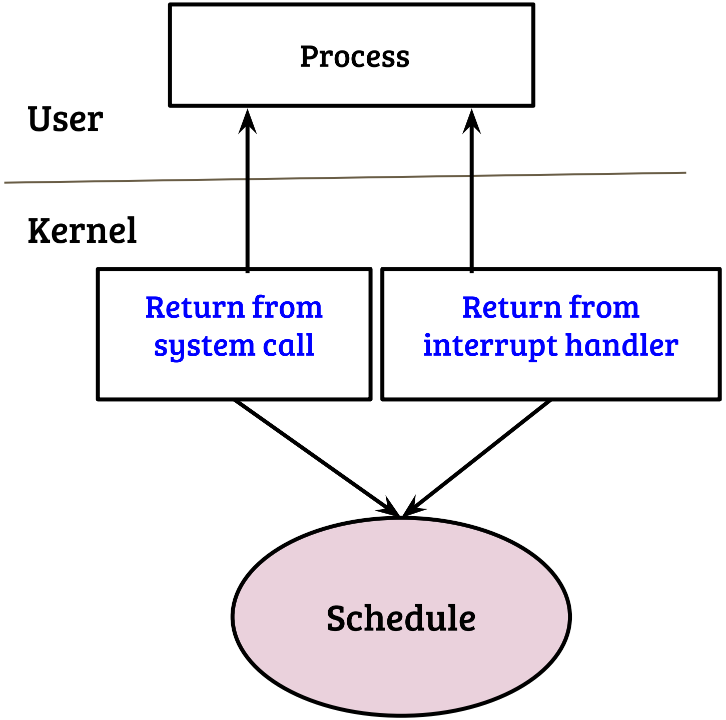
Process management

Scheduling

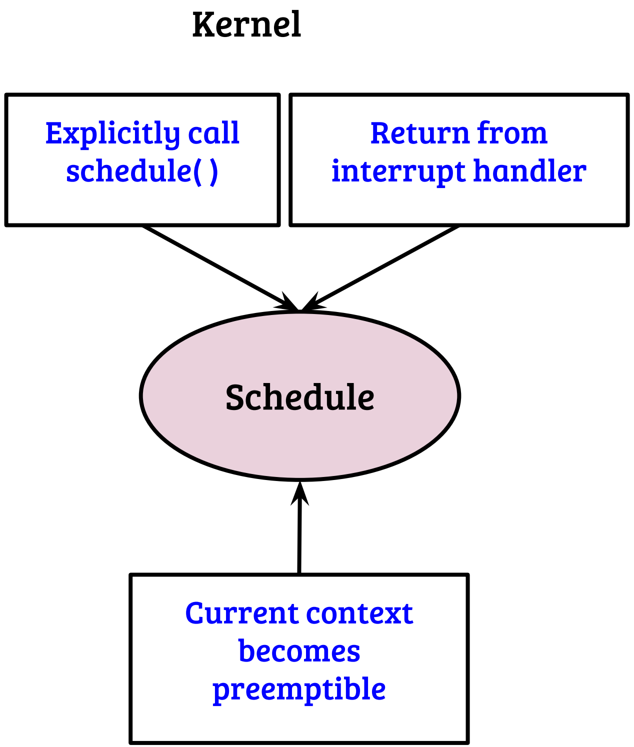
Points of scheduler invocation (recap)



- ➔ Timer interrupts to ensure OS control
- ➔ Return from interrupts, why?
  - ◆ Responsive system (how?)

- ➔ Refer exit_to_usermode_loop( ) in arch/x86/entry/common.c, executed eventually from arch/x86/entry/entry_64.S

Points of scheduler invocation (recap)



- ➔ Why user preemption is not sufficient?

- ➔ Explicit call to schedule scenarios?
- ➔ Avoid lock holder preemption
- ➔ Refer preempt_schedule_irq ( ) in kernel/sched/core.c executed from arch/x86/entry/entry_64.S

Scheduling objectives

- Meet scheduling need of
  - Real-time processes
  - Interactive processes
  - Batch processes
- Fairness
- Throughput, responsiveness
- Optimize multiple objectives, sometimes conflicting
- General strategy
  - Provide user defined scheduling policies for “precise control”
  - Define priorities (static)
  - But users may be biased, uninformed? So, let the good sense prevail (in kernel).

