Operációs rendszerek

ELTE IK.

Dr. Illés Zoltán

zoltan.illes@elte.hu

Miről beszéltünk korábban…

- Operációs rendszerek kialakulása
- Op. Rendszer fogalmak, struktúrák
- Fájlok, könyvtárak, fájlrendszerek
- Folyamatok
- Klasszikus IPC problémák
- Folyamatok ütemezése
- I/O, holtpont probléma
- Memória kezelés

Mi következik ma…

- Operációs rendszer feladatok újragondolása
- Mi hiányzik a ma megoldásaiból?
- Milyen igények jelentkeznek mai rendszer környezetekben?
  - Felhasználói oldalról
  - Környezetünk eszközeitől
  - Ipari, gazdasági oldalról
- Az IDŐ szerepének megjelenése!
  - (Real -Time Systems)

Mai operációs rendszer jellemzők

- Preemptív rendszerek
  - Prioritással rendelkező folyamatok
  - Jellemző ütemezés, CFS vagy ehhez nagyon hasonló elvű.
- Multi task, hasonló elvű fájlrendszerek.
- Szegmentált, virtuális memóriakezelés.
- Réteges szerkezű I/O, nincs felügyelet.
- Nincs kiéheztetés, minden folyamat előbb vagy utóbb végrehajtásra kerül!
  - Grafikus UI

Mi hiányzik?

- Összességében semmi, minden szép, minden jó …
- Egyetlen dolog maradt ki az előző rendszer jellemzők felsorolásából!
- Az idő szerepe másodlagos!
  - Mindenki egyenlő (prioritást is figyelembe véve)
  - Valamikor mindenki erőforráshoz(CPU) jut!
- Elég-e az, hogy valamikor?
- Igaz, hogy a számítógépes rendszerek teljesítménye folyamatosan nő, ennek ellenére az idő szerepét újra kell gondolni!

Valós idejű (Real-Time) rendszer

- Mitől lesz egy rendszer valós idejű?
- Az a rendszer, ami az idő szerepét is figyelembe veszi a folyamatok végrehajtása során, valós idejű rendszer!
- Más megfogalmazással: az operációs rendszer garantálja, hogy egy esemény bekövetkeztekor a rá váró folyamat CPU-hoz jut!
- Miért valós?
  - Mert ha egy feladathoz hozzárendeljük, hogy például délután 5-kor kell végrehajtódnia, (esetleg mikorra kell befejeződnie!) akkor annak nem 5 előtt 1 perccel és nem is 5 után 2 perccel kell!

Szoft vagy Hard Real-Time

- Láttuk egy megfogalmazásban, hogy egy eseményre válaszul egy alkalmazás ütemezésre kerül!
  - Válaszidőnek nevezzük az esemény és az alkalmazás ütemezése között eltelt időt!
- Egy rendszert megengedő(bb)nek (soft) nevezünk ha a megkívánt válaszidőtől „kis mértékben” eltérhetünk! (Soft Real-Time)
  - Mi az a „kis mérték”? Alkalmazástól függ…
- Ha a megkívánt válaszidőtől való eltérés megengedhetetlen, akkor szigorú valós idejű rendszerről beszélhetünk!(Hard Real-Time)

Valós idejű rendszerek alkalmazása

- Kétféle valós idejű rendszerről beszélhetünk:
  - Valós idejű operációs rendszer (RTOS)
  - Valós idejű alkalmazás (RTApp)
- Hol használjuk?
- Korábban elsősorban ipari környezetben, feladatuk jellemzően robotok irányítása.
- Mára a mindennapokban is megjelennek az ilyen jellegű feladatok.
  - Bankkártyás fizetés
  - Biztonsági rendszerek
  - Kártyás ajtó, sorompó rendszerek
  - Stb.

Valós idejű rendszerek – Múlt

- Csak a ma igénye a valós idejű feladatok támogatása?
  - Nem
  - Például DOS esetén nem volt akadálya annak, hogy ilyen alkalmazásokat készítsünk.
  - Igaz, bőséges támogatás se állt rendelkezésre!
- Az operációs rendszer nem volt valós idejű, de jellemzően nem is multi-task rendszerek voltak, így gyakorlatilag nem volt akadálya valós idejű alkalmazások készítésének!
  - Példa: Időzítő előkészítés (1CH), saját gépi rutin beillesztése, saját megszakítás kezelés használat.

