CS353 Linux Kernel

Chentao Wu吴晨涛
Associate Professor
Dept. of CSE, SJTU
wuct@cs.sjtu.edu.cn

上海交通大學

3B. Process Management -- Scheduling

Chentao Wu
Associate Professor
Dept. of CSE, SJTU
wuct@cs.sjtu.edu.cn

上海交通大學

Outline

- Scheduling Policy
- Scheduling Algorithm
- System Calls related to Scheduling
- Completely Fair Scheduler (CFS)
  - New Algorithm in Linux Kernel 2.6

Scheduling Policy

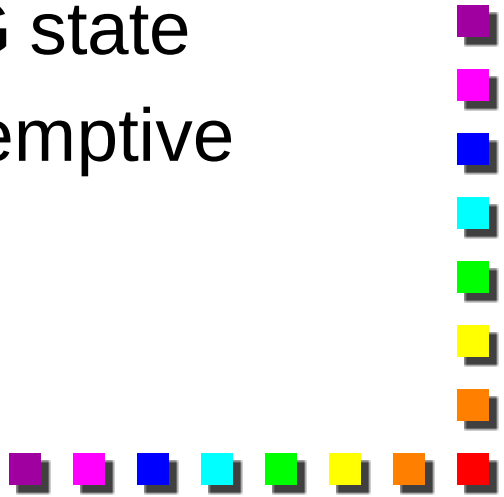
- Based on time-sharing
  - Time slice
- Based on priority ranking
  - Dynamic
- Classification of processes
  - Interactive processes
  - Batch processes
  - Real-time processes

System Calls related to Scheduling

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
sched_setaffinity()	set the CPU affinity mask
sched_getaffinity()	get the CPU affinity mask

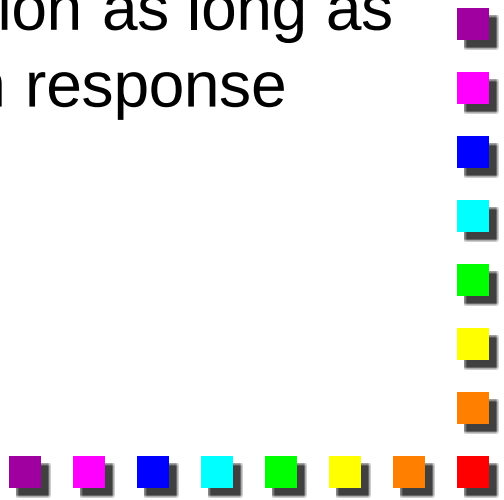
Process Preemption

- Linux (user) processes are preemptive
  - When a new process has higher priority than the current
  - When its time quantum expires, TIF_NEED_RESCHED will be set
- A preempted process is not suspended since it remains in the TASK_RUNNING state
- Linux kernel before 2.6 is nonpreemptive
  - Simpler
  - Linux kernel 2.6 is preemptive



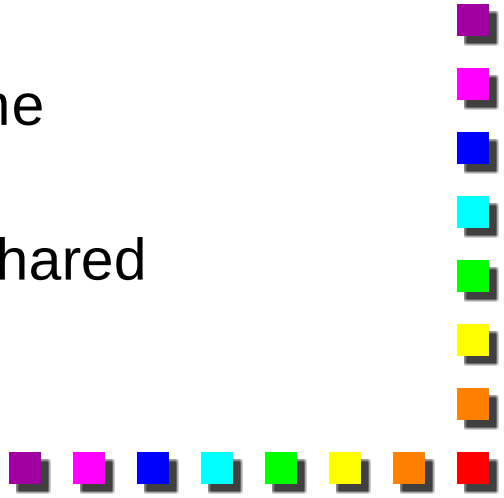
How Long Must a Quantum Last?

- Neither too long nor too short
  - Too short: overhead for process switch
  - Too long: processes no longer appear to be executed concurrently
- Always a compromise
  - The rule of thumb: to choose a duration as long as possible, while keeping good system response time



The Scheduling Algorithm

- Earlier version of Linux: simple and straightforward
  - At every process switch, scan the runnable processes, compute the priorities, and select the best to run
- Much more complex in Linux 2.6
  - Scales well with the number of processes and processors
  - Constant time scheduling
- 3 Scheduling classes
  - SCHED_FIFO: First-In First-Out real-time
  - SCHED_RR: Round-Robin real-time
  - SCHED_NORMAL: conventional time-shared processes