User control: scheduling classes

REAL-TIME APPLICATIONS

SCHED_FIFO | SCHED_RR

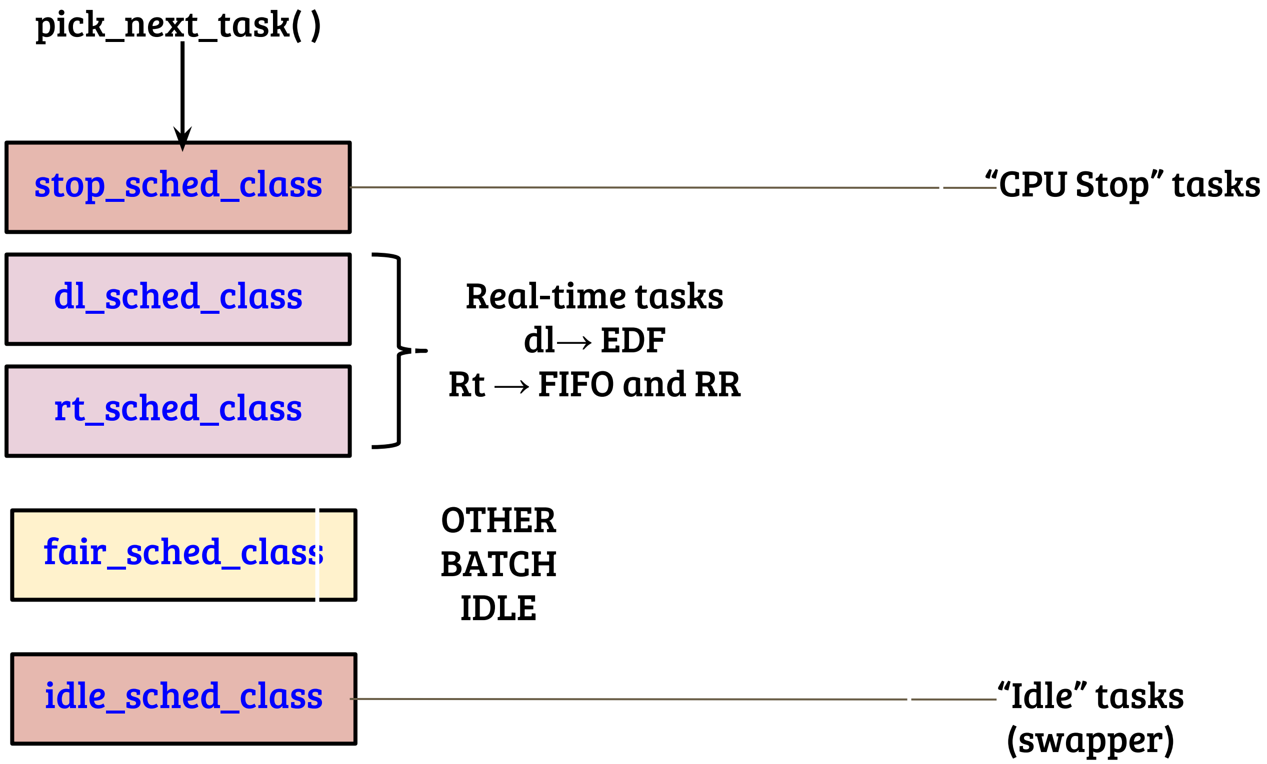
- ➔ Always higher priority than normal processes
- ➔ Priority value: 1 to 99
- ➔ FIFO: run to completion
- ➔ RR: Round robin within a priority-level

NORMAL APPLICATIONS

SCHED_OTHER | SCHED_BATCH | SCHED_IDLE

- ➔ SCHED_OTHER is default
- ➔ SCHED_BATCH: Assume CPU bound while calculating dynamic priorities
- ➔ SCHED_IDLE are for low priority jobs

Scheduling classes (v-4.12.3)



