



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

3



Priority Structure

- Real-time processes have the top 99 priority levels.
- Non-real-time processes have levels 100-139



Two Fundamental Resource Sharing Mechanisms

- ?
- ?



Two Fundamental Resource Sharing Mechanisms

- Prioritization
- Resource partitioning



SCHED_FIFO

- 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

- 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

- Used for non-real-time processes
- Complex heuristic to balance the needs of I/O and CPU centric applications



Process Priority & Timeslice Recalculation

- Static priority
 - A "nice" value
 - Inherited from the parent process
 - Set up by user
- 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

10



Heuristics

```
if (static priority < 120)
  Quantum = 20 (140 - static priority)
else
  Quantum = 5 (140 - static priority)
```

(in ms)



Heuristics

bonus = avg. sleep time mod 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 awoken
- 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()`)

13

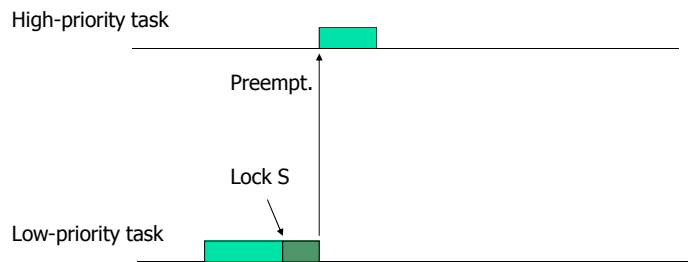


Advanced Topic: Priority Inversion

- What if a higher-priority process needs a resource locked by a lower-priority process?
 - How long will the higher priority process have to wait for lower-priority execution?

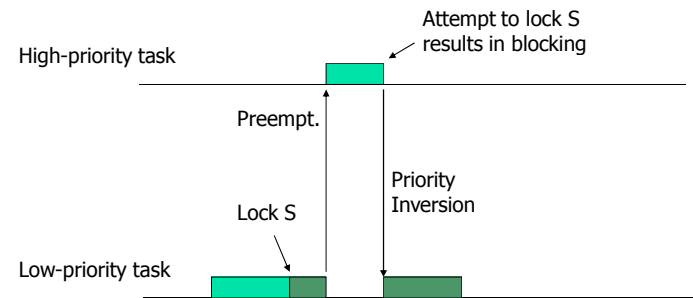
Priority Inversion

- Locks and priorities may be at odds. Locking results in priority inversion



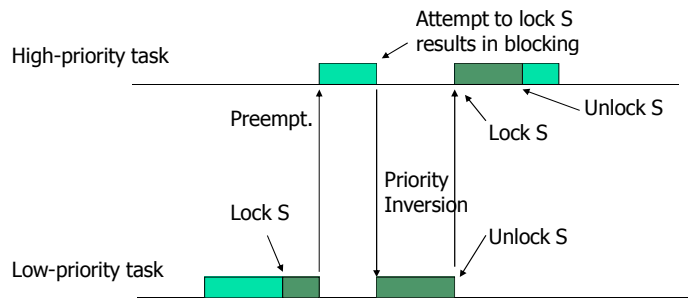
Priority Inversion

- Locks and priorities may be at odds. Locking results in priority inversion



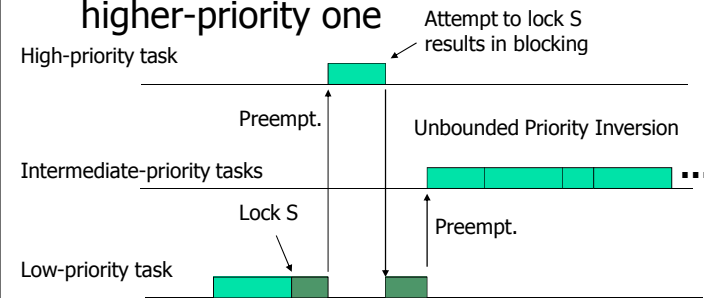
Priority Inversion

- How to account for priority inversion?