Scheduling of Conventional Processes

- Static priority
  - Conventional processes: 100-139
    - Can be changed by nice(), setpriority() system calls
- *Base time quantum*: the time-quantum assigned by the scheduler if it has exhausted its previous time quantum

$$\text{base time quantum (in milliseconds)} = \begin{cases} (140 - \text{static priority}) \times 20 & \text{if static priority} < 120 \\ (140 - \text{static priority}) \times 5 & \text{if static priority} \geq 120 \end{cases} \quad (1)$$

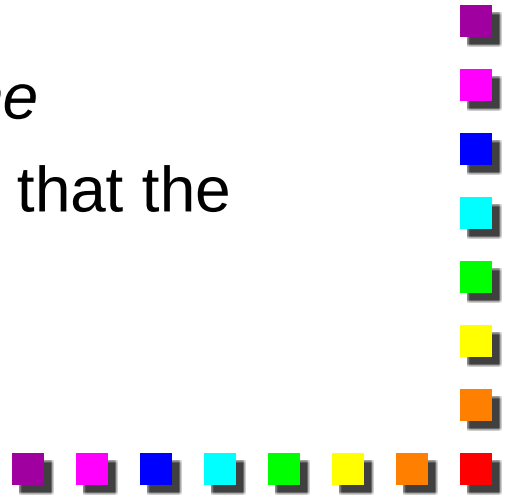
Typical Priority Values for a Conventional Process

Description	Static priority	Nice value	Base time quantum	Interactive delta	Sleep time threshold
Highest static priority	100	-20	800 ms	-3	299 ms
High static priority	110	-10	600 ms	-1	499 ms
Default static priority	120	0	100 ms	+2	799 ms
Low static priority	130	+10	50 ms	+4	999 ms
Lowest static priority	139	+19	5 ms	+6	1199 ms

上海交通大學

Dynamic Priority and Average Sleep Time

- Dynamic priority: 100-139
- Dynamic priority = max (100, min(static_priority – bonus + 5, 139))
  - Bonus: 0-10
    - <5: penalty
    - >5: premium
  - Dependent on the *average sleep time*
    - Average number of nanoseconds that the process spent while sleeping

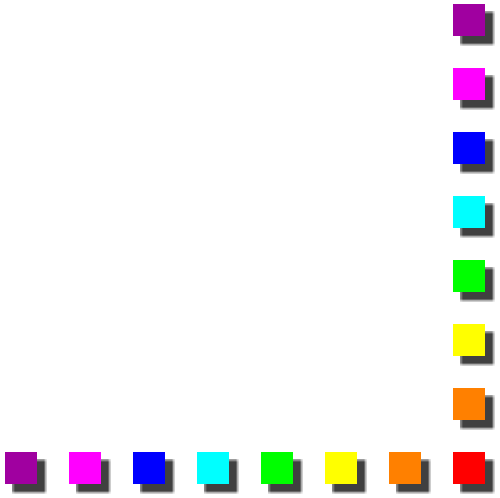


Average Sleep Time, Bonus Values, and Time Slice Granularity

Average sleep time	Bonus	Granularity
>= 0 but < 100 ms	0	5120
>= 100 ms but < 200 ms	1	2560
>= 200 ms but < 300 ms	2	1280
>= 300 ms but < 400 ms	3	640
>= 400 ms but < 500 ms	4	320
>= 500 ms but < 600 ms	5	160
>= 600 ms but < 700 ms	6	80
>= 700 ms but < 800 ms	7	40
>= 800 ms but < 900 ms	8	20
>= 900 ms but < 1000 ms	9	10
1 second	10	10

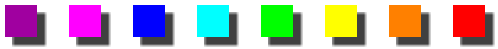
Determine the Status of A Process

- To determine whether a process is considered interactive or batch
  - Interactive if:
    dynamic_priority <= 3* static_priority/4+28
  - Bonus-5 >= static_priority/4-28
    - *interactive delta*



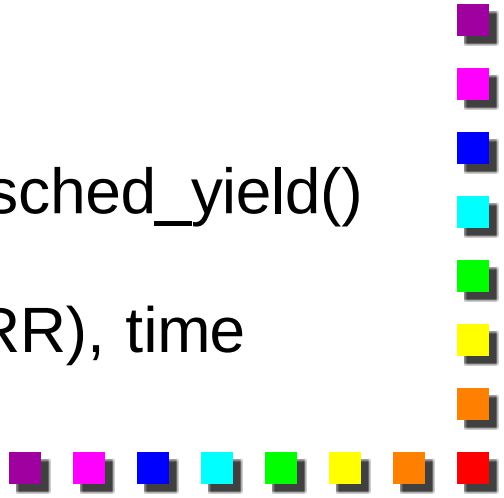
Active and Expired Processes

- *Active processes*: runnable processes that have not exhausted their time quantum
- *Expired processes*: runnable processes that have exhausted their time quantum
- Periodically, the role of processes changes



