

UNIX: folyamatok ütemezése

kiegészítő fóliák az előadásokhoz

Mészáros Tamás

<http://home.mit.bme.hu/~meszaros/>

*Budapesti Műszaki és Gazdaságtudományi Egyetem
Méréstechnika és Információs Rendszerek Tanszék*

Az előző részben történt

- Mi a folyamat?
 - A feladatainkat megoldó program futás alatt álló példánya
- A folyamat viszonya a kernelhez
 - futási mód és kontextus
 - rendszerhívás interfész
 - kétféle futó állapot
 - az FK és F állapotok között kétirányú átmenet: preemptív ütemező
- A folyamat adminisztratív adatai
 - PID: folyamat azonosító (PPID: szülő azonosítója)
 - UID, GID: tulajdonos és csoport
 - aktuális állapot (F, FK, A, ...)
 - ütemezési információk
 - prioritás
 - nice érték

Áttekintés

- A tradicionális UNIX ütemezés
 - tulajdonságai
 - működése kernel és felhasználói módban
 - prioritás számítása
 - részletes ütemezési algoritmus (felhasználói módban)

- Ütemezési mintapéldák

- Modern UNIX ütemezők
 - tulajdonságai, követelmények
 - a Solaris és a Linux ütemezők jellemzői

Feladatok ütemezése (ismétlés)

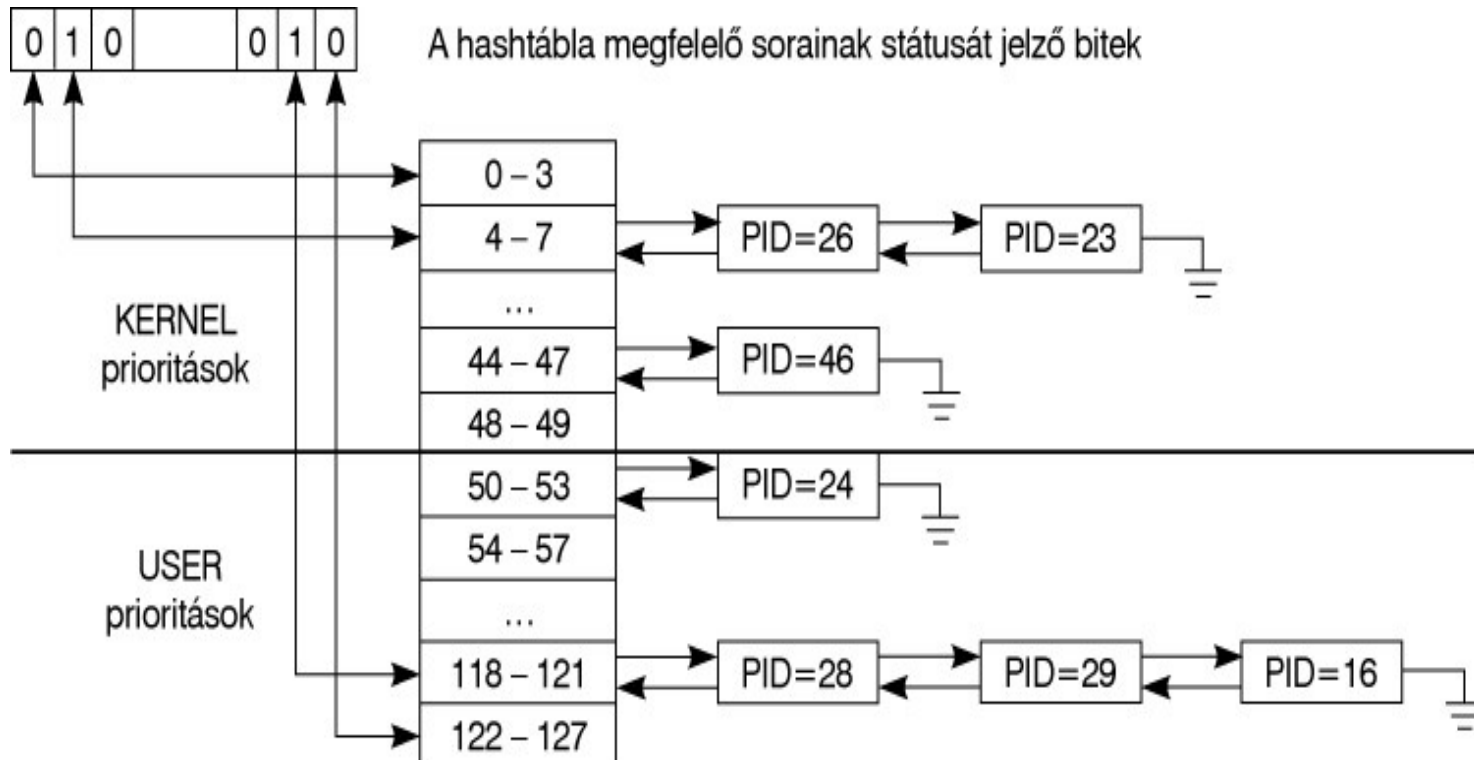
- Az ütemezés alapfeladata:
 - kiválasztani a futásra kész feladatok közül a következő futót
- Alapfogalmak
 - preemptív ill kooperatív ütemező
 - mértékek: CPU kihasználtság, átbocsátó képesség, várakozási idő, körülfordulási idő, válaszidő
 - prioritás
 - statikus és dinamikus ütemezés
- A működés alapelve
 - a feladatokat sorokba (task/job queue) rendezzük (jellemzően FIFO)
 - az ütemező a sorokból választja ki a következő futó folyamatot
- Követelmények
 - minél kisebb erőforrásigény (overhead), komplexitás: $O(1)$, $O(\log N)$
 - optimalitás a mértékek valamilyen kombinációja szerint
 - determinisztikus, fair, elkerüli a kiéhezést, összeomlás elkerülése
 - speciális körülmények egyedi igényei (real-time, batch, multimédia, stb.)

