Project 3

SPL, SNU

Project 3 Overview

- Design and implement WRR (Weighted Round-Robin) scheduler working in your ARTIK
  - Define and Implement a new scheduler
    - Implement load balancing mechanism
  - Make the scheduler class as the default scheduler for init process
    - For both systemd & kthreadd
  - Examine the scheduler performance with Trial
  - Improve the WRR scheduler(for 4 members)
    - Open question
    - Extra.md

WRR Scheduler

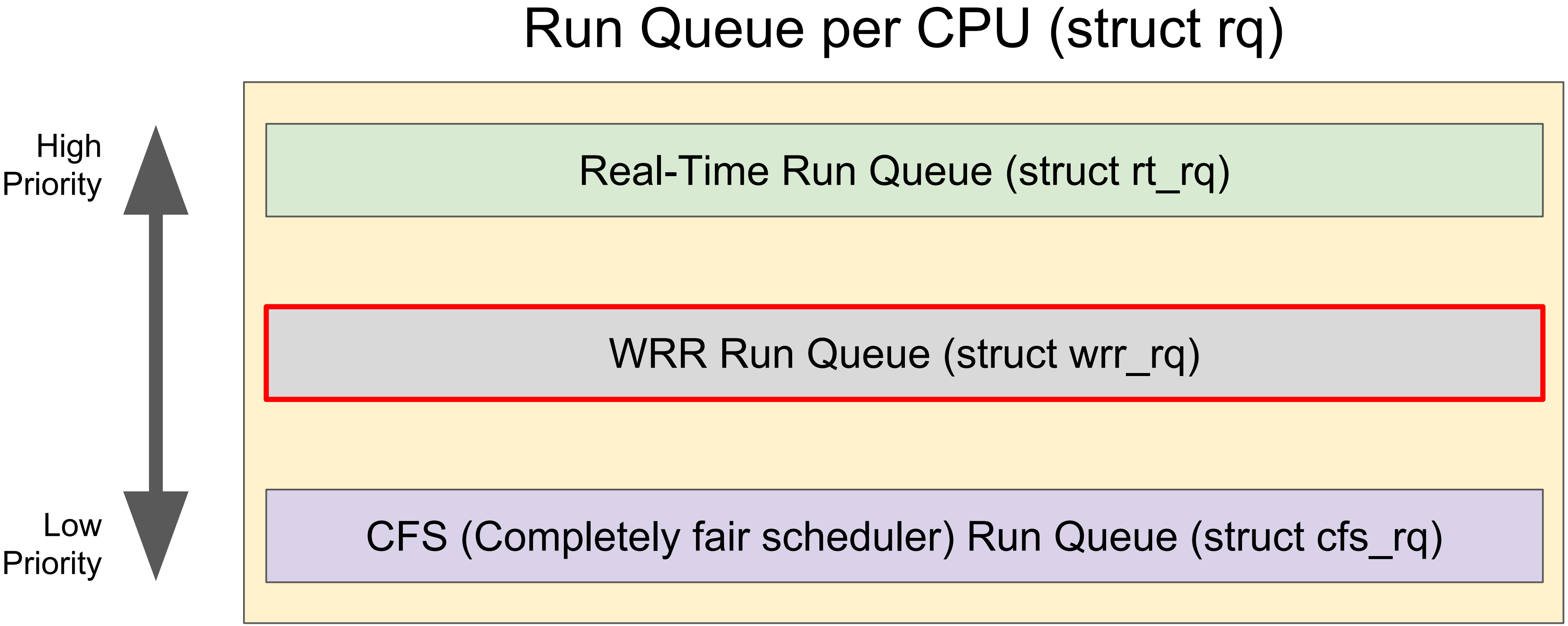
Linux Scheduler Basic

- Multi-level scheduling
    - Real-time tasks has priority over other tasks
- Real-time tasks are scheduled in FCFS or RR fashion
- Other tasks are scheduled by CFS (Completely-Fair Scheduler) algorithm
- Each CPU maintains separate run queues for tasks
    - To prevent contention while accessing run queue

WRR Scheduler

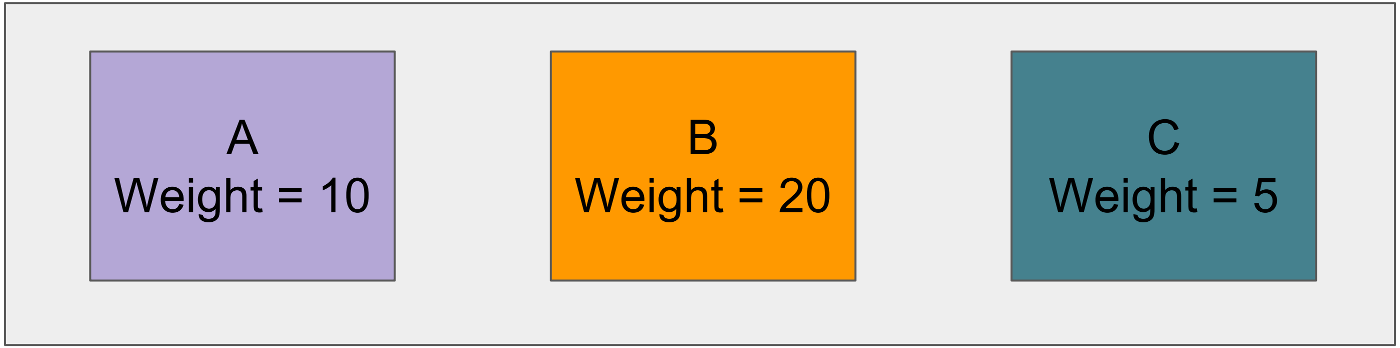
- Weighted Round-Robin Scheduler
- Tasks are executed in round-robin fashion, but gets different time slices according to their weights
  - Default weight is 10
  - Time slice = Weight * 10ms
- WRR has higher priority than CFS, but lower priority than RT (Real-Time) scheduler
- Load balancing should be implemented