Scheduling on Real-Time Processes

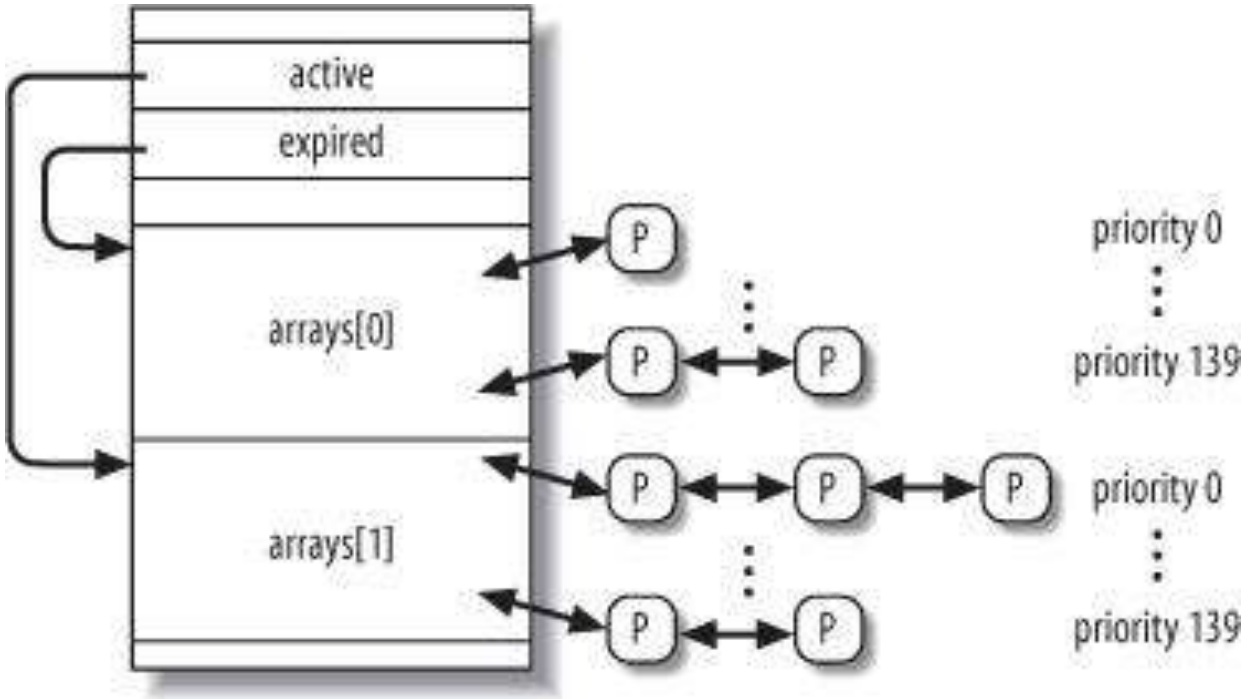
- Real time priority: 1-99
- Real-time processes are always considered active
  - Can be changed by sched_setparam() and sched_setscheduler() system calls
- A real-time process is replaced only when
  - Another process has higher real-time priority
  - Put to sleep by blocking operation
  - Stopped or killed
  - Voluntarily relinquishes the CPU by sched_yield() system call
  - For round-robin real-time (SCHED_RR), time quantum exhausted



Data Structures Used by the Scheduler

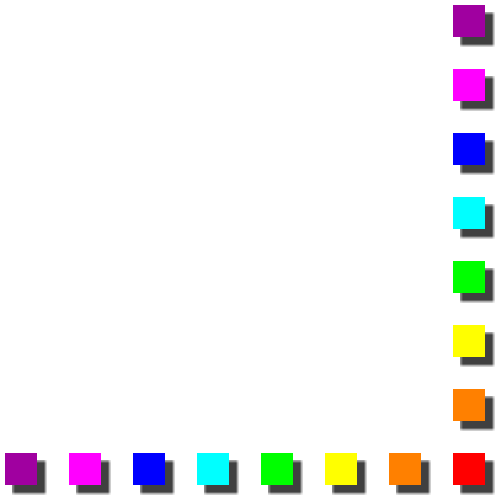
- *runqueue* data structure for each CPU
  - lock: spin lock
  - nr_running: number of runnable processes
  - cpu_load
  - nr_switches: number of process switches
  - nr_uninterruptable
  - expired_timestamp
  - timestamp_last_tick
  - curr: process descriptor pointer of the currently running process
  - idle:
  - prev_mm
  - active: pointer to the list of active processes
  - expired: pointer to the list of expired processes
  - arrays: two sets of active of expired processes
  - best_expired_prio
  - nr_iowait
  - sd
  - active_balance
  - push_cpu
  - migration_thread
  - migration_queue

