Overview of the Challenges faced in implementing the Linux Scheduler

By,
Preeti U Murthy
Energy Management Team
Linux Technology Center,IBM

Agenda

- Listing and brief explanation of the requirements demanded and addressed by the Linux scheduler
- Testing and Profiling of the Linux scheduler

Increasing Requirements from the Linux Scheduler

1. Fast Algorithm - O(1) scheduling
2. Fast and Fair Algorithm - CFS scheduling
3. Scalable scheduling
4. Group scheduling
5. Real Time task scheduling
6. Power Aware Scheduling
7. Resource Monitored scheduling
8. Heterogeneous platform scheduling
9. Smarter Load Balancing

O(1) scheduling

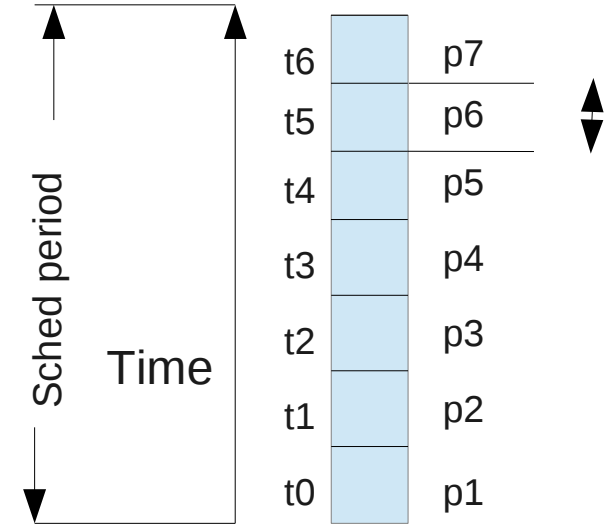
Requirement: Constant time scheduling

- Improvement over O(n) scheduler
- Real Time systems benefit from this because of deterministic execution times

CFS scheduling

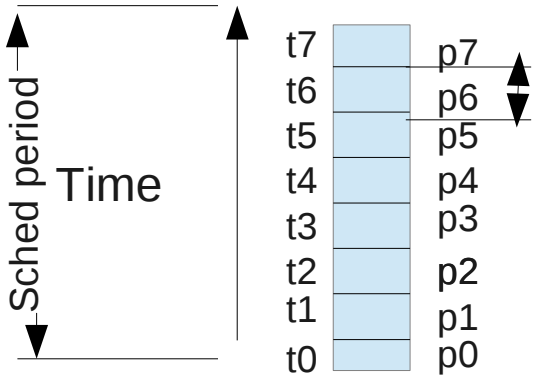
Requirement: The CPU bandwidth needs to be “fairly” distributed among tasks

- Data Structure used is the Red Black Tree
- The tasks are sorted out in this tree in the increasing order of CPU bandwidth received
- A new task or a woken up task is positioned at the left most end of the tree
- The CPU bandwidth consumed by a task is calculated based on its weight and is called the virtual run-time