Multi-level Run Queue with WRR


Run Queue per CPU (struct rq)
High Priority
Real-Time Run Queue (struct rt_rq)
WRR Run Queue (struct wrr_rq)
CFS (Completely fair scheduler) Run Queue (struct cfs_rq)
Low Priority

WRR Scheduling Example

Three tasks are currently in WRR run queue



A
Weight = 10

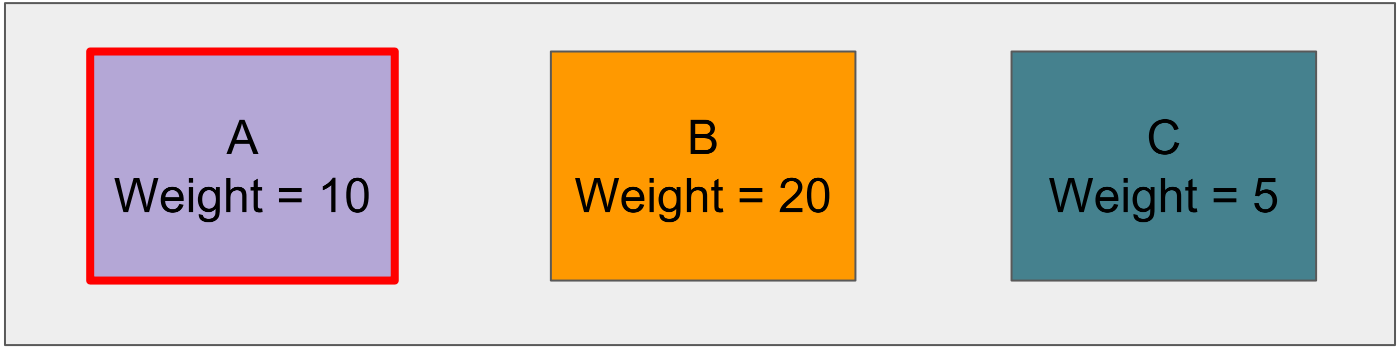
B
Weight = 20

C
Weight = 5

WRR Scheduling Example

0ms passed

A starts running first



A
Weight = 10

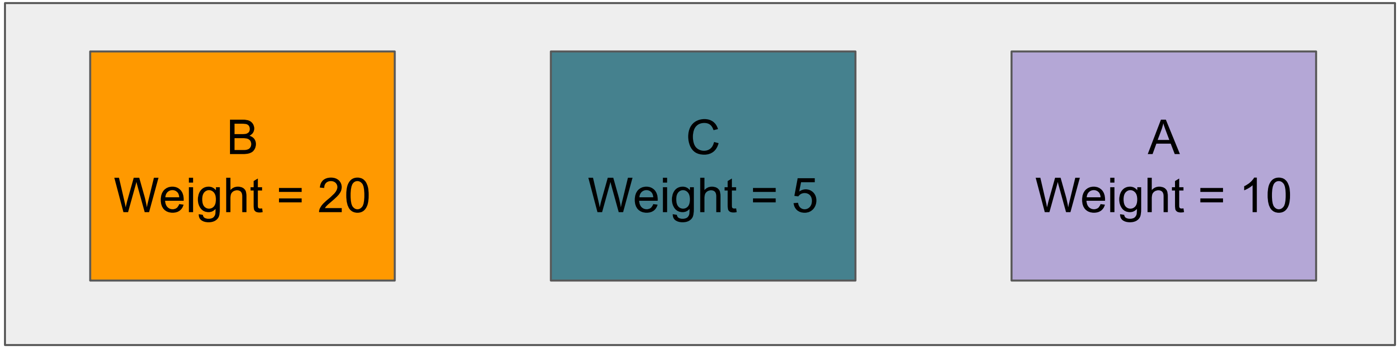
B
Weight = 20

C
Weight = 5

WRR Scheduling Example

100ms passed

A stopped, and moved to the tail of the run queue because the task is not finished