Two Sets of Runnable Processes in the *runqueue* Structure



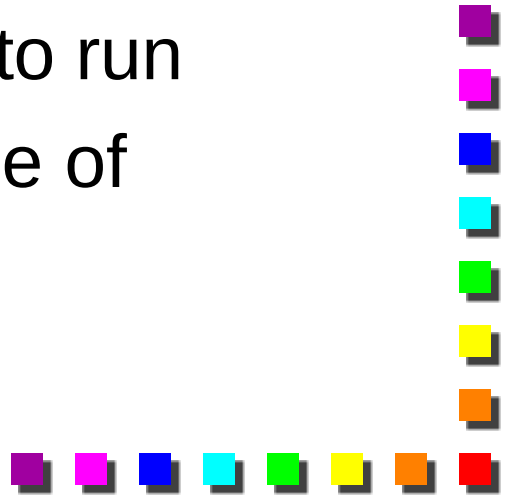
Process Descriptor Fields Related to the Scheduler

- thread_info->flags
- thread_info->cpu
- state
- prio
- static_prio
- run_list
- array
- sleep_avg
- timestamp
- last_ran
- activated
- policy
- cpus_allowed
- time_slice
- first_time_slice
- rt_priority



Functions Used by the Scheduler

- scheduler_tick(): keep the time_slice counter up-to-date
- try_to_wake_up(): awaken a sleeping process
- recalc_task_prio(): updates the dynamic priority
- schedule(): select a new process to run
- load_balance(): keep the runqueue of multiprocess system balanced
- (…Details ignored…)



The schedule() Function

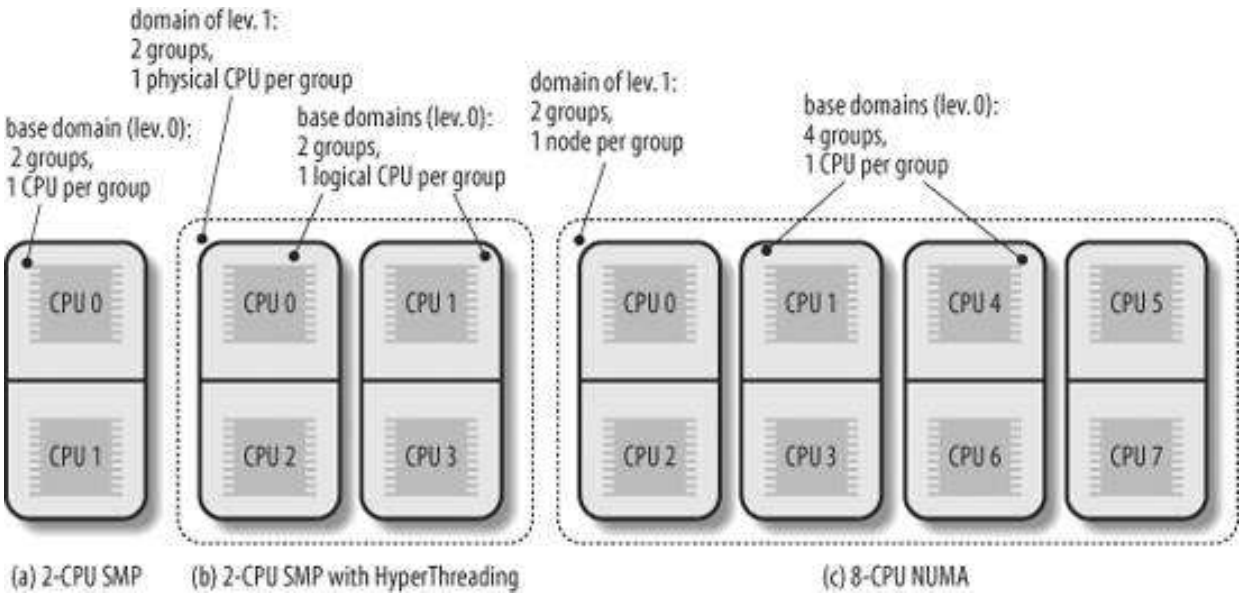
- Direct invocation
  - When the process must be blocked to wait for resource
  - When long iterative tasks are executed in device drivers
- Lazy invocation: by setting the TIF_NEED_RESCHED flag
  - When current has used up its quantum, by scheulertick()
  - When a process is woken up, by try_to_wake_up()
  - When a sched_setscheduler() system call is issued
- Actions performed by schedule() before and after a process switch (…Details ignored…)