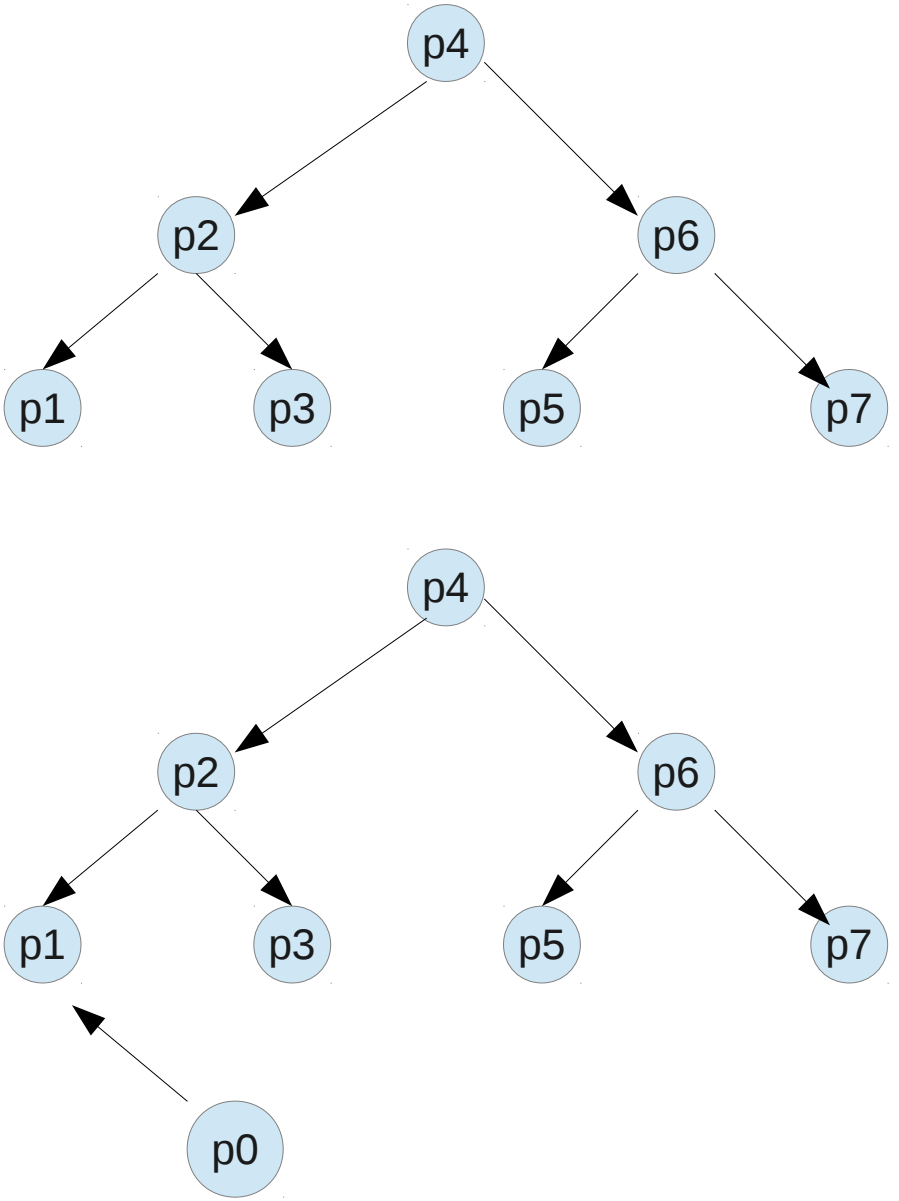
Max allowable vruntime for p6

Tasks scheduled on cpu as time progresses



Max allowable vruntime for p6

Tasks scheduled on cpu as time progresses



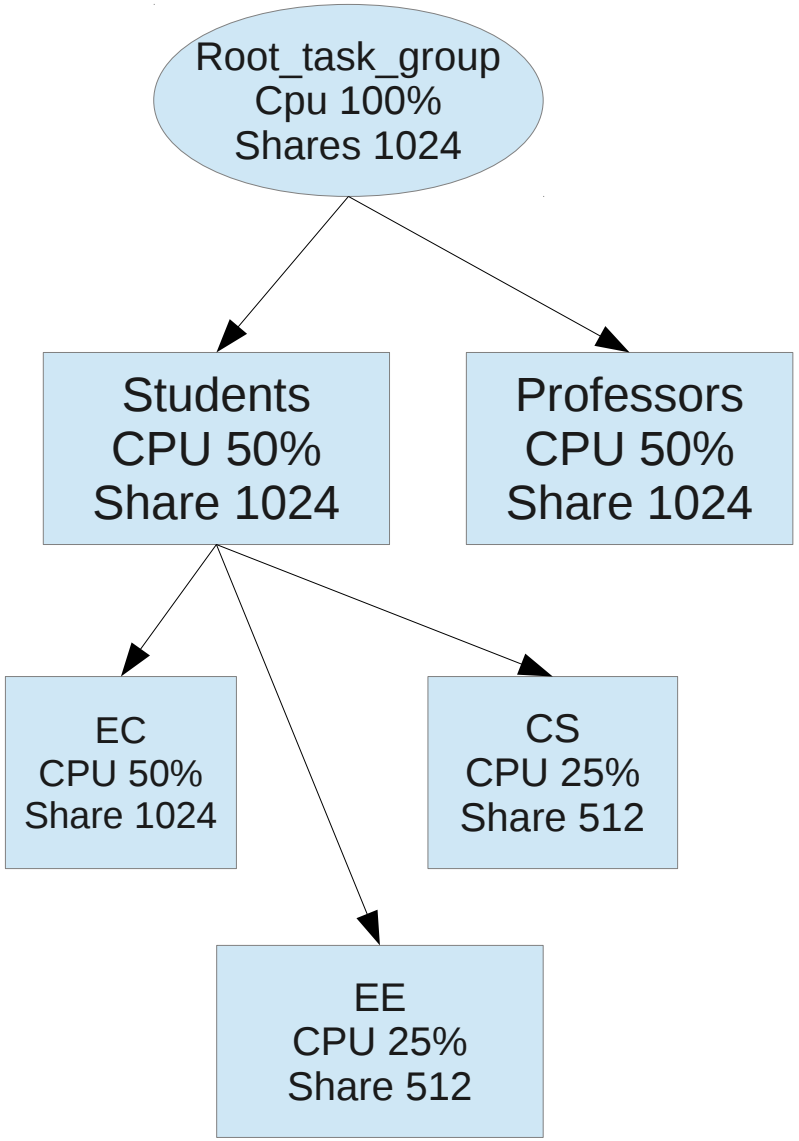
New Task

CFS scheduling on CPU