A tradicionális UNIX ütemezők jellemzése

- A UNIX ütemező preemptív, prioritásos és időosztásos
- Az ütemezés **prioritásos**
 - minden folyamat kap egy dinamikusan számított prioritás értéket
 - a legnagyobb prioritású folyamatot választja a futásra kész sorból
- Az ütemezés **időosztásos**
 - több folyamat konkurens futását támogatja
 - minden folyamat kap egy időszeletet, amíg futhat
 - az idő letelte után az azonos prioritásúak közül a következő fog futni
- **A felhasználói és kernel mód ütemezése eltérő**
 - felhasználói módban: preemptív és időosztásos, időben változó prioritás
 - kernel módban: nem preemptív, nincs időosztás, rögzített prioritás

A tradicionális UNIX ütemezés adatstruktúrái

- A prioritás 0-127 közötti értéket vehet fel
 - a 0 a legmagasabb prioritás, a 127 a legkisebb
 - 0-49: kernel prioritási szintek 50-127: felhasználói szintek
- Az ütemező a folyamatokat prioritásuk szerint 32 db sorba rendezi:



Ütemezés kernel módban

- A kernel módú ütemezés tulajdonságai:
 - a prioritás rögzített értékű
 - nem preemptív, így egyszerűbb a kernel adatstruktúrák védelme
- A prioritás nem függ attól, hogy
 - mennyi volt a prioritása felhasználói módban
 - mennyi processzor időt kapott korábban
- A kernel módú prioritás meghatározása
 - A prioritást a folyamat elalvásának az oka határozza meg.
 - Ez az ún. **alvási prioritás** (sleep priority).
 - A folyamat felébredése után ez fogja meghatározni a prioritását.
 - Az alvási prioritás az alvás okához kötött, pl.:
 - 20 diszk I/O-ra vár
 - 28 inputra vár a karakteres terminálról