Valós idejű rendszerek- Jelen

- A ma jellemzően használt operációs rendszerek (Windows, Linux) nem valós idejűek.
  - Bár a Windowsban is megjelent a real-time prioritású folyamat beállítás lehetősége, de ettől még nem RTOS!
- Jellemzően kétféle valós idejű rendszerről beszélhetünk.
  - Beágyazott rendszerek
  - Teljes valós operációs rendszerek

Beágyazott rendszerek

- Különböznek az általános operációs rendszerektől.
- Jellemző feladatuk, ipari berendezések, vezérlések, elektronikai eszközök működésének biztosítása!
  - Pl: Digitális kamerák, GPS eszközök
  - Set-top box firmware,
  - Automotive Infotainment systems
  - stb.
- Windows CE rendszerek(Windows Embedded Compact 7, 2013)
- QNX Neutrino

Teljes RTOS rendszer(ek)

- Nincs Windows alapú rendszer!
- Linux alapú több is létezik.
  - Pl: RTLinux
  - Real-Time Linux for Debian
  - Suse Linux Enterprise Real-Time Extension
  - Stb.
- Jellemzően ingyenesen elérhető támogatás nélküli verzió is!
  - Jelenleg például a SLE 11 SP4 RT a legfrissebb verzió!

Rendszerjellemzők-fontosabb témakörök

- Real-Time tulajdonságok
- Teljesítmény, CPU védelem(CPU shield)
- Ütemezés
- RT interprocess kommunikáció
- Szinkronizáció
- RT szignálok
- Órák, időzítők

SLE RT tulajdonságok

- Processzor védelem
- Processzor hozzárendelés
- Ütemezés, prioritás váltás
- I/O prioritás változtatása
- Posix RT Extensions
  - Memória rezidens programok
  - Folyamatok szinkronizációja
  - Aszinkron, szinkron I/O
  - Szignálok, időzítők, üzenetek

Processzor védelem(shield)

- Időosztásos, preemptív rendszerben minden folyamatot egy az ütemezőtől függő processzor hajt végre!
- Dedikáljunk ki egy (vagy több) CPU magot a magas prioritású (RT) folyamatok számára.
  - Eredményül a védett központi egységek biztosítják a gyors válaszidőt, megszakításkezelést, determinisztikus végrehajtást, határidők betartását!
  - A nem védett magok a rendszer többi folyamatát, eszközeit szolgálják ki!

CPU halmazok kialakítása

- CPU Set, CPU halmazok kezelése: cset
  - cset shield --cpu=3
- Fontosabb alparancsok:
  - Set
    - cset set -l # cpu set list
  - Shield
    - Cset shield -cpu=1,2,4-6 #1,2,4,5,6 CPU is shielded
  - Proc
    - Cset proc -exec command # run cmd in shield CPU set
- A parancs használatához admin jogosultság kell!
- Help: cset -help