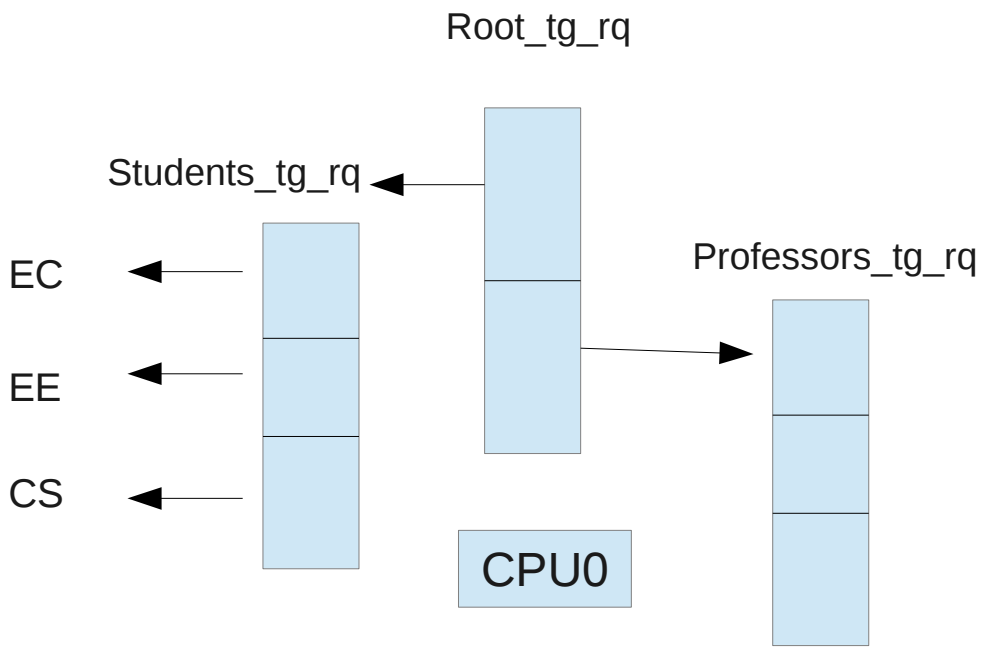
Group Scheduling

Requirement: First fair among groups,then fair among tasks within these groups

- Every group has a parent task group,children task groups and sibling task groups
- Every group has a run-queue and a sched-entity associated with every CPU