Ütemezés felhasználói módban

- Felhasználói módban a számított prioritás az ütemezés alapja
 - mindig a legnagyobb prioritású folyamatot ütemezi be a kernel
 - megnézi a hash táblát, és kiválasztja a legmagasabb futási szintet, ahol van folyamat
 - ha ez magasabb, mint a futó folyamat szintje, akkor átütemez
 - ha ez azonos a futó folyamat szintjével, és letelt a futó folyamat időszelete, akkor szintén átütemez
- Az ütemezési tevékenység idő szerint rendezve
 - minden óraciklusban
 - Van-e magasabb szinten folyamat? Ha igen, akkor átütemezés.
 - A legmagasabb, folyamatot tartalmazó futási szint sorban első folyamata fog futni.
 - minden időszelet végén (10 óraciklus)
 - Van-e másik folyamat az éppen futó folyamat prioritásának megfelelő futási szinten? Ha igen, akkor átütemezés.
 - Az adott szint első folyamata fog futni. A preemtált folyamat a sor végére kerül.
 - Azaz az időosztásos ütemezési rész **Round-Robin algoritmus** szerinti.

Az ütemezés további adatai (prioritás számításhoz)

- A kernel a következő adatokkal írja le a folyamatokat
 - p_pri a folyamat aktuális prioritása
 - p_usrpri a folyamat felhasználói módban érvényes prioritása
 - p_cpu a CPU használat mértékére vonatkozó szám
 - p_nice a felhasználó által adott prioritás módosító értéke
- Az ütemező aktuális prioritás érték alapján dönt az átütemezésről
- Kernel módba váltáskor a prioritás értéke „elmentődik” annak érdekében, hogy felhasználói módba visszalépéskor visszaállítható legyen a p_usrpri értékéből.
- A CPU érték kifejezi, hogy a folyamat az elmúlt időszakban mennyi processzor időt kapott. Ez csökkenteni fogja a prioritását, azaz biztosítható a fair ütemezés.

A prioritás számítása felhasználói módban

- Minden óraciklusban a futó folyamatra növeli a `p_cpu` értékét
 p_cpu++

- Minden 100. óraciklusban a kernel kiszámítja a prioritás értékét
 - minden folyamatnál „öregíti” a `p_cpu` értéket

$$p_cpu = p_cpu * KF$$

KF: korrekciós faktor

- kiszámolja a prioritást

$$p_pri = P_USER + p_cpu / 4 + 2 * p_nice \quad P_USER = 50$$

- A korrekciós faktor számítása
 - SVR3: $KF = 1/2$ (Mi ezzel a baj?)
 - 4.3 BSD: a futásra kész folyamatok átlagos számától (`load_avg`) függ:
 $KF = 2 * load_avg / (2 * load_avg + 1)$

A felhasználói módú ütemezés összefoglalója

- Minden óraütésnél
 - Ha van magasabb prioritási sorban folyamat, akkor átütemezés
 - A futó folyamat `p_cpu` értékének növelése (csökkenni fog a prioritása)
- Minden 10. óraütésnél
 - Round-Robin ütemezés a legmagasabb (felhasználói) futási szinten levő folyamatok között
- Minden 100. óraütésnél
 - a korrekciós faktor kiszámítása az elmúlt 100 ciklus átlagos terhelése alapján
 - a `p_cpu` „öregítése”
 - a prioritás újraszámolása
 - szükség szerint átütemezés

Ütemezési mintapéldák

- Három folyamat, Round-Robin ütemezés nélkül

...

- Ütemezés Round-Robin ütemezéssel együtt

...

A tradicionális UNIX ütemezés értékelése

- + Egyszerű és hatékony.
- + Általános célú időosztásos rendszerben megfelelő.
- + Jól kezeli az interaktív és batch típusú folyamatok keverékét.
- + Elkerüli az éhezést.
- + Jól támogatja az I/O műveleteket végrehajtó folyamatokat.