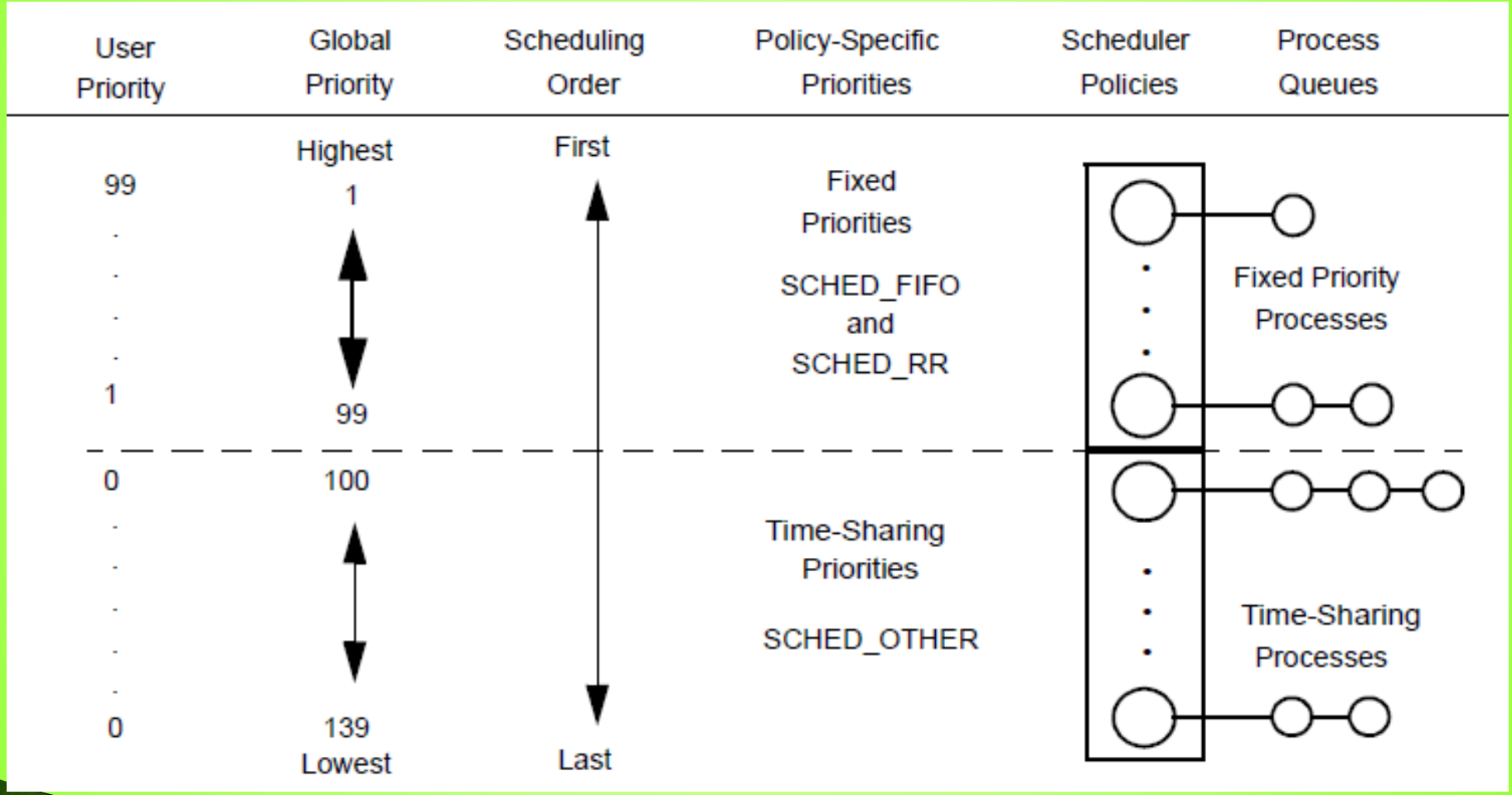
Processzor hozzárendelés

- CPU affinity – taskset command
- Default behavior of the kernel: to keep a running process on the same CPU
- Ugyanakkor a kernel törekszik az azonos CPU terhelés elérésére!(load balance)
- Ütemezés esetén így előfordulhat, hogy egy folyamatot egyik CPU-ról egy másikra helyez át!
- Ezt akadályozza meg a processzor hozzárendelés!