Group Hierarchy Example in a University Server



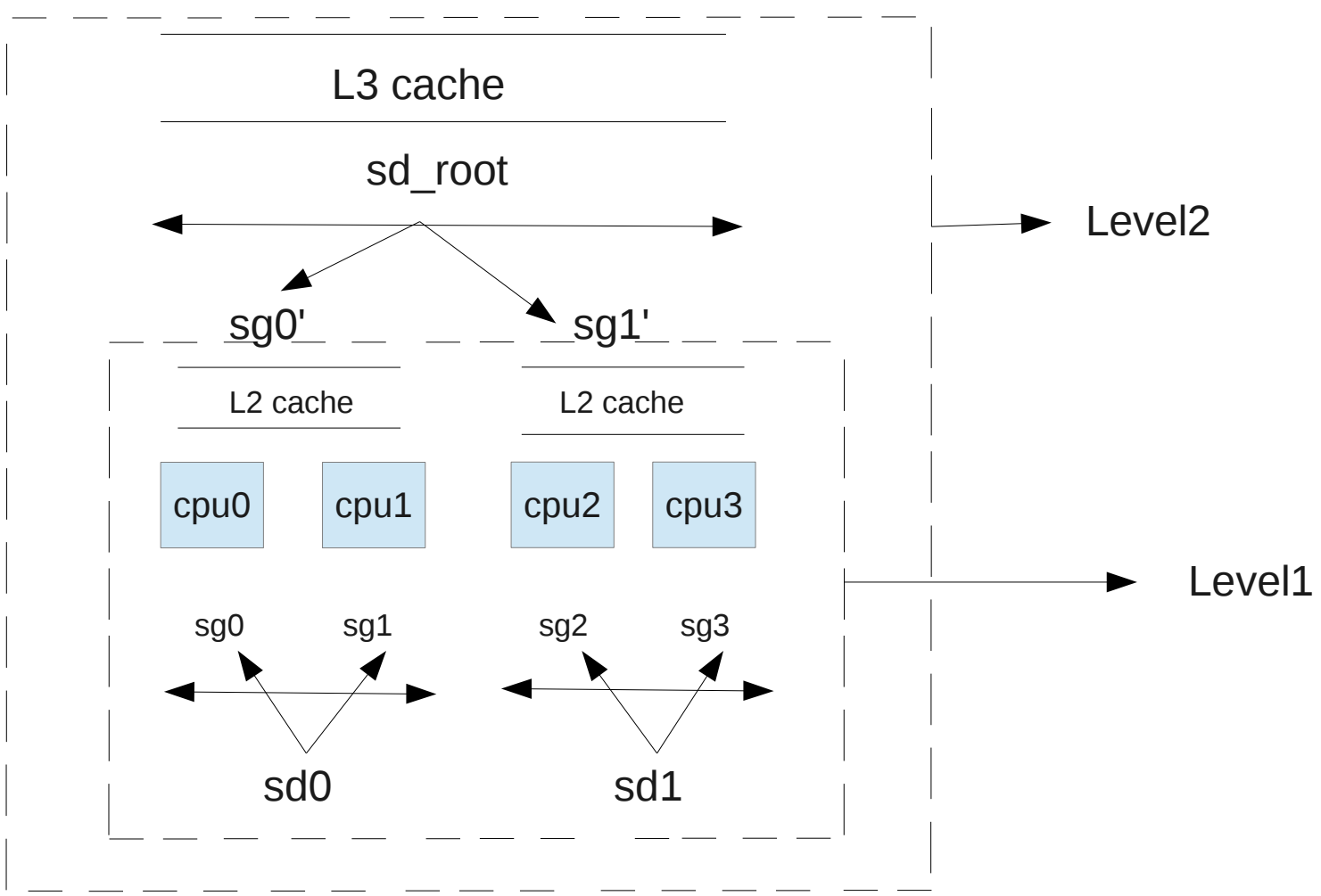
Run-queue Hierarchy of groups

Use case and Run-queue Layout of Group Scheduling

Scalable scheduling

Requirement: As CPU package increases in size and complexity,performance of scheduler must not decline.

- CPUs being divided into scheduling domains and scheduling groups
- Every scheduling domain consists of non-overlapping scheduling groups
- Every CPU is associated with its own scheduling domain and scheduling group at each domain
- Every scheduling domain is associated with a child and parent scheduling domain
- The scheduling groups within a scheduling domain are arranged as a linked list
- Bottom up and top down movement of this hierarchy querying idle and busy cpus depending on the situation



cpu0->sd = sd0; sd0->parent = sd_root
cpu0->local_group = sg0 @ level1
cpu0->local_group = sg0' @ level2