- Nem skálázódik jól: ha sok a folyamat, akkor sok a számítás.
- Nem lehet folyamat (csoportok) számára CPU-t garantálni.
- Nem lehet a folyamatokat igényeik szerint eltérően ütemezni.
- Nincs garancia a válaszidőre.
- A többprocesszoros támogatás nem kidolgozott.
- A kernel nem preemptív:
 - Ha egy folyamat sokáig fut kernel módban, akkor feltartja az egész rendszert.
 - prioritás inverzió:** egy alacsonyabb prioritású folyamat feltart egy nála magasabb prioritású folyamatot.

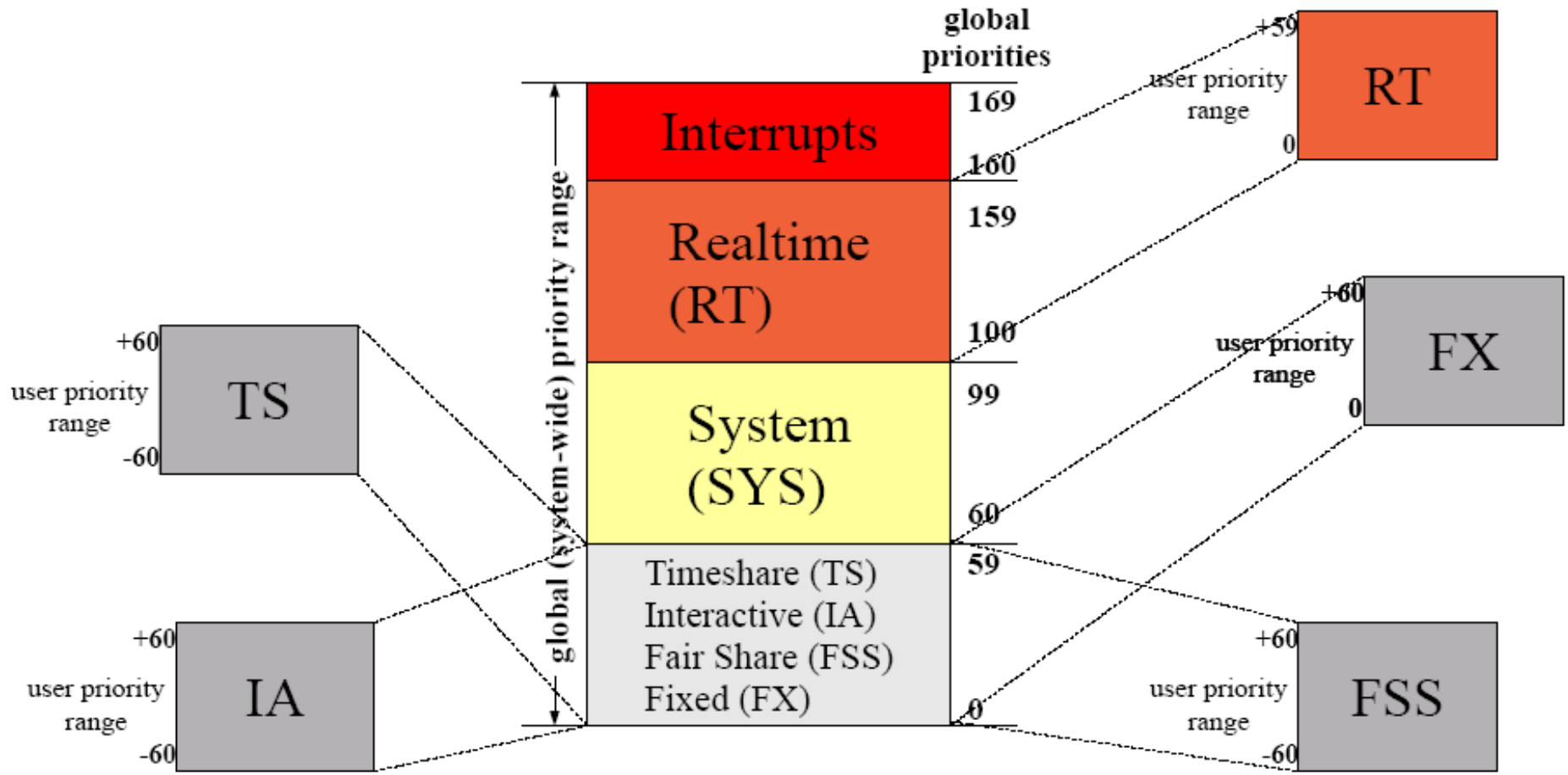
Modern UNIX ütemezők szempontjai

- 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 szükséges: bővíthető keretrendszer
- Kernel megszakíthatóság
 - a tradicionális UNIX kernel nem preemptív (szűk keresztmetszet)
 - a többprocesszoros (többmagos) ütemezéshez elengedhetetlen
- Az ütemező erőforrásigénye (teljesítménye)