Runqueue Balancing in Multiprocessor Systems

- 3 types of multiprocessor systems
  - Classic multiprocessor architecture
  - Hyper-threading
  - NUMA
- A CPU can execute only the runnable processes in the corresponding runqueue
- The kernel periodically checks if the workloads of the runqueues are balanced

Scheduling Domains

- Scheduling domain: a set of CPUs whose workloads are kept balanced bt the kernel
  - Hierarchically organized
  - Partitioned into groups: a subset of CPUs



System Calls Related to Scheduling

- nice(): for backward compatibility only
  - Allows processed to change their base priority
  - Replaced by setpriority()
- getpriority() and setpriority()
  - Act on the base priorities of all processes in a given group
  - which: PRIO_PROCESS, PRIO_PGRP, PRIO_USER
  - who: the value of pid, pgrp, or uid field to be used for selecting the processes
  - niceval: the new base priority value: -20 - +19
- sched_getaffinity() and sched_setaffinity()
  - The bitmask of the CPUs that are allowed to run the process

System Calls Related to Real-Time Processes

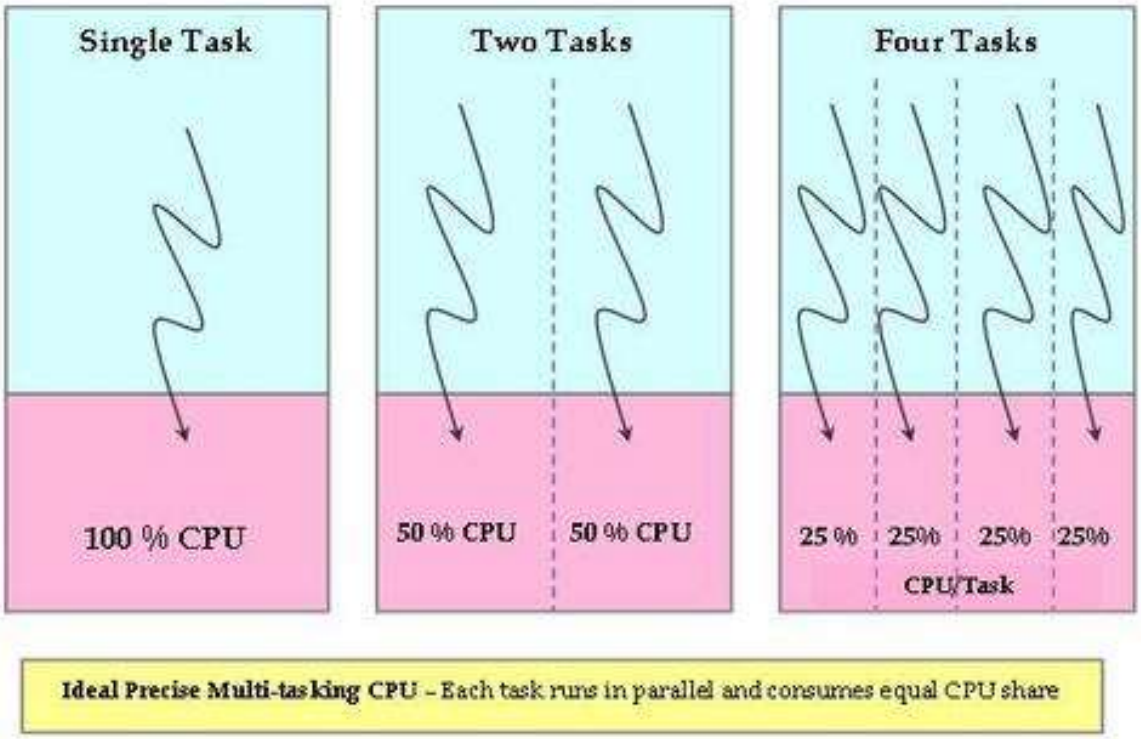
- sched_getscheduler(), sched_setscheduler()
  - Queries/sets the scheduling policy
- sched_getparam(), sched_setparam()
  - Retrieves/sets the scheduling parameters
- sched_yield()
  - To relinquish the CPI voluntarily without being suspended
- sched_get_priority_min(), sched_get_priority_max()
  - To return the minimum/maximum real-time static priority
- sched_rr_get_interval()
  - To write into user space the round-robin time quantum for real-time process