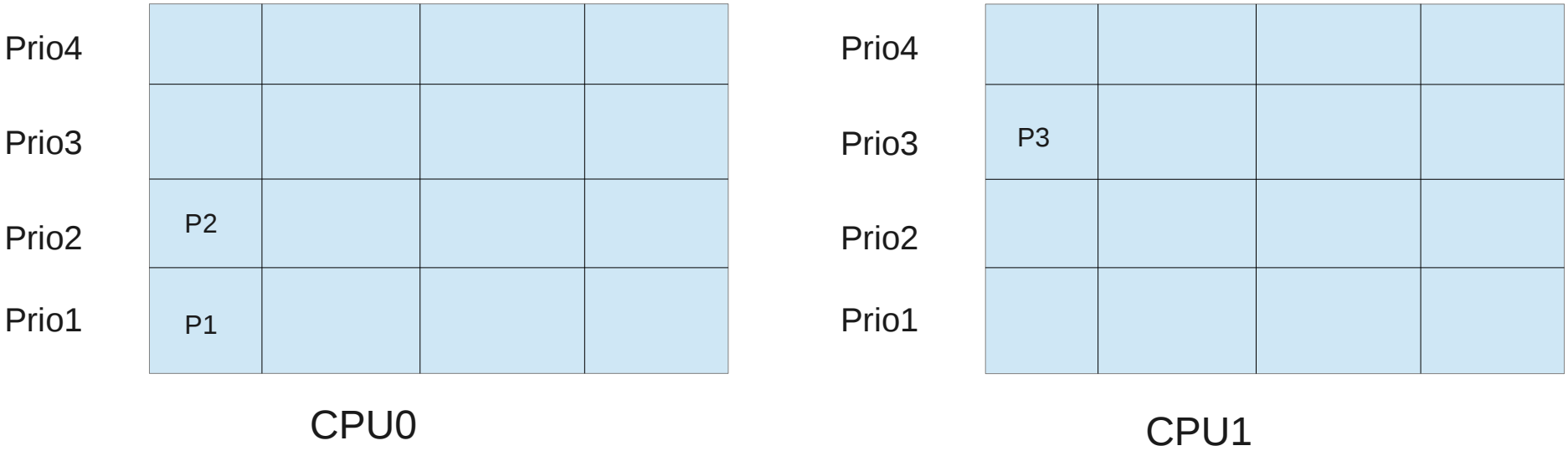
sd0->groups = sg0->sg1
sd_root->groups = sg0'->sg1'

Scheduling Hierarchy in a 4 core 1 socket machine

Real Time Scheduling

Requirement: Predictability of response time

- Different queues for different priorities on each CPU
- Exhaust the queues in order of their priorities
- SCHED_FIFO and SCHED_RR policies
- Push and pull operations on tasks between run-queues
- Run-queue gets overloaded when it has more than 1 rt task and they can be migrated



RT run-queues on CPU

1. Push Operation: P2 be pushed on CPU1 since it can be scheduled on CPU1 before P3,but cannot be scheduled on CPU0 before P1.

2. Pull Operation: P2 can be pulled by CPU1 before scheduling P3 onto itself.

Power Aware Scheduling

Requirement: Optimize scheduler for performance and energy efficiency

- Optimize scheduler for power efficiency
- Metric to measure the heaviness/lightness of tasks
- Mapping of scheduler domains to power domains
- Balance policy vs power policy vs performance policy

Resource Monitored Scheduling

Requirement: Manage the amount of CPU allotted to tasks.

- Allocating CPU Shares to task groups
- Allocating CPU Bandwidth to task groups
- Limiting CPU Bandwidth of task groups
- Limiting Concurrency of tasks in task groups

Heterogeneous Platform Scheduling

Requirement: Map nature of the task to nature of the CPU

- ARM's Big.Little CPUs

Smarter Load Balancing

Requirement: Use CPUs optimally

- Identify Light weight tasks
- Wake up idle CPUs only if necessary
- Offload timers,interrupts away from idle CPUs
- Hardware and Software Buddy CPUs

Testing the Performance of the scheduler

- Run different kinds of workloads/benchmarks
- Run different kinds of workloads on different platforms
- Performance should not degrade
- Benchmarks:
  - Hackbench: Tests effect on latency of scheduler operations
  - Kernbench: Tests effect on throughput of scheduler operations
  - Sysbench: Simulates transactional workload
  - AIM: Simulates bursty workloads
  - Ebizzy: Simulates different CPU utilization workloads

Profiling of the scheduler

- /proc statistics to measure number of migrations, latency of the scheduler,the state of the process,wake ups,sleeps of processes,other load balancing metrics.
- Perf sched :
  - Record scheduler events
  - Report details of the scheduler events
  - Show migration of events
  - Latency details of tasks
  - Replay the exact behavior of workload
- Use scripting to parse trace files for extended information

Learning Resources

- IBM Developer Works articles
- LWN articles
- Scheduling chapter in “Professional Linux Kernel Architecture” Book by Wolfgang Mauerer
- Linux journal article on Real Time Linux Kernel scheduler by Ankita Garg
- Documentation/scheduler/*
- Code resides in kernel/sched/* and include/linux/sched.h

Thank you