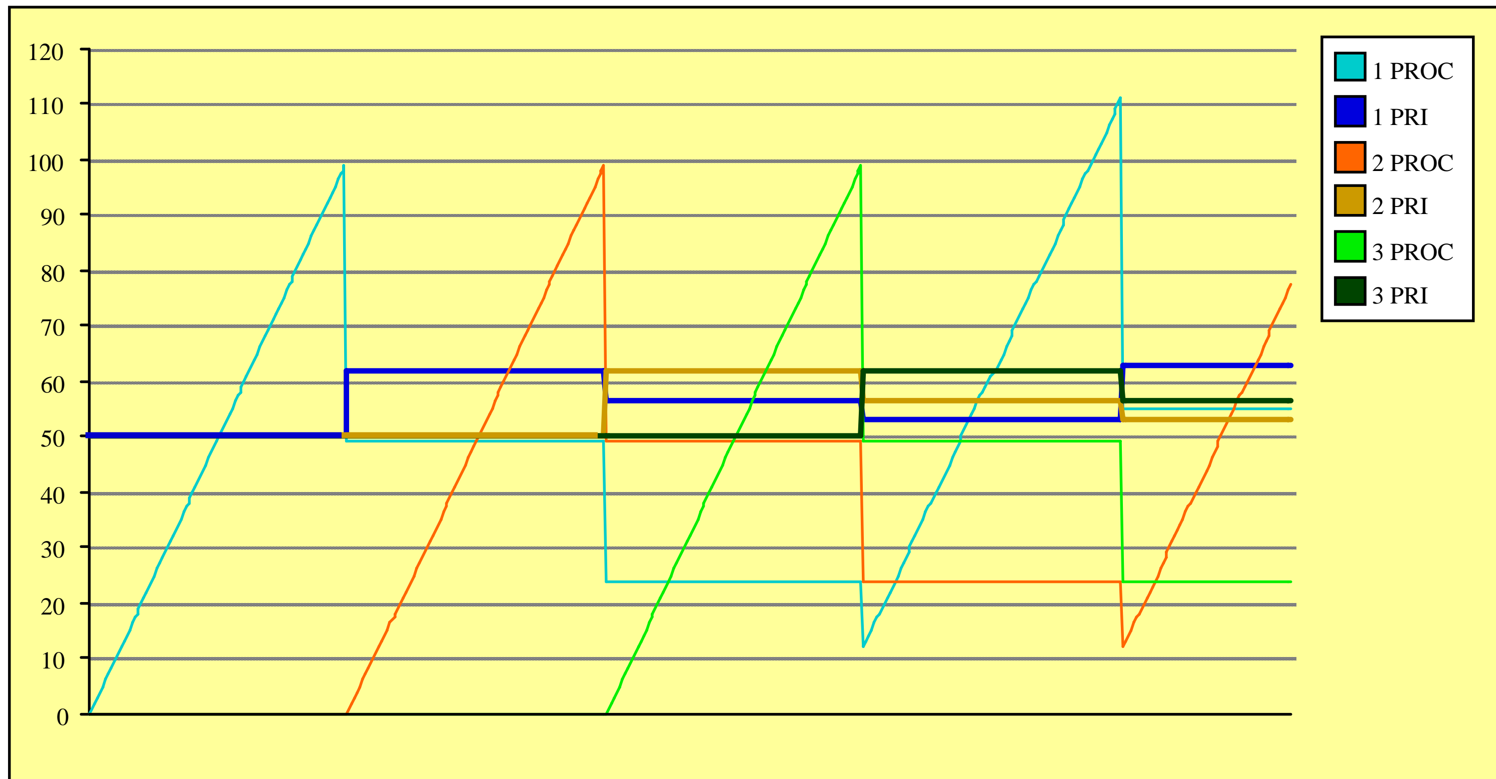
- Minden óramegszakításnál:
  p_cpu := p_cpu +1.
- Minden 100-adik óra-interrupt-nál:
  p_cpu = p_cpu * KF = p_cpu * ½.
  p_usrpri = P_USER+p_cpu/4+2*p_nice.

- P_USER=50, p_nice=0

UNIX ütemezési példa

A		B		C			
p_pri	*p_cpu*	*p_pri*	*p_cpu*	*p_pri*	*p_cpu*	*Lépés*	*Futó foly.*
50	*0*	50	0	50	0	1	A
50	*1*	50	0	50	0	2	**A**
…	...	…	…	…	…	…	**A**
50	*99*	50	0	50	0	99	**A**
50+50/4 **63**	***100/2*** ***50***	**50**	**0**	**50**	**0**	**100**	A
63	50	50	*1*	50	0	101	**B**
…	…	….	...	…	…	…	**B**
63	50	50	*99*	50	0	199	**B**
50+25/4 **56**	**50/2** **25**	**50+50/4** **63**	***100/2*** ***50***	**50**	***0***	**200**	**B**
56	25	63	50	50	*1*	201	**C**
…	…	….	…	…	...	…	**C**
56	25	63	50	50	*100*	299	**C**
50+13/4 **53**	**25/2** **13**	**50+25/4** **56**	**50/2** **25**	**50+50/4** **63**	***100/2*** ***50***	**300**	**C**

Példa az ütemezés számolására II.