B
Weight = 20

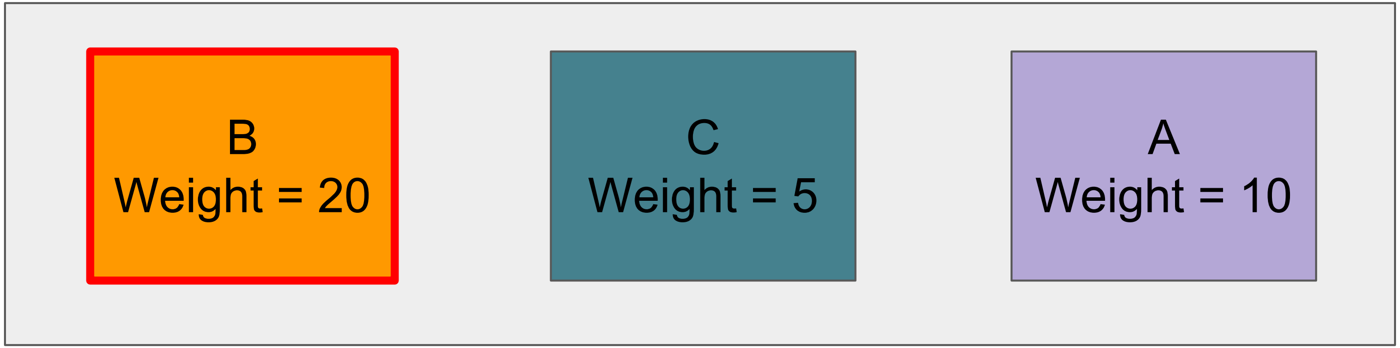
C
Weight = 5

A
Weight = 10

WRR Scheduling Example

100ms passed

The next task (B) starts running



B
Weight = 20

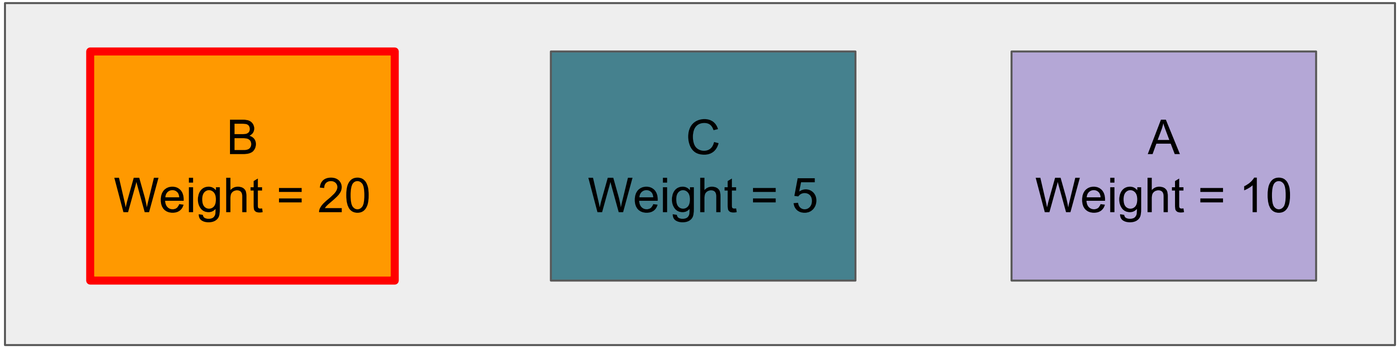
C
Weight = 5

A
Weight = 10

WRR Scheduling Example

200ms passed

B is still running and not stopped, because it got a 200ms time slice



B
Weight = 20

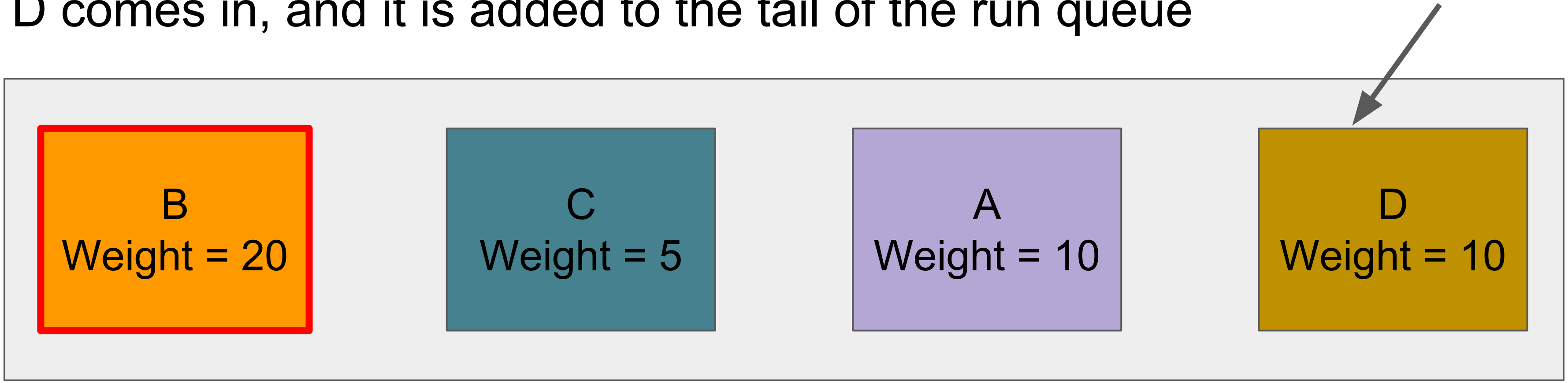
C
Weight = 5

A
Weight = 10

WRR Scheduling Example

250ms passed

D comes in, and it is added to the tail of the run queue



Newly added!

B
Weight = 20

C
Weight = 5

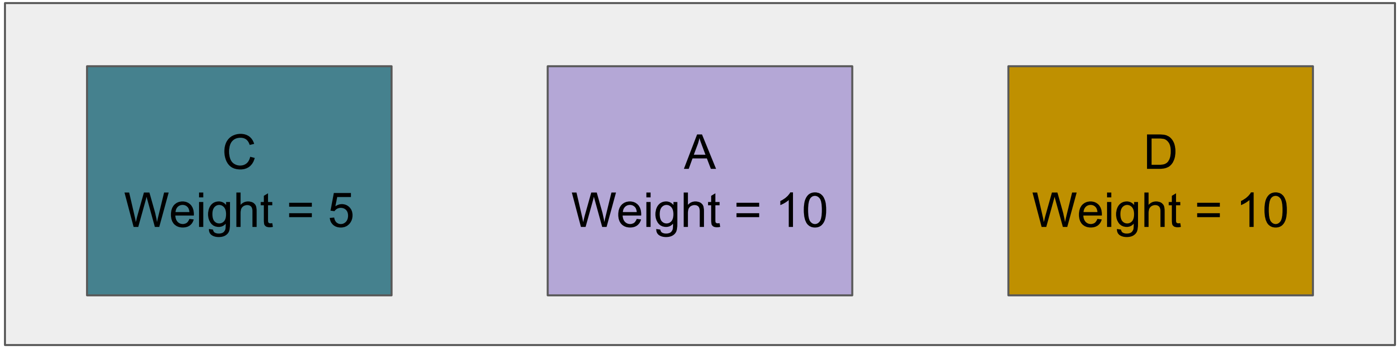
A
Weight = 10

D
Weight = 10

WRR Scheduling Example

280ms passed

B has been finished, and removed from the run queue because there is no more work left B