Taskset parancs

- Taskset --help # alapvető lehetőségek
  - Taskset paraméter nélkül ugyanezt végzi!
- Fontosabb parancsok:
  - Egy folyamat CPU hozzárendelését megnéz:
    - Taskset -p pid
  - Egy folyamat CPU hozzárendelését beállítja:
    - Taskset -p mask pid
  - Folyamat futtatása:
    - Taskset mask command
  - Mask helyett CPU sorszám írható
    - /proc/cpuinfo állomány

Schedulers in RT environment

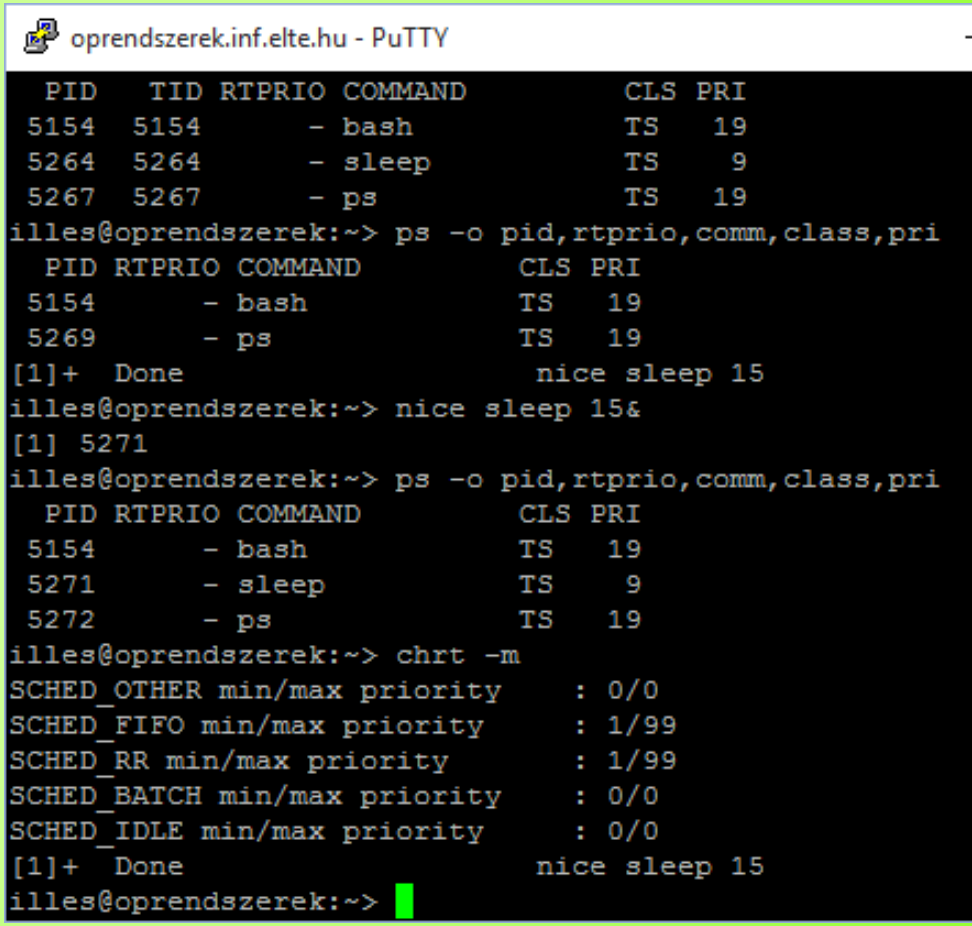


Valós idejű attribútum állítás

- Chrt parancs – ütemező, prioritás változtatása
- Chrt –help # alapvető parancsopciók
- Chrt –m # lehetséges ütemezési # algoritmusok
  - SCHED_OTHER (TS)– alapértelmezett CFS ütemező
  - SCHED_FIFO (FF)– RT FIFO ütemező
  - SCHED_RR (RR)– RT Round Robin ütemező
- Chrt --fifo –p 42 50234 # prioritás:42 # pid: 50234

Chrt használata

- RT prioritás állítás admin jogkörhöz kötött!
- A nice parancs alapból 10-el csökkenti a normál prioritást!
- RT prioritások: 1-99



SCHED_BATCH, SCHED_IDLE

- Láttuk a SCHED_OTHER mellett léteznek!
- SCHED_BATCH
  - Hasonló a SCHED_OTHER alapértelmezett ütemezőhöz!
  - Csak a 0 statikus prioritáson használható! Csak nice 0!
  - CPU intenzív, nem interaktív feladatok esetén lehet hasznos!
- SCHED_IDLE
  - Csak a 0 statikus prioritással használható, a nice érték nem befolyásolja!
  - Alacsony prioritású, háttér folyamatokhoz ajánlott!
  - Alacsonyabb prioritású mint a nice +19! (Ez a táblázat 139-es értéke!)

