Operating systems

ELTE IK.

Dr. Illés Zoltán

zoltan.illes@elte.hu

Financed from the financial support ELTE won from the Higher Education Restructuring Fund of the Hungarian Government

We have talked earlier…

- Evolution of operating systems
- Notions of Operating system, structures
- Files,directories, filesystems
- Processes
- Classical IPC problems
- Scheduling of processes
- I/O, deadlock problem
- Memory management

What comes today…

- The rethinking of the operation system tasks
- What is missing from today solutions?
- What type of requirements appear in nowadays system environments?
  - From the users’ side
  - From the devices of our environment
  - From industrial, economical side
- Appearing the importance of TIME!
  - (Real-Time Systems)

Features of nowadays operating systems

- Preemptive systems
  - Processes with priority
  - Typical scheduling, CFS or very similar to it.
- Multi task, similar file systems.
- Segmented, virtual memory management.
- Layered I/O, no supervisoring.
- No starvation, each process will be executed sooner or later!
  - Graphical UI

What is missing?

- On the whole nothing, everything is nice, everything is good …
- Only one thing is missing from the previously mentioned system features!
- The role of time is subsidiary!
  - Everybody is equal (taking notice of priority)
  - Sometime everybody gets resource (CPU)!
- Is it enough, „sometime”?
- It is true, the efficiency of the computer systems continously grow but we must rethink the role of time again!

Real-Time system

- From what becomes a system real-time?
- The system which takes care of the time during the execution of the processes is a real time system!
- With another words: the operating system garantees that in the case of an event the process which waits for it gets to CPU!
- Why it is real?
    - If we assign to a task that it should be executed at 5 o'clock (maybe must finish at 5 o'clock) then we insist on the punctual time – not before or after it with 2 minutes!

Soft or Hard Real-Time

- Due to one of the wordings, an application must be scheduled as a response to an event!
  - We call response time the ellapsed time between the event and the scheduling of the application!
- We call a system soft if we may differ from it „slightly”! (Soft Real-Time)
  - What does it mean „slightly” in this case? It depends on the application…
- If it is not permitted to differ from the required response time that it is called a hard real-time system!

Applying real-time systems

- We may speak about two different types of real-time systems:
  - Real-time operating systems (RTOS)
  - Real-time applications (RTApp)
- Where do we use them?
- Recently typically in industrial environment, their task was to control robots.
- Nowadays these kinds of tasks appeared in every day life too.
  - Paying with credit cards
  - Security systems
  - Card door lock system, barrier systems
  - etc.

Real-time systems – Past

- It is only the requirement of nowadays to support real-time tasks?
  - No
  - In the case of DOS there was not any barrier to write such applications!
- The operating system itself was not real-time, but typically they were not multi-task systems, so you might implement real-time applications!
  - E.g.: Timer preparing (1CH), to insert your own machine code, to use a custom made interrupt handling.

Real-time systems – Present

- Nowadays typically used operating systems (Windows, Linux) are not real-time ones.
- Though in Windows too appeared the possibility to set real-time priority for processes, but it is not an RTOS!
- Typically we can speak about two types of real-time systems.
  - Embedded systems
  - Full real-time operating systems

Embedded systems

- They are differ from general operating systems.
- Typical tasks are, to grant the working of industrial equipments, controllings, electronic devices!
  - e.g.: Digital cameras, GPS devices
  - Set-top box firmware,
  - Automotive Infotainment systems
  - etc.
- Windows CE systems (Windows Embedded Compact 7, 2013)
- QNX Neutrino

Full RTOS system(s)

- There is no such Windows based system!
- There are several Linux based RTOSs.
  - e.g.: RTLinux
  - Real-Time Linux for Debian
  - Suse Linux Enterprise Real-Time Extension
  - Etc.
- Typically there is a free version too without support!
  - Today the SLE 11 SP4 RT is the newest version!

System features – important themes

- Real-Time features
- Efficiency, CPU shield
- Scheduling
- RT interprocess communication
- Synchronozation
- RT signals
- Clocks, timers

SLE RT features

- Processor shield
- Processor affinity
- Scheduling, priority changing
- I/O priority changing
- Posix RT Extensions
  - Memory rezident programs
  - Process synchronization
  - Asynchron, synchron I/O
  - Signals, timers, messages

Processor shield

- In a time sharing, preemptive system each process is executed by a processor controlled by the scheduler!
- Let us dedicate one (or more) CPU core to the high priority (RT) processes.
  - Result of it: the shielded CPU-s grant the quick response time, interrupt handling and keeping of deadlines!
  - Not shielded cores will serve the other processes, devices of the system!