C
Weight = 5

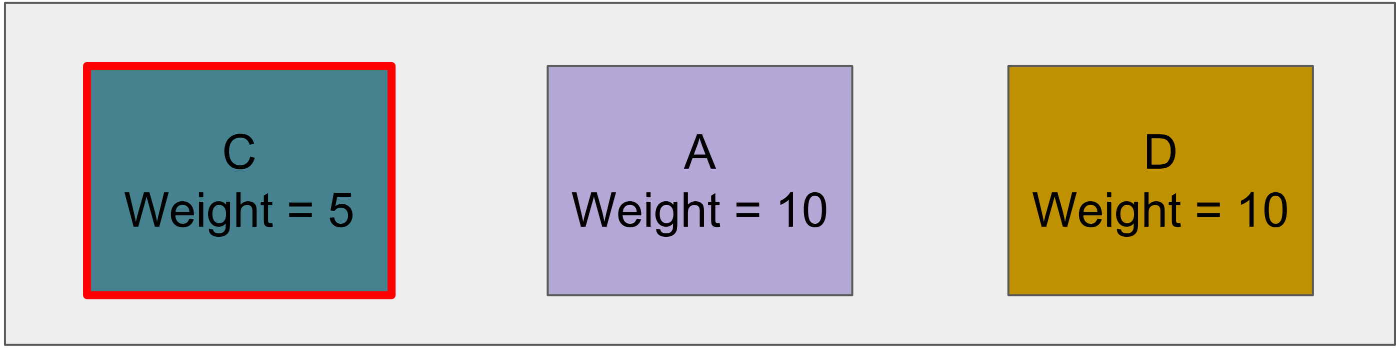
A
Weight = 10

D
Weight = 10

WRR Scheduling Example

280ms passed

C starts running



C
Weight = 5

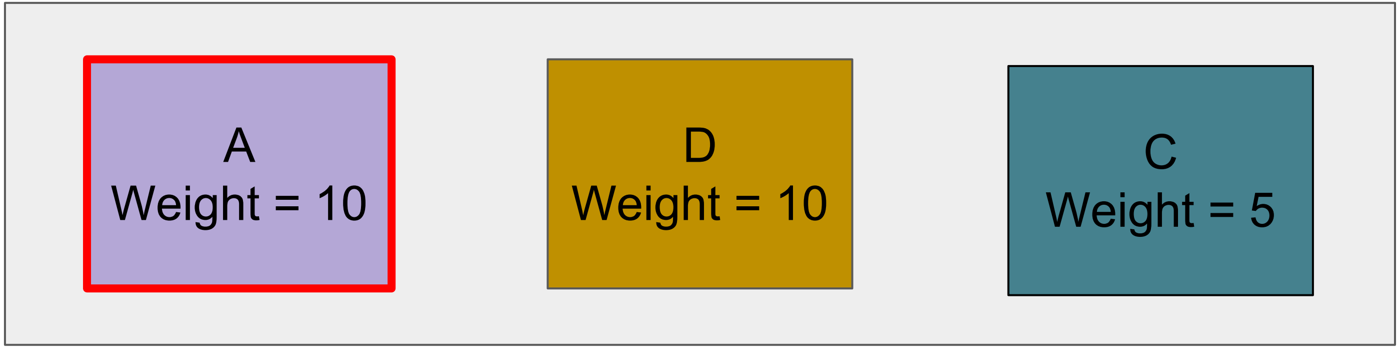
A
Weight = 10

D
Weight = 10

WRR Scheduling Example

330ms passed

C has been stopped and moved to the tail. A starts running again



Load balancing

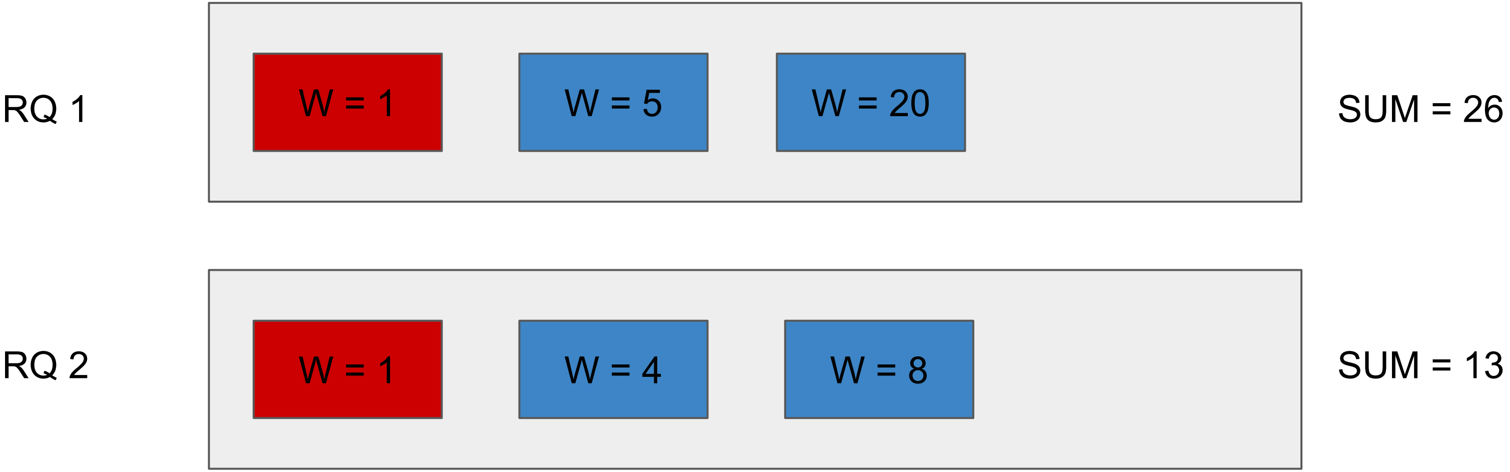
- Balance loads among each CPU’s run queue
- Make sure that it only works when more than one CPUs are active
    - When there is no heavy task, only one CPU is active with high probability!
    - CPU hotplug
    - for_each_online_cpu(cpu)
- Should be done every 2000ms

Load balancing algorithm

- Pick two run queues with the minimum weight sum and the maximum weight sum, respectively.
    - Let’s call them RQ_MIN and RQ_MAX
- Pick a task with the biggest weight among tasks meeting these conditions:
    - The picked task should be able to be migrated to RQ_MIN
    - Migration should not cause weight of RQ_MIN to become **bigger than or equal** to RQ_MAX
    - The currently running task cannot be picked
- Migrate the task if an eligible task exists (there may be no eligible tasks)