I/O prioritások

- Láttuk a klasszikus I/O ütemezések jellemzőit, ezek a mai RT környezetben kicsit módosulnak.
- 3 prioritás osztályt használ a rendszer
  - Idle – 3-as osztály(legalacsonyabb), ezen belül nincs nice szint, a nem időkritikus folyamatok számára
  - Best Effort – 2-es osztály, ezen belül 8 szint(0-7), 0 a legmagasabb. Ez az alapértelmezett. Ionice illeszkedik a folyamat nice értékekhez!
  - Real Time – 1-es osztály, szintén 8 szint ezen belül, ez a legmagasabb osztály, ennek kiszolgálása mindenki más előtt megtörténik!

I/O prioritások használata

- Ionice parancs, man ionice
- -c paraméter, I/O prioritás osztály állítása
  - -c1,-c2 vagy -c3 a lehetséges érték
- -p paraméter, process megadása
  - -p 5021 # az 5021 pid számú folyamat
- -n paraméter (elhagyható), ionice megadása
  - -n 3 # 3-as ionice megadása
- Példa:
  - ionice -c3 -p$$ # aktuális shell: idle
  - Ionice -c1 -p5031 -n5 # 5031-es folyamat RT/n=5

Block device I/O scheduler

- Diszk alrendszer ütemező.
- Minden blokkos eszközre beállítható külön-külön saját ütemező!
- Ahogy a fájlrendszereknél láttuk, fő cél a felesleges fejmozgás csökkentése, ezáltal a sávszélesség növelése!
- Jellemző I/O block ütemezések:
  - Noop – Alapértelmezett kérések sorba állítása, tipikusan RAID rendszerek esetén használt!
  - Deadline – Adott határidő előtt válasz, tipikus RT rendszerek esetén haznált.
  - Cfq – Completely Fair Queuing, ez az alapértelmezett.

Blokk I/O ütemező módosítása

- /sys/block könyvtárban találhatók az eszköz leírók! (ezek linkek)
  - Sda – alapértelmezett diszk
  - Fd01 – floppy 01
  - Stb.
- /sys/block/device/queue
  - Adott device paraméterei, adatai
  - Cat /sys/block/sda/queue/scheduler
    - Noop deadline [cfq] # cfq a kiválasztott
  - Sysfs parancs lehetőséget adott blokk eszköz ütemező paraméterek beállításához.
    - Ezek a paraméterek a /sys/block/sda/queue/iosched könyvtárban találhatók!

Deadline I/O ütemezés

- Cél: határidőre el kell végezni az I/O műveletet!
- Két listát használ az ütemező:
  - Egyikben a blokk sorrend alapján szerepelnek a kérések. (Sorban az egymáshoz „közeliek”.)
  - A másikban a határidők alapján rendezettek a kérések!
- Alapértelmezetten a blokk sorrend alapján történik a kiszolgálás, kivéve, ha határidő van, akkor az kerül előre!

Real Time IPC – Queues

- Message Queue and Shared memory
  - Exists in System V too! (msgget, shmget…)
- Posix message queue: implemented as files in /dev/mqueue file system!
- System limits for Posix message queues:
  - Resides in /proc/sys/fs/mqueue directory
  - Msg_max=10, max. message number in each queue, limits by HARD_MAX=131072/sizeof(void*) (appr: 32768)
  - Msgsize_max=8192 (bytes) max message size
  - Queues_max=256, max number of queues

RT message queues

- Compile: -lrt
- Include: <mqueue.h>
- In a message queue each message has the same message slot size!
  - A message size may differ (less or equal) from slot size!
- Every message has a priority!
- The oldest, highest priority message is received first by a process!
- A message is a simple byte set! (char *)
  - Sending numbers, etc, it must cast!