Shaping up of CPU sets

- Handling CPU sets: cset
  - cset shield --cpu=3
- Important subcommands:
  - 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
- One can use this command only with administrator permission!
- Help: cset -help

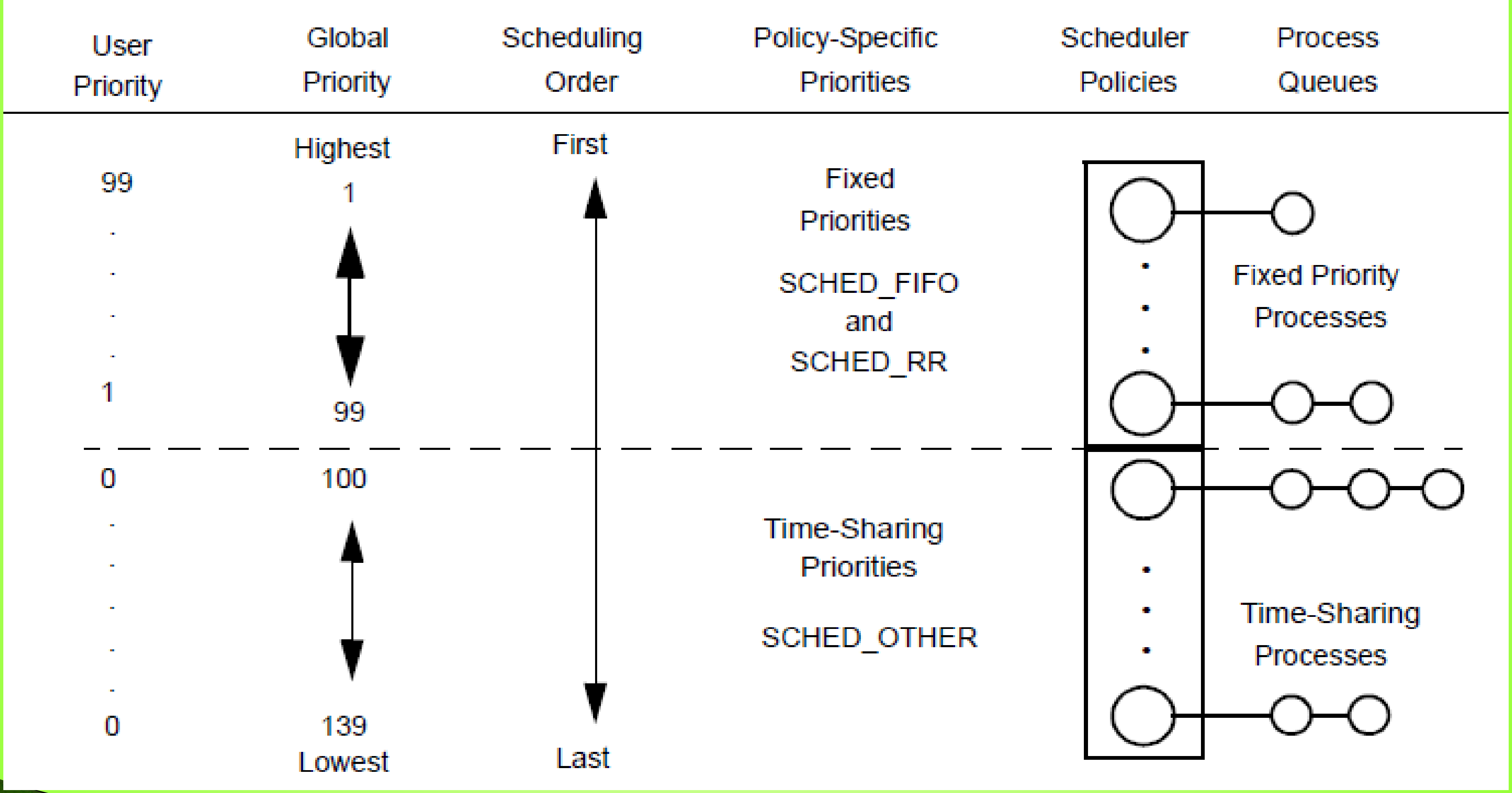
Processor affinity

- CPU affinity – taskset command
- Default behavior of the kernel: to keep a running process on the same CPU
- In the same time the kernel try to get a balanced CPU load! (load balance)
- During the scheuling sometimes may happen that a process is transfered from a CPU to an other one!
- Processor affinity prevents this!

