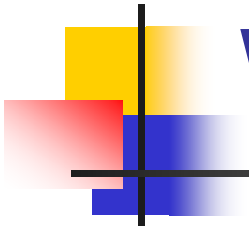
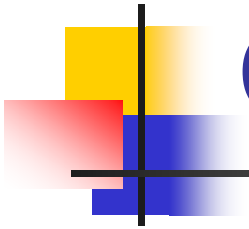


The Linux Scheduler

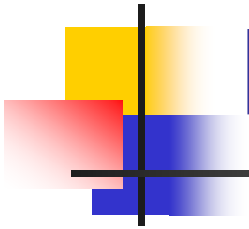


What Are Scheduling Goals?



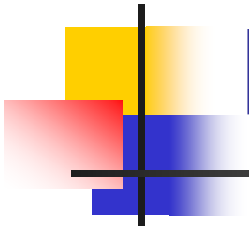
Goals of the Linux Scheduler

- Generate illusion of concurrency
- Maximize resource utilization (hint: mix CPU and I/O bound processes appropriately)
- Meet needs of both I/O-bound and CPU-bound processes
  - Give I/O-bound processes better interactive response
  - Do not starve CPU-bound processes
- Support Real-Time (RT) applications



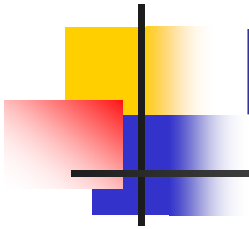
Early Schedulers

- 1.2: circular queue with round-robin policy.
  - Simple and minimal.
  - Does not meet many of the aforementioned goals
- 2.2: introducing scheduling classes (real-time, non-real-time).



Linux 2.4 Scheduler

- 2.4: O(N) scheduler.
  - Epochs → slices: when blocked before the slice ends, half of the remaining slice is added in the next epoch.
  - Simple.
  - Lacked scalability.
  - Weak for real-time systems.



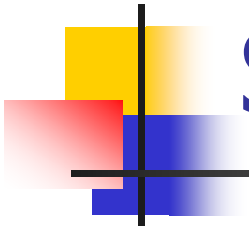
Linux 2.6

- O(1) scheduler
- Tasks are indexed according to their priority [0,139]
  - Real-time [0, 99]
  - Non-real-time [100, 139]

Scheduler API

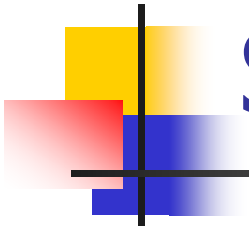
System call	description
nice()	change the priority
getpriority()	get the maximum group priority
setpriority()	set the group priority
sched_getscheduler()	get the scheduling policy
sched_setscheduler()	set the scheduling policy and priority
sched_getparam()	get the priority
sched_setparam()	set the priority
sched_yield()	relinquish the processor voluntarily
sched_get_priority_min()	get the minimum priority value
sched_get_priority_max()	get the maximum priority value
sched_rr_get_interval()	get the time quantum for Round-Robin

Two Fundamental Resource Sharing Mechanisms

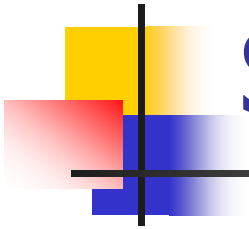


- ?
- ?

Two Fundamental Resource Sharing Mechanisms

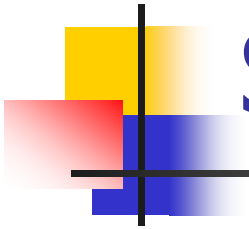


- Prioritization
- Resource partitioning



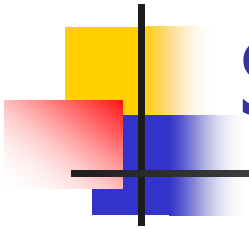
SCHED_FIFO (Prioritization)

- Used for real-time processes
- Conventional preemptive fixed-priority scheduling
  - Current process continues to run until it ends or a higher-priority real-time process becomes runnable
- Same-priority processes are scheduled FIFO



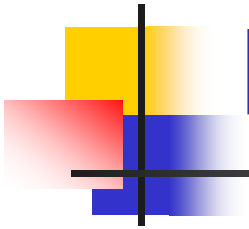
SCHED_RR (Partitioning)

- Used for real-time processes
- CPU "partitioning" among same priority processes
  - Current process continues to run until it ends or its time quantum expires
  - Quantum size determines the CPU share
- Processes of a lower priority run when no processes of a higher priority are present



SCHED_NORMAL

- Processes start at 120 by default
- Static priority
  - A "*nice*" value: 19 to -20.
  - Inherited from the parent process
  - Altered by user (negative values require special permission)
- Dynamic priority
  - Based on static priority and applications characteristics (interactive or CPU-bound)
  - Favor interactive applications over CPU-bound ones
- Timeslice is mapped from priority



Heuristics

if (static priority < 120)

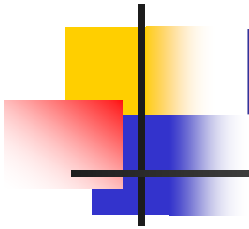
    Quantum = 20 (140 – static priority)

else

    Quantum = 5 (140 – static priority)

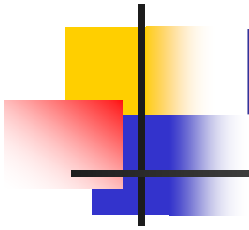
(in ms)