A UNIX-ütemezés értékelése

- Nem méretezhető megfelelően a terhelés függvényében:
  - a korrekciós faktor használata nem elég hatékony.
- Az algoritmussal nem lehet meghatározott CPU időt allokálni.
- Nem lehet a folyamatok fix válaszidejét garantálni.
- Ha egy *kernel* rutin sokáig fut, az feltartja az egész rendszert.
- A felhasználó a folyamatainak prioritását nem tudja megfelelő módon befolyásolni.

Call-out függvények

Call-out függvények I.

- A *call-out* mechanizmus egy adott tevékenység későbbi időpontban történő végrehajtása.
- Ezen függvények adott időben történő meghívását a *timeout*(hívott függvény) rendszerhívással lehet megtenni, illetve az *untimeout*(hívott függvény) rendszerhívással lehet törölni.
- Rendszerkörnyezetben futnak, így:
  - nem alhatnak,
  - nem érhetik el a folyamatok környezetét.
- Ismétlődő feladatok (kernel funkciók) végrehajtására valók. Pl.:
  - hálózati csomagok ismételt elküldése,
  - ütemezési és memóriakezelő függvények hívása,
  - perifériák monitorozására,
  - perifériák lekérdezésére, amelyek nem támogatják a megszakításokat.

Call-out függvények II.

- Meghívásukat az óra-megszakítás végzi, ám ez alatt az egyéb megszakítások tiltottak.
- Ezért a megszakítás-kezelő nem közvetlenül hívja meg a *call-out* függvényeket, hanem beállít egy erre a célra rendszeresített jelző-bit-et, ha eljött az idő valamelyik függvény meghívására.
- A *kernel* ezt mindig ellenőrzi, miután kiszolgálta a megszakításokat és visszatér a megszakított tevékenységhez.
- Tehát a *call-out* függvények csak a kiszolgált megszakítások után kerülhetnek meghívásra, az óra-megszakítás után és nem alatta!

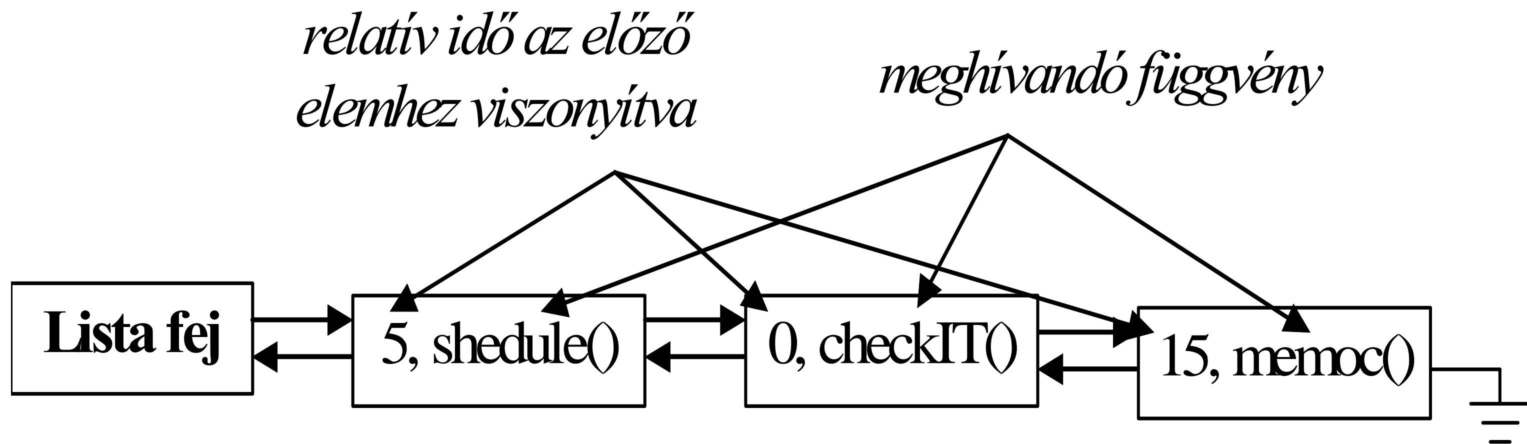
Call-out függvények II.

- A meghívás adminisztrálására dinamikus adatszerkezeteket kell használni. Ilyen a:
  - láncolt listás,
  - időkerekes.

Call-out függvények láncolt listás ábrázolása

- Meghívási sorrendbe rendezett lista.
- Minden listaelem első paramétere a meghívásig hátralévő időt tárolja tikk-ben, az előző listaelemhez képest.
- Az első listaelem tikk-je minden óramegszakítás esetén csökken eggyel. Nulla esetén a második paraméterben adott függvény aktivizálódik, illetve az utána álló összes, 0-ás időpont-számlálójú elemben meghatározott függvény.
- Egyszerűen kezelhető, de ha hosszú a lista, a beszúrás időigényes lehet.

Call-out függvények láncolt listás ábrázolása



Call-out függvények tárolása időkerékkel

- Részlistás *hash* tábla kialakítása, a meghívási idő alapján.
- Minden listaelem első paramétere a meghívásig hátralévő abszolút időt tárolja tikk-ben.
- N elemű *hash* tábla esetén az 1. listában azok a függvények szerepelnek, amelyek tikk-je N-nel osztva 1-et adnak maradékul. A 2.-ban 2-t, a 3.-ban 3-mat az N.-ben N-1-et.
- Így átlagosan egy részlista az N-ed részét teszi ki az előbbihez képest.

Call-out függvények tárolása időkerékkel