 - az ütemezés egyre összetettebb feladat (több processzor, többféle sor)
 - 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

A Solaris ütemező tulajdonságai

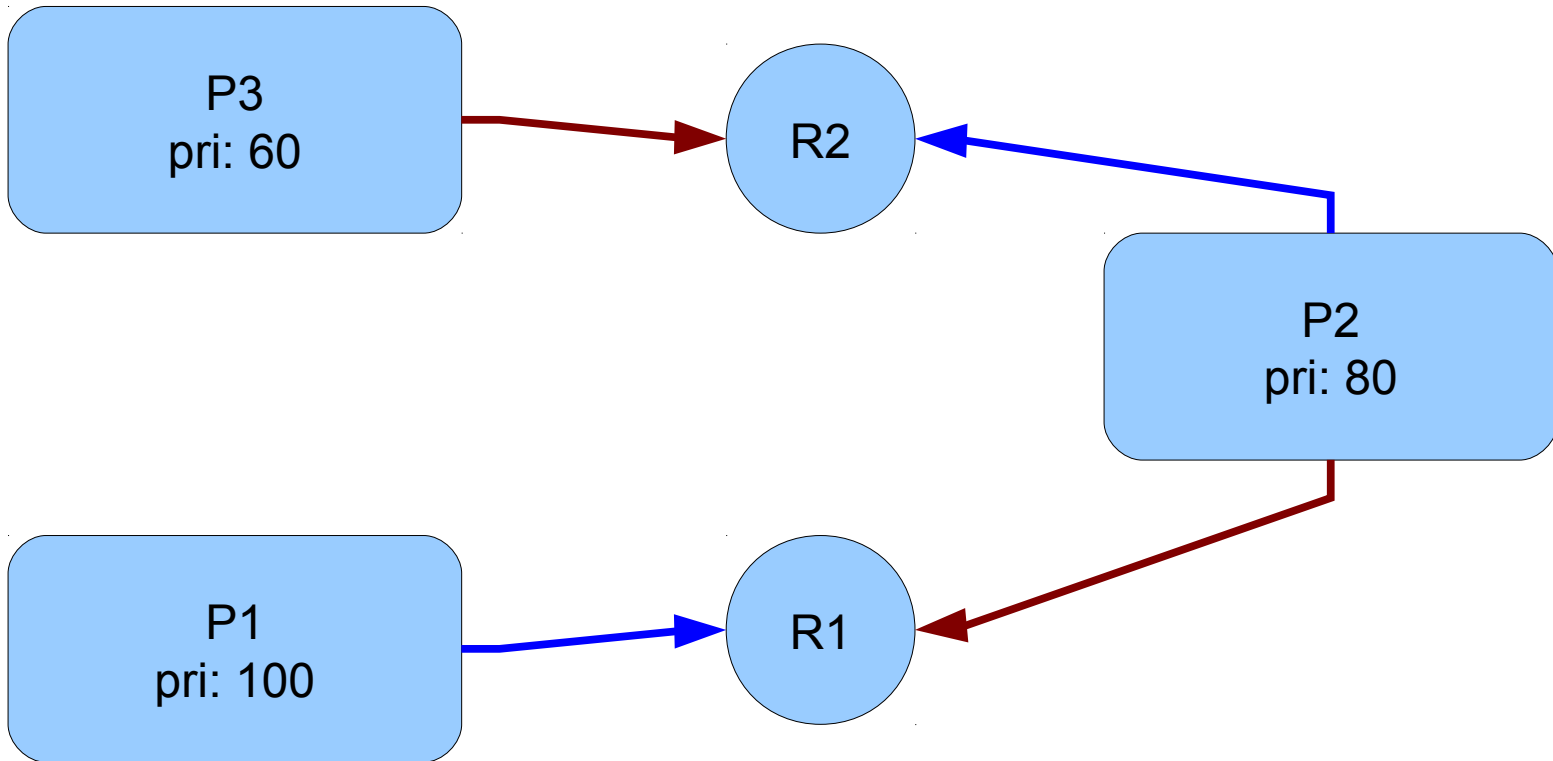
- Alapvető tulajdonságok
 - az ütemezés szálalapú
 - a kernel is preemptív
 - támogatja a többprocesszoros rendszereket és a virtualizációt (zónák)
- Többféle ütemezési osztály
 - Időosztásos (TS): az ütemezés alapja az, hogy ki mennyit várt/futott
 - Interaktív (IA): azonos a fentivel, de az éppen aktív ablakhoz tartozó folyamat kiemelése
 - Fix prioritás (FX): állandó prioritású folyamatok
 - Fair share (FSS): CPU erőforrások folyamatcsoportokhoz rendelése
 - Real-time (RT): a legkisebb késleltetést biztosító legmagasabb szint
 - Kernel szálak (SYS): az RT szint kivételével mindennél magasabb szint