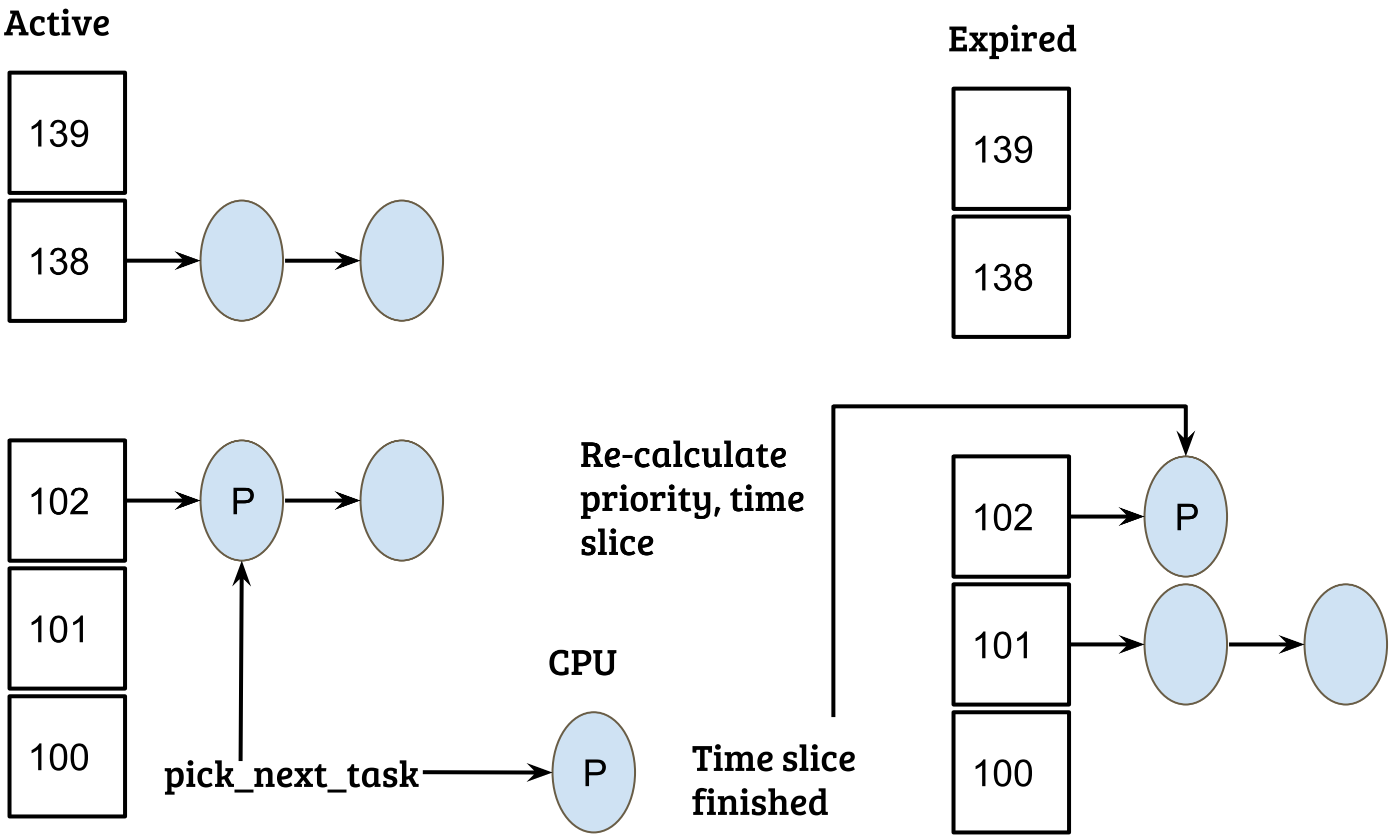
Priority of non-RT processes

- Static priority, Unix “niceness”
  - -20 to 19
  - How “nice” is this process to others?
  - Low value→ not nice to others→ higher priority to myself
- Dynamic priority
  - Interactive tasks, How users know?
  - How to determine?
  - Can be used to calculate time slice

Scheduling legacy: O(1) scheduler

- ➔ Two sets of run queues, one queue for each priority level
  - Active
  - Expired
- ➔ Total 40 dynamic priority levels
  - 40 lists in active and expired
- ➔ Select the first task from the highest priority queue
  - Move it to inactive after its time slice is finished
- ➔ Swap the lists when active is empty

O(1) scheduler: example



O(1) scheduler: Details

- Blocked tasks not part of active or expired
- Time slice calculation
  - proportional to priority
- Interactive tasks vs. CPU bound tasks
  - Dynamic priority = MAX(100, min(static priority - bonus + 5, 139))
  - 0 <= bonus <=10
  - Value of bonus determined by “wait time”
- Issues
  - Heuristic based
  - Can be tricked! how?

Completely Fair Scheduler (CFS)

- Idea: “I am the ideal, Catch me if you can!”
- If there are N tasks competing for CPU during T time units, each task should ideally get T/N CPU time
  - CFS is tries to maintain this basic fairness!
  - Favor the process to which the system is most unfair (so far)
  - But not very easy
    - What is T?
    - Sometimes a catchup game can lead to an elegant solution

CFS details

- A global virtual clock ticks every N real ticks
  - N is the number of processes
  - Represents the ideal CPU time
- Each process keeps track of its CPU usage ticks
- The smallest tick count task gets the CPU
- Issues
  - Data structure
  - Startup boost?
  - How to accommodate priorities?
  - What happens to interactive tasks?
  - Scale per-task CPU usage ticks to enforce priority, per-user fairness etc.

Scheduling: SMP

- One task should ideally run on a fixed core
  - Why?
  - Should there be rebalancing?
  - What if another CPU is idle?
- Cost vs. benefit
  - Better resource utilization
  - Initial penalty?
  - Will the process execute “slower” on another CPU?

NUMA awareness