RT message queue features

- Mq_open – opens a message queue
- Mq_send, mq_timedsend – send a message (char*) with a priority (and timespec time delay)
- Mq_receive, mq_timedreceive – receive a message (with a max timespec amount), this call bloks if queue is empty!
- Mq_notify – the calling process is registered for notification of the arrival a message!
- Mq_unlink – removes message queue.

Posix shared memory

- A storage object is defined as a named region and can be mapped by one or more processes!
- Shm_open – creates a shared memory object with zero size!
- Ftruncate – sets the size of memory object
- Mmap – maps to the virtual memory portion
- Fstat – gets the memory size
- Shm_unlink – delete shared memory

Memory mapping

- Establish memory mapping to a target process' address space!
  - Mmap – map a portion of memory to a /proc/pid/mem file and thus directly access the content of another process address space.
    - See man mmap
  - Usermap – an alternativ way for mapping procedure!
    - See man usermap

Interprocess szinkronizáció

- Rescheduling control
- Busy-Wait mutexes
- Posix semaphores
- Extensoins to Posix mutexes
- Condition synchronization

Rescheduling control

- It defers CPU scheduling for brief periods of time.
- Variables Rescheduling
  - Resched_cntl(cmd,arg) registers a variable, and the kernel examines it before rescheduling decision!
  - Resched_lock(v) – locks the variable (increase the number)
  - Resched_unlock(v) – unlocks the variable, if variable is zero or less then 0, preemption is enable!
  - Resched_nlocks(v) – returns the number of locks

Busy-Wait mutexes

- This is a very low overhead operations!
- Often called: spin lock
  - It uses the spin_mutex structure, <spin.h>
  - General interface functions
    - Spin_init
    - Spin_lock
    - Spin_trylock
    - Spin_islock
    - Spin_unlock
  - The spin lock often used in conjunction with rescheduling control variable!
  - Nopreempt_spin_mutex – this automatically uses a rescheduling control variable!

Posix semaphores

- Sem_init – initializes an unnamed semaphore
- Sem_open – creates, init, a named semaphore
- Sem_destroy – remove an unnamed semaphore
- Sem_unlink – remove a named semaphore
- Sem_wait – down the semaphor value (--)
- Sem_post – up the semaphor value (++)
- Sem_trywait, sem_timedwait, sem_getvalue
- Link: -lpthread

Extensoins to Posix mutexes

- Standard Posix mutex functionality:
  - Pthread_mutex…functions
- Robust mutexes – it can detect whether the previous owner is terminated while holding this mutex(errno=EOWNERDEAD). If the new cleanup of mutex can't be done the errno=ENOTRECOVERABLE.
- Priority inheritance – boost the mutex owner priority. A thread locks a mutex and an other higher priority thread goes to sleep for that mutex. In this case the priority of sleeper is temporarily transfers to the owner of mutex!

Condition synchronization

- These functions allows an easy to manipulate cooperating processes!
  - Postwait services – efficient sleep/wakeup/timer mechanism used between cooperating threads.
    - A thread identified with his UKID (Unified global thread Id).
    - Pw_getukid, pw_wait, pw_post,…
  - Server system calls
    - Server_block, server_wake1, server_wakevec

Posix clocks, timers

- CLOCK_REALTIME – the system wide clock, defined in <time.h>
- CLOCK_MONOTONIC – the system time measuring the time in secons and nanosec since the system was booting. It can not be set!
- Timespec, itimerspec structures – see details in manual!
- Clock_settime, clock_gettime, clock_setres
- Timers: sets a value, and sends a notification when it expires!
  - Timer_create, timer_delete,timer_settime,gettime, nanosleep,etc.

Local timers, global timer

- Each CPU has a local (private) timer. It is used as a source of periodic interrupt to that CPU!
  - Cc 100 times per sec
  - Functionality:
    - CPU accounting(eg. For top command, etc)
    - Process time quantum for SCHED_OTHER and SCHED_RR
    - CPU balancing, rescheduling
    - Timing source for Posix timers
  - Local timers can be disabled via shield func!
- Global system wide timer (only one)- uses Int 0, cannot be disabled.

Köszönöm a figyelmet!

zoltan.illes@elte.hu