Load Balancing Example

Migrate a task from RQ 1 to RQ 2

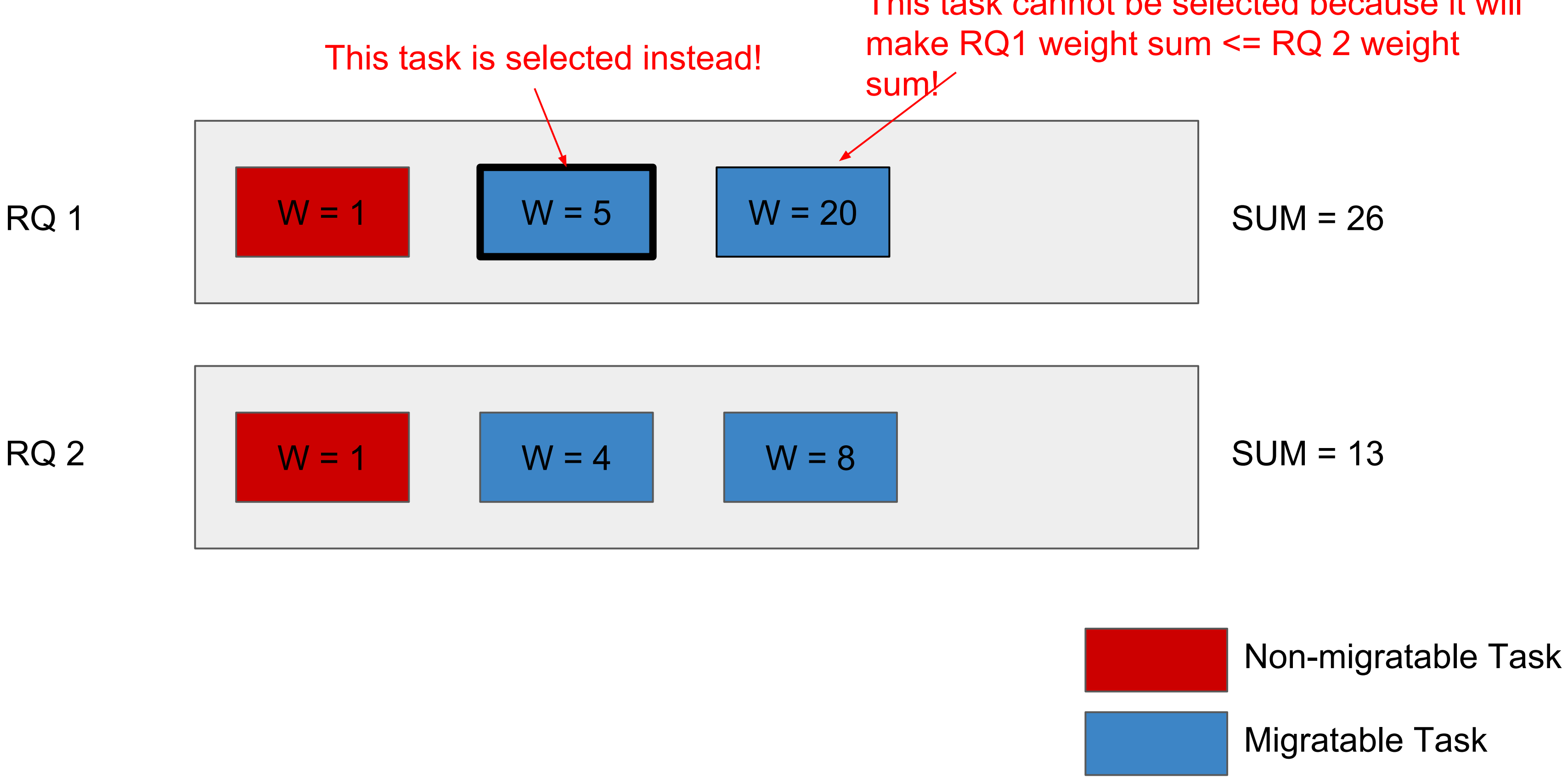


RQ 1	🟥 W = 1	🟦 W = 5	🟦 W = 20	SUM = 26
RQ 2	🟥 W = 1	🟦 W = 4	🟦 W = 8	SUM = 13

🟥 Non-Migratable Task

🟦 Migratable Task

Load Balancing Example



This task is selected instead!

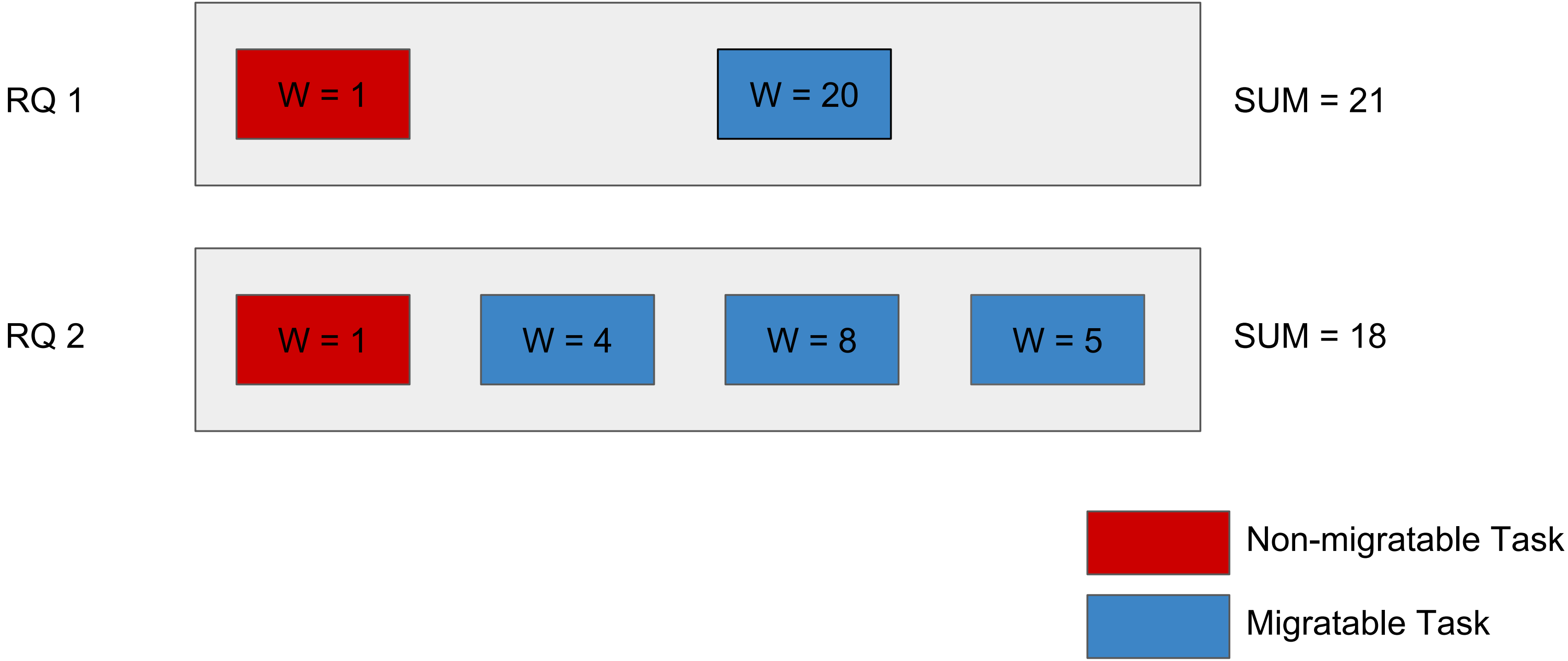
This task cannot be selected because it will make RQ1 weight sum <= RQ 2 weight sum!