Higher priority → Larger quantum



Example

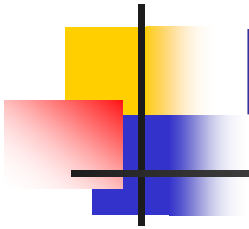
Description	Static priority	Nice value	Base time quantum
Highest static priority	100	-20	800 ms
High static priority	110	-10	600 ms
Default static priority	120	0	100 ms
Low static priority	130	+10	50 ms
Lowest static priority	139	+19	5 ms



Heuristics

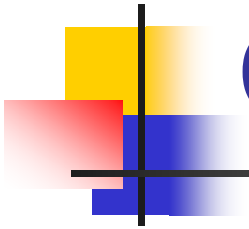
bonus = min (10, avg. sleep time / 100) (ms)

dynamic priority = max (100, min (static priority – bonus + 5, 139))



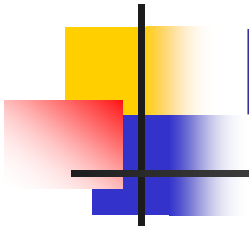
How & When to Preempt?

- Kernel sets the *need_resched* flag (per-process var) at various locations
  - scheduler_tick(), a process used up its timeslice
  - try_to_wake_up(), higher-priority process awaken
- Kernel checks *need_resched* at certain points, if safe, *schedule()* will be invoked
- User preemption
  - Return to user space from a system call or an interrupt handler
- Kernel preemption
  - A task in the kernel explicitly calls *schedule()*
  - A task in the kernel blocks (which results in a call to *schedule())*



Completely Fair Scheduler

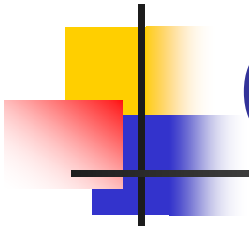
- Merged into the 2.6.23 release of the Linux kernel and is the default scheduler.
- Scheduler maintains a red-black tree where nodes are ordered according to received virtual execution time
- Node with smallest virtual received execution time is picked next
- Priorities determine accumulation rate of virtual execution time
  - Higher priority → slower accumulation rate



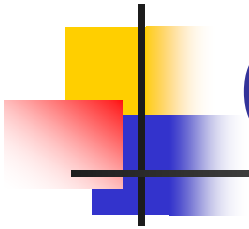
Example

- Three tasks accumulate virtual execution time at a rate of 1, 2, and 3, respectively.
- What is the expected share of the CPU that each gets?

Other Scheduling Policies (… that you can implement)

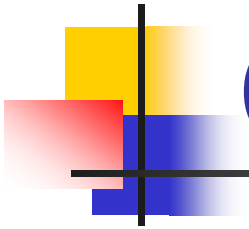


- What if you want to maximize throughput?



Other Scheduling Policies (… that you can implement)

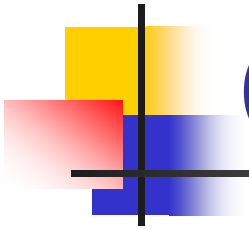
- What if you want to maximize throughput?
  - Shortest job first!



Other Scheduling Policies (… that you can implement)

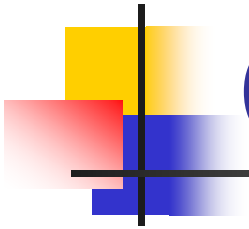
- What if you want to meet all deadlines?

Other Scheduling Policies (… that you can implement)



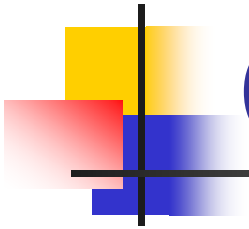
- What if you want to meet all deadlines?
  - Earliest deadline first!

Other Scheduling Policies
(… that you can implement)

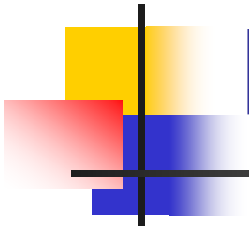


- What if you want to meet all deadlines?
  - Earliest deadline first!
- Problem?

Other Scheduling Policies (… that you can implement)

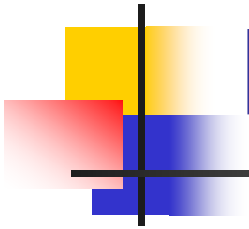


- What if you want to meet all deadlines?
  - Earliest deadline first!
- Problem?
  - Works only if you are not "overloaded". If the total amount of work is more than capacity, a domino effect occurs as you always choose the task with the nearest deadline (that you have the least chance of finishing by the deadline), so you may miss a lot of deadlines!



Example of EDF Domino Effect

- Problem:
  - You have a homework due tomorrow (Tuesday), a homework due Wednesday, and a homework due Thursday
  - It takes on average 1.5 days to finish a homework.
- Question: What is your best (scheduling) policy?



Example of EDF Domino Effect

- Problem:
  - You have a homework due tomorrow (Tuesday), a homework due Wednesday, and a homework due Thursday
  - It takes on average 1.5 days to finish a homework.
- Question: What is your best (scheduling) policy?
  - Note that EDF is bad: It always forces you to work on the next deadline, but you have only one day between deadlines which is not enough to finish a 1.5 day homework – you might not complete any of the three homeworks!
  - You could instead skip tomorrow's homework and work on the next two, finishing them by their deadlines