3B. Process Management -- Scheduling

Example: Completely Fair Scheduler (CFS)

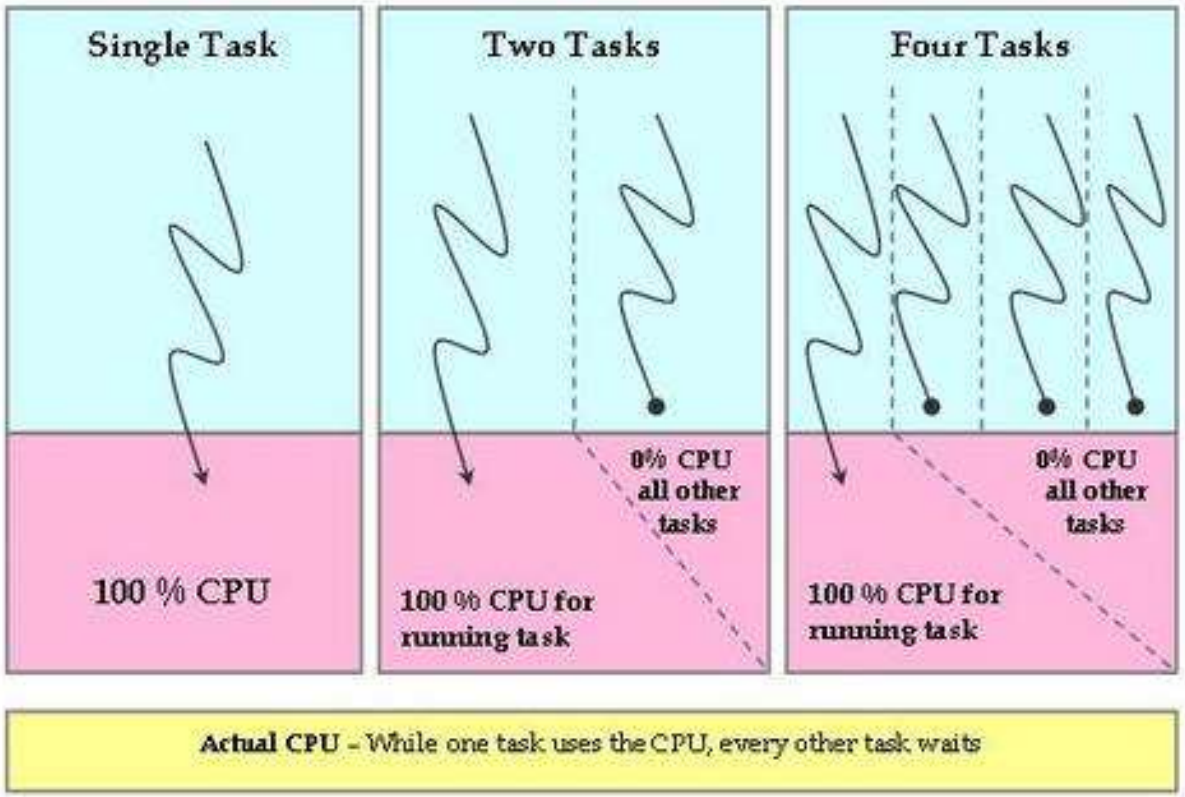
Motivation of Completely Fair Scheduler (CFS)

- Running task gets 100% usage of the CPU, all other tasks get 0% usage of the CPU



Motivation of Completely Fair Scheduler (CFS)

- Running task gets 100% usage of the CPU, all other tasks get 0% usage of the CPU



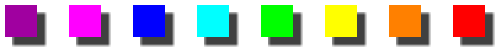
上海交通大学

Completely Fair Scheduler (CFS)

- New Scheduling Algorithm in Linux Kernel 2.6
- Design
  - The same **virtual runtime** for each task
  - Increasing the priority for **sleeping task**
- Implementation
  - Stores the records about the planned tasks in a **red-black tree**
  - pick efficiently the process that has used the least amount of **time (the leftmost node of the tree)**
  - The entry of the picked process is then removed from the tree, the spent execution time is updated and the entry is then returned to the tree where it normally takes some other location.