🟥 Non-migratable Task

🟦 Migratable Task

Load Balancing Example

After migration...



RQ 1: W = 1, W = 20 — SUM = 21

RQ 2: W = 1, W = 4, W = 8, W = 5 — SUM = 18

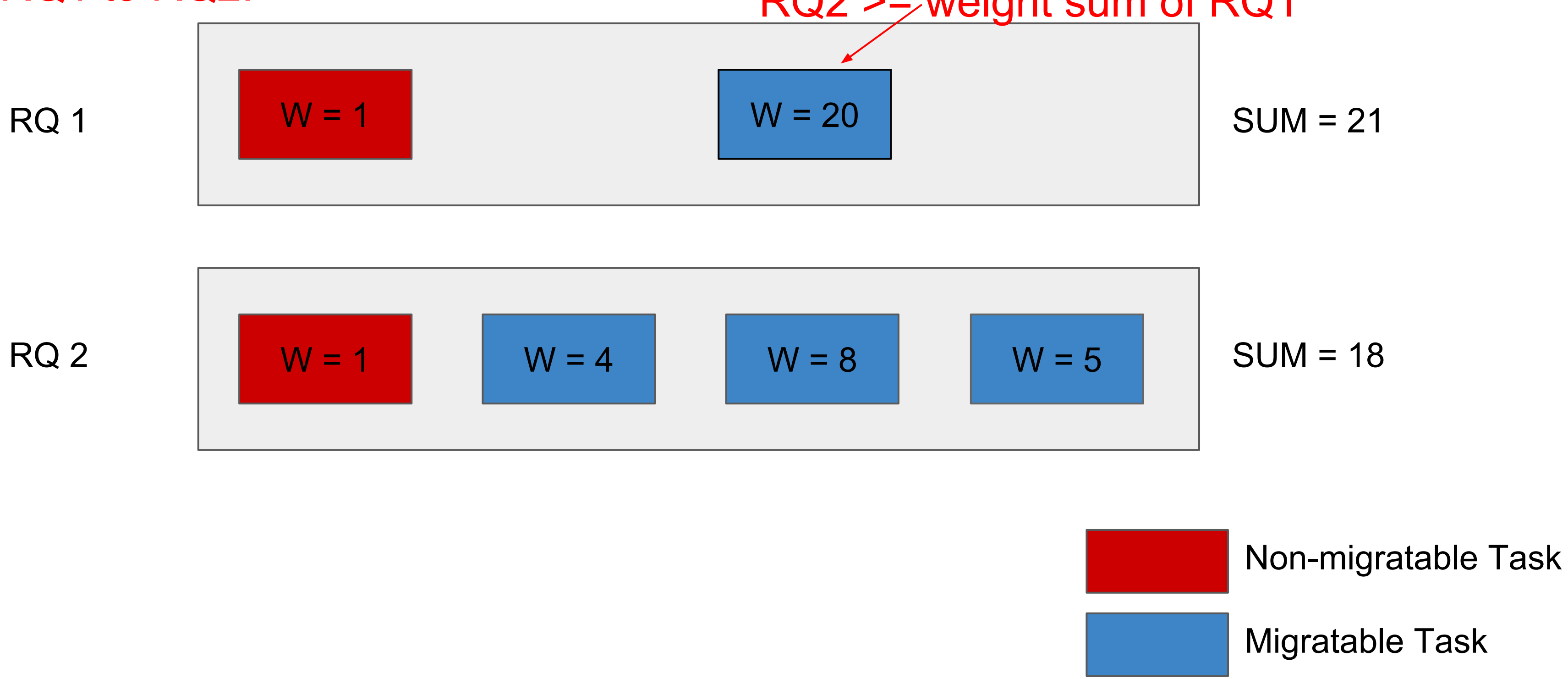
🟥 Non-migratable Task

🟦 Migratable Task

Load Balancing Example

Cannot migrate a task from RQ1 to RQ2!