Taskset command

- Taskset --help # basic possibilities
  - Taskset #does the same by using it without parameters!
- Important commands:
  - To have a look at the CPU affinity of a process:
    - Taskset –p pid
  - To set the CPU affinity of a process:
    - Taskset –p mask pid
  - To execute a process:
    - Taskset mask command
  - Instead of Mask you may write the ordinal number of the CPU
    - /proc/cpuinfo file

Schedulers in RT environment



Setting of real-time attributes

- Chrt command – scheduler, priority changing
- Chrt –help # default command options
- Chrt –m # possible scheduling # algorithms
  - SCHED_OTHER (TS)– default CFS scheduler
  - SCHED_FIFO (FF)– RT FIFO scheduler
  - SCHED_RR (RR)– RT Round Robin sheduler
- Chrt --fifo –p 42 50234 # priority:42 # pid: 50234

Chrt usage

- To set an RT priority you should have an admin permission!
- The nice command default meaning is to decrease the normal priority with 10!
- RT priorities are: 1-99

oprendszerek.inf.elte.hu - PuTTY

```
  PID   TID RTPRIO COMMAND         CLS PRI
 5154  5154      - bash            TS   19
 5264  5264      - sleep           TS    9
 5267  5267      - ps              TS   19
illes@oprendszerek:~> ps -o pid,rtprio,comm,class,pri
  PID RTPRIO COMMAND         CLS PRI
 5154      - bash            TS   19
 5269      - ps              TS   19
[1]+  Done                    nice sleep 15
illes@oprendszerek:~> nice sleep 15&
[1] 5271
illes@oprendszerek:~> ps -o pid,rtprio,comm,class,pri
  PID RTPRIO COMMAND         CLS PRI
 5154      - bash            TS   19
 5271      - sleep           TS    9
 5272      - ps              TS   19
illes@oprendszerek:~> chrt -m
SCHED_OTHER min/max priority    : 0/0
SCHED_FIFO min/max priority     : 1/99
SCHED_RR min/max priority       : 1/99
SCHED_BATCH min/max priority    : 0/0
SCHED_IDLE min/max priority     : 0/0
[1]+  Done                    nice sleep 15
illes@oprendszerek:~>
```

SCHED_BATCH, SCHED_IDLE

- Besides SCHED_OTHER there are others we have seen it before!
- SCHED_BATCH
  - Similar to SCHED_OTHER default scheduler!
  - It can be used only with static priority! (There is only nice 0!)
  - It can be useful in the case of CPU intensive, not interactive tasks!
- SCHED_IDLE
  - It can be used only with 0 static priority, nice value does not affect it!
  - It is suggested for low priority background processes!
  - It has a lower priority then nice +19! (This is the 139 value of the table!)

I/O priority

- We have seen the features of classical I/O schedulers they are modified a bit in a modern RT environment.
- The system uses 3 priority class
  - Idle – 3 class (lowest), it is for not time critical processes, in it there is no nice level
  - Best Effort – 2 class, in it there are 8 levels (0-7), 0 is the highest. This is the default one. Ionice value fits to the nice value of the process!
  - Real Time – 1class, 8 levels, this is the highest class, serving this is the first before anything else!

I/O priority usage

- Ionice command, man ionice
- -c parameter, setting of I/O priority class
  - -c1,-c2 or -c3 possible values
- -p parameter, process assigning
  - -p 5021 # process with pid number 5021
- -n parameter (you may omit it),setting ionice
  - -n 3 # to set ionice 3
- E.g.
  - ionice -c3 -p$$ # actual shell: idle
  - ionice -c1 -p5031 -n5 # process 5031 RT/n=5

Block device I/O scheduler

- Disk subsystem scheduler.
- We may set schedulers separatly to each block-type device!
- As we have seen at the filesystems the main goal is to reduce the unwanted head movements and hereby to increase the bandwidth!
- Typical I/O block schedulings:
  - Noop – The alignment of default requirements, mainly used in RAID systems!
  - Deadline – To give a response before the deadline, it is used typically in RT systems.
  - Cfq – Completely Fair Queuing, it is the default.

Modifying the block I/O scheduler

- Device descriptors are in /sys/block directory! (they are links)
  - Sda – default disk
  - Fd01 – floppy 01
  - Etc.
- /sys/block/device/queue
  - The parameters, data of a given device
  - Cat /sys/block/sda/queue/scheduler
    - Noop deadline [cfq] # cfq the choosen
  - Sysfs command makes possible to set the parameters of block device scheduler.
    - These parameters are in the /sys/block/sda/queue/iosched directory!

Deadline I/O scheduling

- Goal: we must finish the I/O task before the deadline!
- The scheduler uses two lists:
  - In one of it there are the requires in block order. (One after the other the „nearby" ones.)
  - In the other one the requests are sorted by the deadlines!
- The default serving is done by the block order except there is a deadline, in this case it will be executed before!

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 alternative way for mapping procedure!
    - See man usermap

Interprocess synchronozation

- 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 operation!
- 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

Extensions 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.

Thanks for your attention!

zoltan.illes@elte.hu