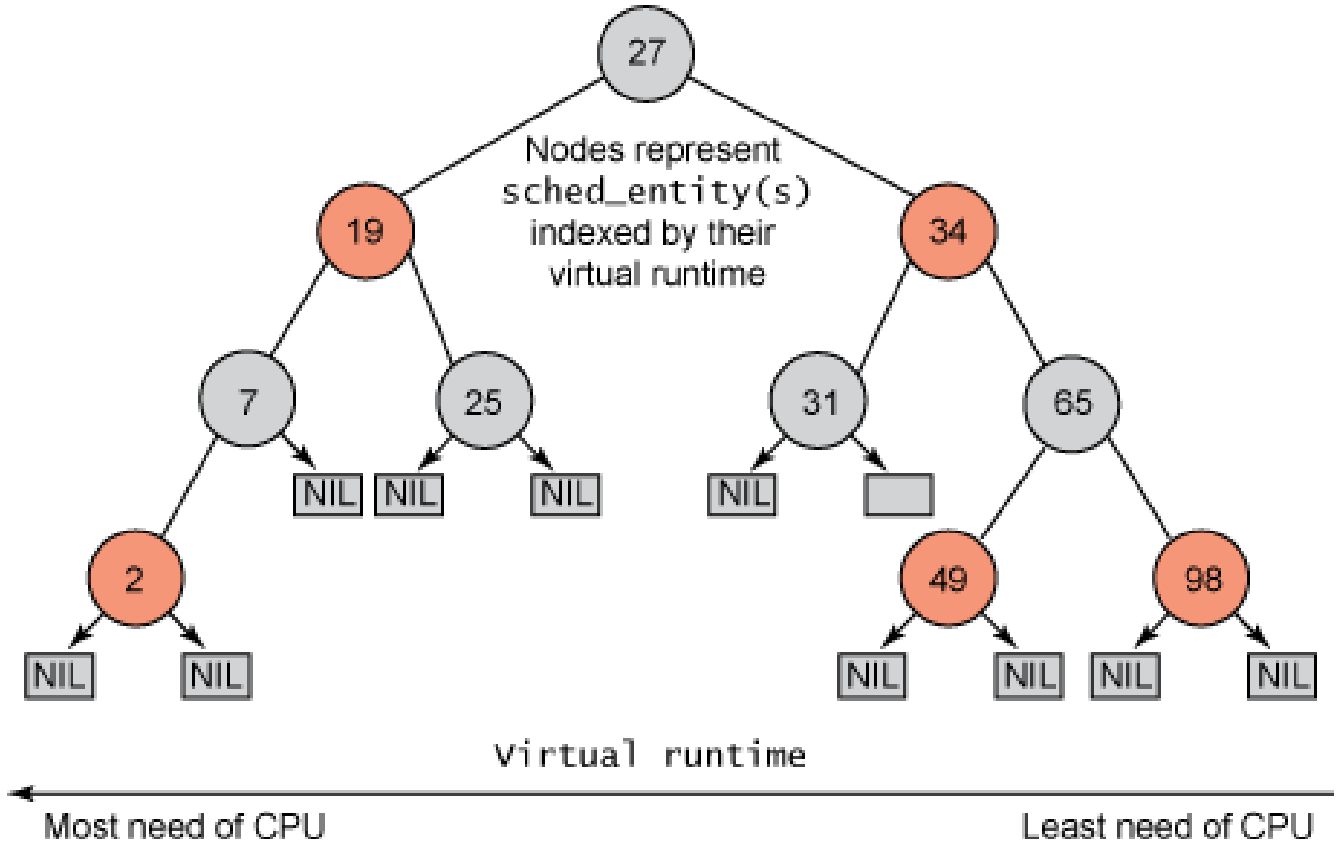
Red-Black Tree (1)

- The red-black tree is always balanced.
- The leftmost node pointer is always cached.
- The red-black tree is *O(log n)* in time for most operations, while the previous scheduler employed *O(1)*. (*O(log n)* behavior is measurably slower, but only marginally for very large task counts.)
- A red-black tree can be implemented with internal storage—that is, no external allocations are needed to maintain the data structure.