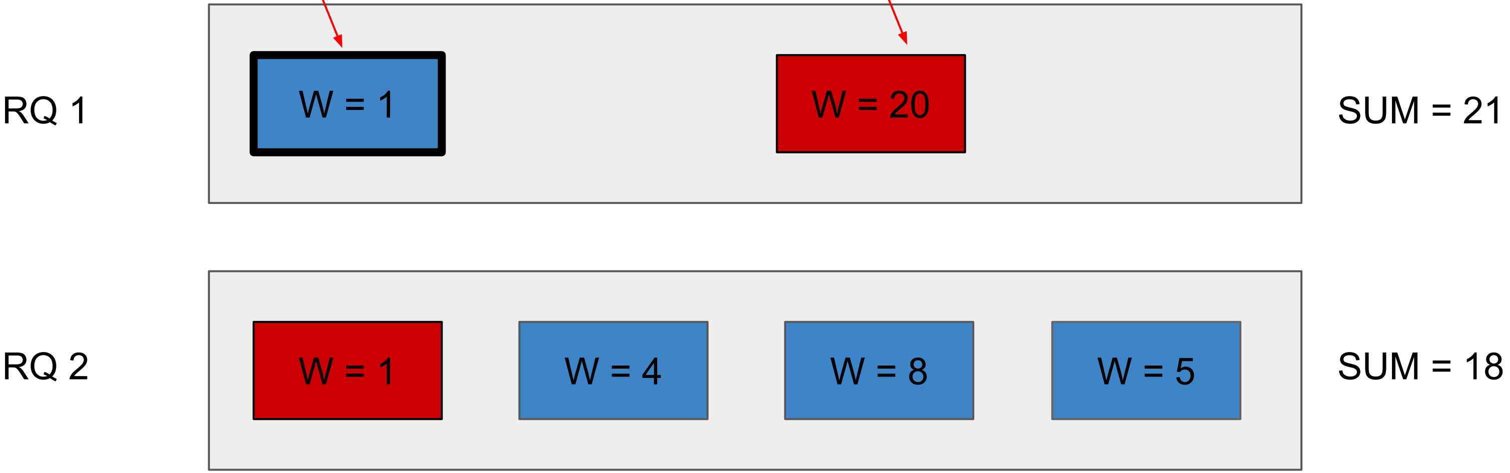
Migration of this task makes weight sum of RQ2 >= weight sum of RQ1



Load Balancing Example

Can be migrated now!

Non-migratable task has been changed!
Ex) Context switching

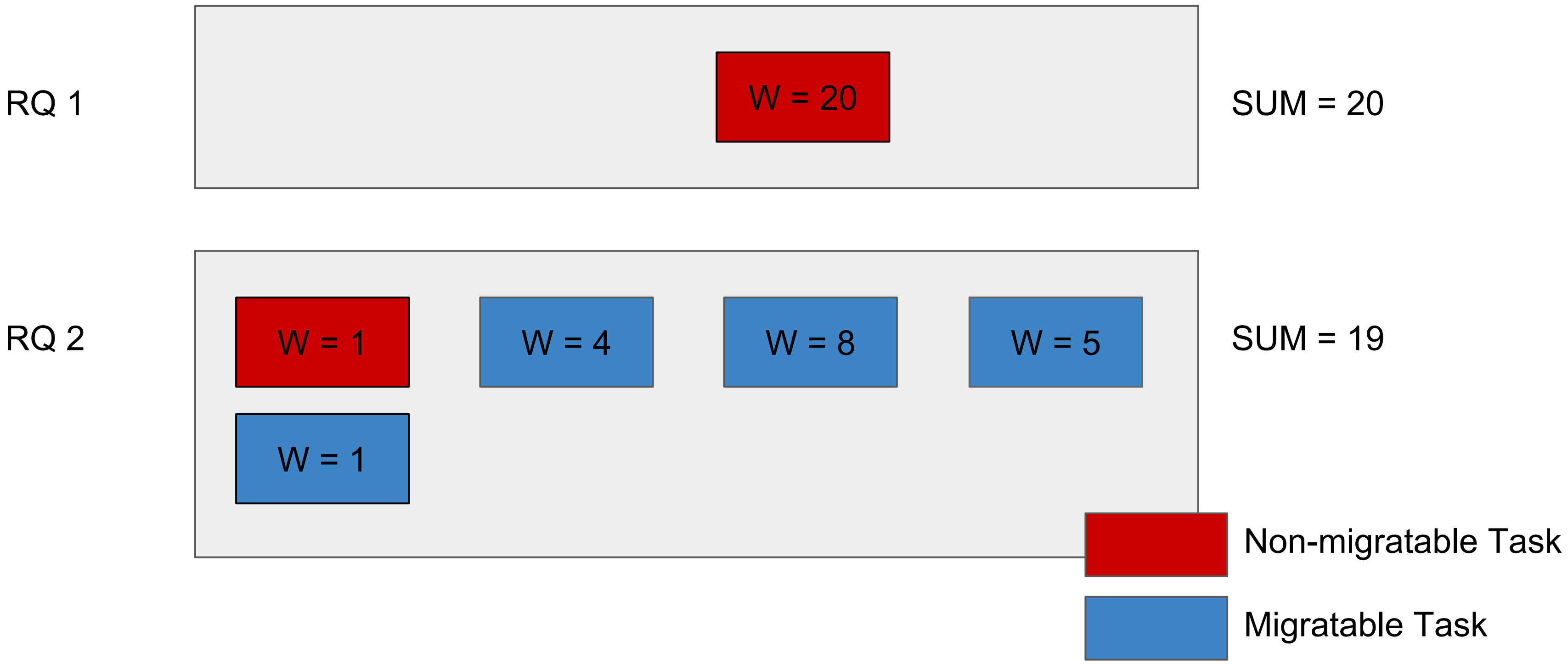


🟥 Non-migratable Task

🟦 Migratable Task

Load Balancing Example

After migration...



RQ 1

W = 20

SUM = 20

RQ 2

W = 1

W = 4

W = 8

W = 5

W = 1

SUM = 19

🟥 Non-migratable Task

🟦 Migratable Task

Scheduler Implementation

Before Start...