A Solaris ütemezési szintjei



Örökölt prioritások (Solaris)

- A prioritás inverzió problémája (kék nyíl: vár, piros nyíl: foglal)
- Megoldás: prioritás emelés, avagy örökölt prioritások



Linux ütemezők története, tulajdonságai

- Az első változatok (v2 előtt) a tradicionális ütemezőre épültek
- 2.4-es kernel előtt
 - ütemezési osztályok: real-time, nem-preemptív, normál
 - összetettebb ütemezés, amely $O(N)$ komplexitású
 - még egyetlen ütemezési sor (nincs SMP támogatás)
 - nem preemptív kernel
- kernel v2.6 (Molnár Ingo)
 - $O(1)$ ütemező, jobban skálázható
 - processzoronkénti futási sorok (jobb SMP támogatás)
 - I/O és CPU terhelésű folyamatok heurisztikus szétválasztása
 - a futási és alvási idők összevetése alapján (a számítás elég sok idő)
 - előbbieket előnyben részesíti
- 2.6.23 kernel: CFS (Completely Fair Scheduler)
 - Con Kolivas egyes ütemezési ötleteit felhasználva Molnár Ingo munkája
 - a cpu idő biztosításának kiegyensúlyozása egy speciális adatstruktúra segítségével („időben rendezett piros-fekete fa”)

Összefoglalás

- A klasszikus UNIX ütemező
 - felhasználói módban: prioritásos, időosztásos, preemptív
 - többszintű ütemezési tevékenység
 - azonos prioritási szinten round-robin ütemezés (időosztás)
 - a prioritás alapja a korábbi CPU használat és a *nice* érték
 - kernel módban
 - rögzített prioritás, alapja az ún. alvási prioritás
 - nem preemptív
 - egyszerű, kerüli az éhezést, de rosszul skálázható és nem támogat sok mai alkalmazási igényt (valósidejű rendszerek, multimédia, stb.)
- Modern UNIX ütemezők
 - moduláris felépítésű ütemező rendszer
 - több ütemezési osztály az alkalmazási igényeknek megfelelően
 - többprocesszoros rendszerek támogatása
 - jobb erőforrás-allokáció
 - szálak ütemezése