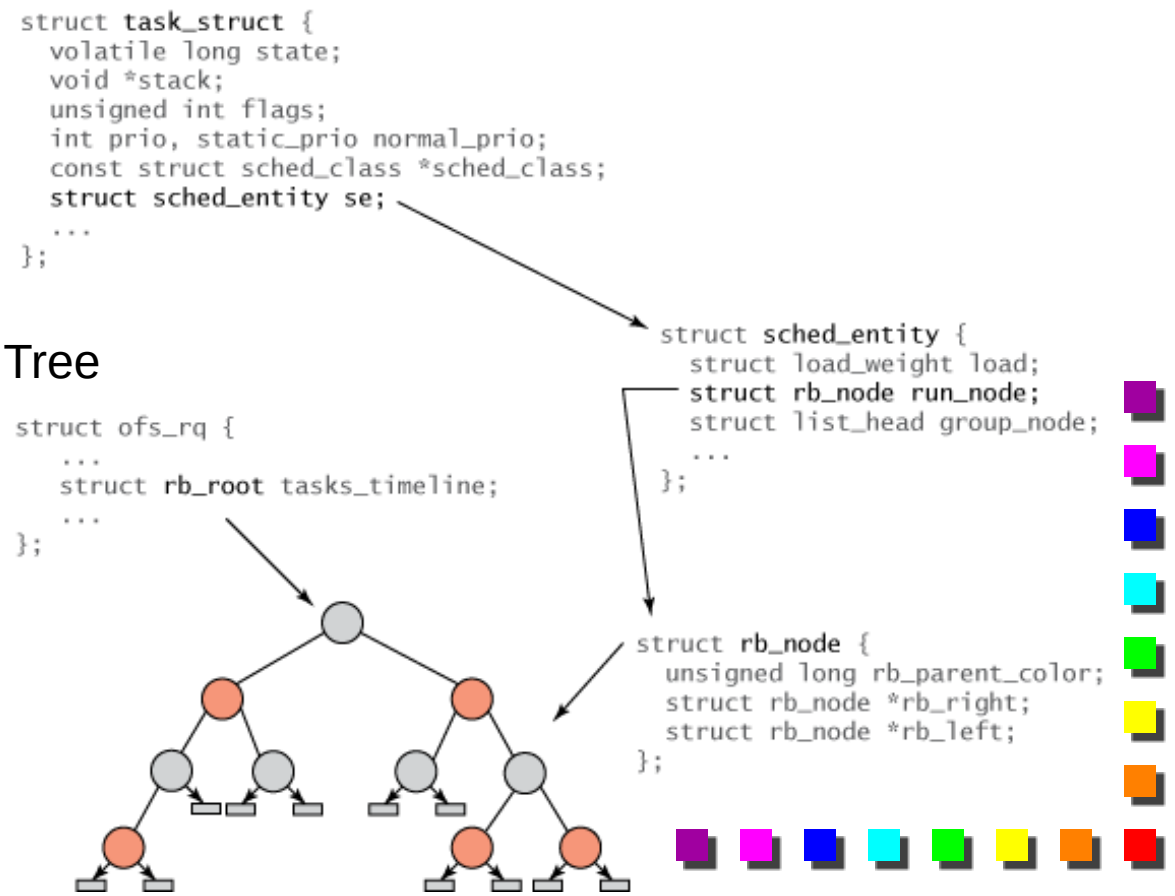
Red-Black Tree (2)

- Value = fair_clock − wait_runtime + nice (smaller value→higher priority)



Data Structures of CFS

- Task (task_struct)
  - ./linux/include/linux/sched.h
  - Root in Red-Black Tree (ofs_rq)
  - ./kernel/sched.c
  - sched_entity
  - weight
  - vruntime
  - Node in Red-Black Tree (rb_node)
  - ./kernel/sched.c
  - CFS Algorithm
  - ./kernel/sched.c



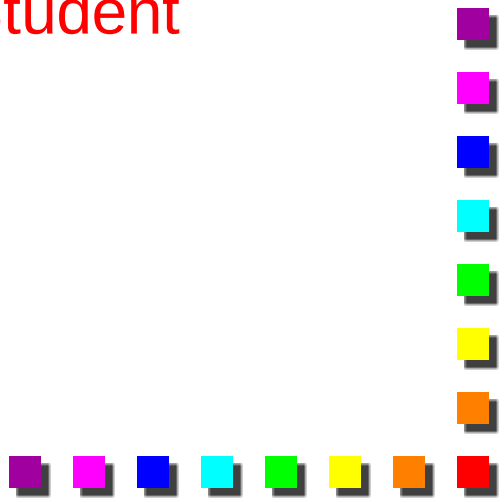
Project 2B:
Process Management

上海交通大學

Process: schedule in times

- **Add ctx, a new member to task_struct to record the schedule in times of the process;**
  - **When a task is scheduled in to run on a CPU, increase ctx of the process;**
  - **Export ctx under /proc/XXX/ctx;**

- More detailed information is shown in Student Experimental Handbook.



Tips

- Modify task_struct, add “int ctx”
- Initialization ctx
  - fork(), exec() → ?
- Schedule ()
  - ctx++
- Output ctx
  - Add a file in /proc/