- Modify arch/arm/configs/artik10_defconfig
    - Find CONFIG_SCHED_DEBUG and make it to “CONFIG_SCHED_DEBUG=y”
        - You need this option to debug your scheduler
        - Performance degradation could happen
    - (Optional) Enable CONFIG_SCHEDSTATS for more detailed debugging

Implementation Overview (1)

- Define necessary constants and data structures
  - include/linux/sched.h
  - include/uapi/linux/sched.h
  - ...
- Register a new scheduler class for WRR and implement necessary functions in **kernel/sched/wrr.c**
- Modify **kernel/sched/debug.c** to print additional necessary information about WRR scheduler
  - Optionally kernel/sched/stats.c too

Implementation Overview (2)

- Modify **kernel/sched/core.c** to support WRR
  - Ex) Trigger load balancing function
  - You might need to register function signatures of wrr.c in kernel/sched/sched.h for them to be used in other files (ex: core.c)
- Implement necessary system calls, sched_setweight & sched_getweight
- After confirming WRR is working, make WRR as the default scheduling policy of init & kthread
  - include/linux/init_task.h
  - kernel/kthread.c

Define constants and data structures

- Define SCHED_WRR as **6**
    - include/**uapi**/linux/sched.h
- Define WRR scheduler information inside struct task_struct
    - Like RT or CFS
    - List head for putting into WRR run queue
    - weight, time slice, …
- Define run queue for tasks scheduled by WRR
    - “struct rq” should also have WRR run queue
        - struct rq is CPU run queue
    - What information should it have?
    - Should this have locking mechanism inside it?

Register a new scheduler for WRR

- Declare and define **wrr_sched_class** in kernel/sched/sched.h and in kernel/sched/wrr.c
  - Take a look at kernel/sched/fair.c & kernel/sched/rt.c
  - Its next scheduler class should be fair_sched_class because it has higher priority than that!
  - Similarly, the next scheduler class of rt_sched_class should be wrr_sched_class
- Define necessary functions used for defining wrr_sched_class
  - enqueue_task, dequeue_task, pick_next_task, …
  - You don’t need to implement all those functions
- Define other necessary functions for load balancing or debugging

Modify kernel/sched/core.c to support WRR

- Problem: It assumes that there are only rt_sched_class and fair_sched_class
- We need to make sure that they are aware of wrr_sched_class too!
    - Initialize WRR run queue
    - Make SCHED_WRR policy valid
    - Manage forked tasks
        - the child task should follow the same scheduler policy of parent
    - ...

Debugging WRR scheduler

- Remind: You should turn on CONFIG_SCHED_DEBUG option in artik10_defconfig
- You might want to modify kernel/sched/debug.c to check whether your WRR scheduler works properly or not
- Scheduling information is written to /proc/sched_debug
- In print_...()
  - You can print additional informations here like scheduling policy, wrr_weight, …
- In print_...()
  - Print additional statistics for WRR

Implement system calls

- You all know how to implement system calls!
- Authentication is important in sched_setweight()
  - Only the administrator may increase and decrease a process’ weight
  - The user who owns the process may decrease its process’ weight
  - Other users cannot adjust the process’ weight
  - You can check the process’ uid and euid to justify the ownership
- Nothing hard here :)

Experiment on WRR scheduler

- Main question: How the weight of WRR affects performance
- Measure the time for Trial program to finish for varying
    - Weights
    - Number of processes
    - …
- Important: You should make sure that all 8 cores are active when you start your experiment!
    - Initially, it is highly likely to have only one core active
    - You can make some number (about 10) of processes run for some time to make all CPUs active

More things...

- CFS is highly optimized, while your scheduler is not → Slow!
    - When the shell is not responding using WRR, just wait for a while
        - It's worse when only one CPU is on
        - It takes some time for other CPUs become active...
    - Do not make many processes in once (ex: forking 100 processes)
- Write multicore-related code only when CONFIG_SMP option is on!
    - You can make a use of #ifdef
- It's safe to have rcu_read_lock() when you are iterating on CPU cores
- This project is harder than project 2, so start early!

About submission (IMPORTANT!)

- Make sure your branch name: *proj3*
- Don't be late!
  - TA will not grade the commits after the deadline.
- Slides and Demo
  - Send it to the TA's email (os-tas@spl.snu.ac.kr) before the deadline.
  - os-tas@spl.snu.ac.kr
  - Title: **[OS-ProjX] TeamX slides&demo submission**
  - File name: **TeamX-slides.ppt(.pdf)**, **TeamX-demo.mp4(.avi….)**
- Check for format : slides title / demo name / branch name and directory name
- Please aggregate your demo videos (=submit only one video!)

Announcement

- Design Review
  - Team1~7 ⇒ Sungwoo Cho: pigbug419 **at** gmail **dot** com
    - Available schedule
      - Fri, 13:00~17:00
      - Tue, 10:00~15:25, 17:00~
  - Team8~14 ⇒ Kyungtae Kim: heaven **at** snu **dot** ac **dot** kr
- Check your source code before submission
  - There were some codes which were not compiled….

Q&A

Implement load balancing (1)

- Q1. How to check the remaining time slice or figure out when to trigger load balance?
- scheduler_tick()
    - in kernel/sched/core.c
    - Called every tick
- Tick frequency: HZ
    - A macro which represents the number of ticks in a second
    - For arm architecture, HZ = 100
        - **You shouldn’t make an assumption that it is 100**

Implement load balancing (1)

- Q1. How to check the remaining time slice or figure out when to trigger load balance?
- scheduler_tick()
- Tick frequency: HZ
- jiffies
  - A global variable contains the number of ticks after system booting
  - unsigned long value - overflow could happen!
  - There are macros for comparing time
    - time_after(), time_before(), time_after_eq(), time_before_eq()
  - More things: http://www.makelinux.net/books/lkd2/ch10lev1sec3

Implement Load Balancing (2)

- Q2. How to determine the task is migratable or not?
- Tasks that are currently running are not eligible to be moved
- Some tasks may have some restrictions on which CPU they can be run on
    - How can we know it? and Why do they have restrictions?
    - Refer to existing load balancing codes to find the answer

Implement load balancing (3)

- Q3. How to prevent race condition while load balancing?
- scheduler_tick is called for every available CPU!
    - You need to make sure that only one thread is working on load balancing at any time!
- One seemingly simple & plausible solution
    - Make only a certain CPU can do load balancing
    - But, because CONFIG_HOTPLUG_CPU is on by default, the designated CPU could be turned off anytime…
    - What happens if the designated CPU is turned off? How can we prevent it?
- Think carefully about synchronization issues and hotplug CPU!