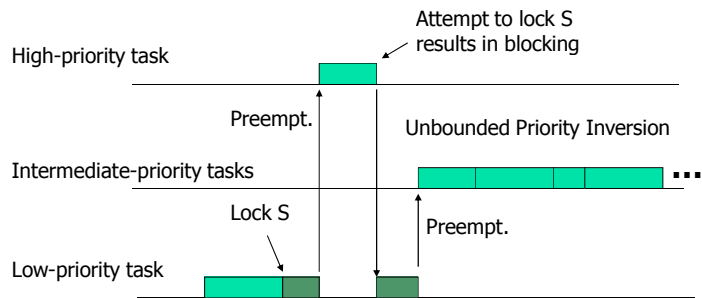
Unbounded Priority Inversion

- Consider the case below: a series of intermediate priority tasks is delaying a higher-priority one



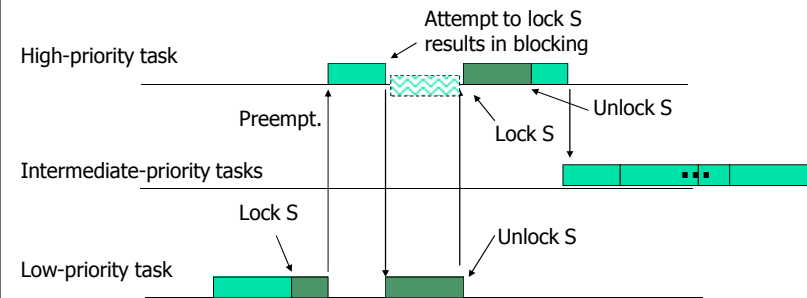
Unbounded Priority Inversion

- How to prevent unbounded priority inversion?



Priority Inheritance Protocol

- Let a task inherit the priority of any higher-priority task it is blocking

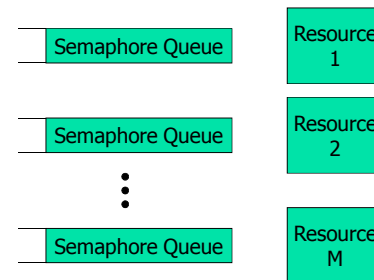


Priority Inheritance Protocol

- Question: What is the longest time a task can wait for lower-priority tasks?
 - Let there be N tasks and M locks
 - Let the largest critical section of task i be of length B_i
- Answer: ?

Computing the Maximum Priority Inversion Time

- Consider the instant when a high-priority task that arrives.
 - What is the most it can wait for lower priority ones?

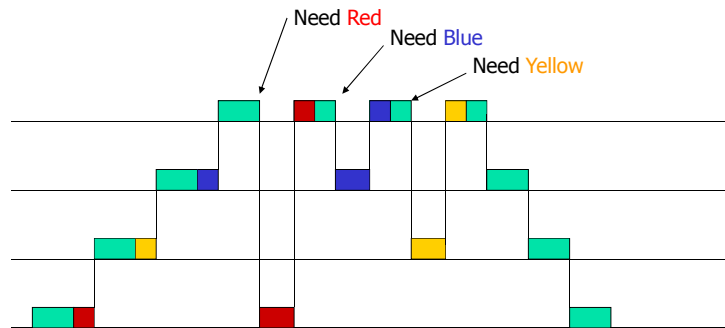


If I am a task, priority inversion occurs when

- Lower priority task holds a resource I need (direct blocking)
- Lower priority task inherits a higher priority than me because it holds a resource the higher-priority task needs (push-through blocking)

Maximum Blocking Time

Priority Inheritance Protocol



Priority Ceiling Protocol

- Definition: The priority ceiling of a semaphore is the highest priority of any task that can lock it
- A task that requests a lock R_k is denied if its priority is not higher than the highest priority ceiling of all currently locked semaphores (say it belongs to semaphore R_h)
 - The task is said to be blocked by the task holding lock R_h
- A task inherits the priority of the top higher-priority task it is blocking



MP2 Bonus Points

- Implement the priority ceiling protocol in your thread and lock library (due separately from MP2) – details to follow.