Operating System Project / Lecture 2

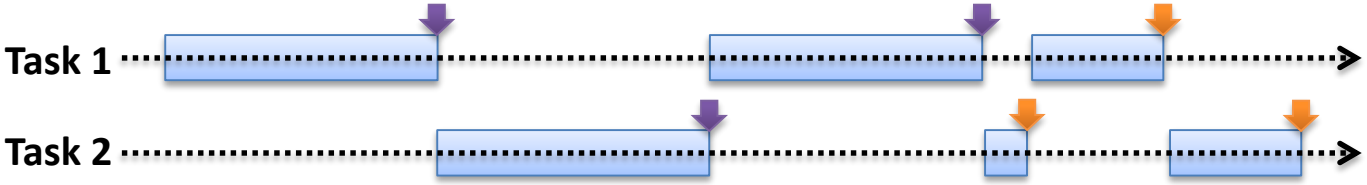
Scheduler

<Chapter 4 Process Scheduling>

Bon Keun Seo

CPU scheduling

- ***Time sharing*** to run more threads than processors

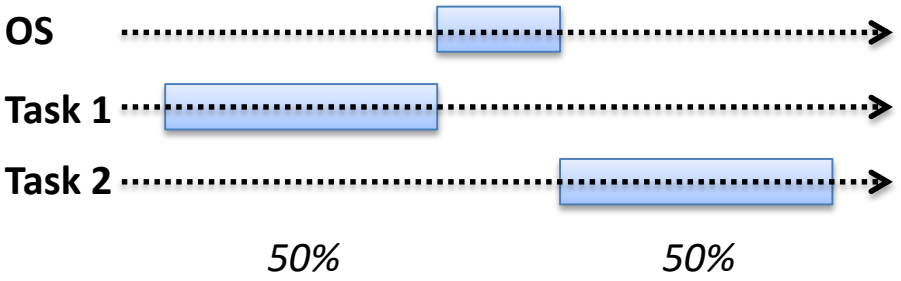


  - Interleaving processes
  - When a process waits for an event (I/O response)
  - Preemption (forced stopping of a task)

- **Scheduler determines**
  - Which task to run
  - How long it will run

Scheduling criteria

- ***CPU utilization***
  - Longer time slice, better CPU utilization
- Throughput
  - *# of tasks finished*
- Waiting time
  - *Interactive tasks*
- ***Fairness***



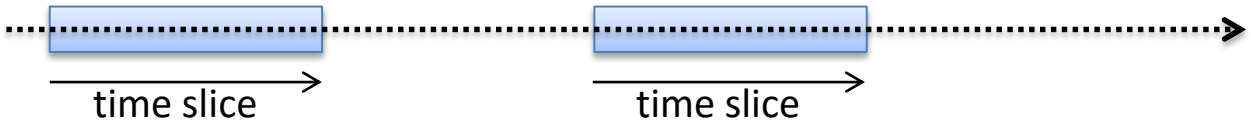
Scheduling algorithms

- **In OS course,**
  - First come, first served
    - *cf. First come, last served*
  - Shortest-Job-First
  - Priority
  - Round robin

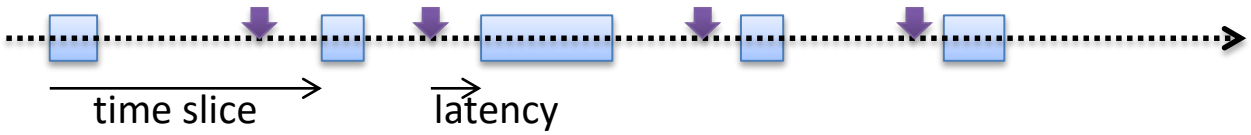
Is it so simple in practice?

CPU vs. I/O bound tasks

- **CPU-bound tasks**
  - Compression, encryption, encoding, …



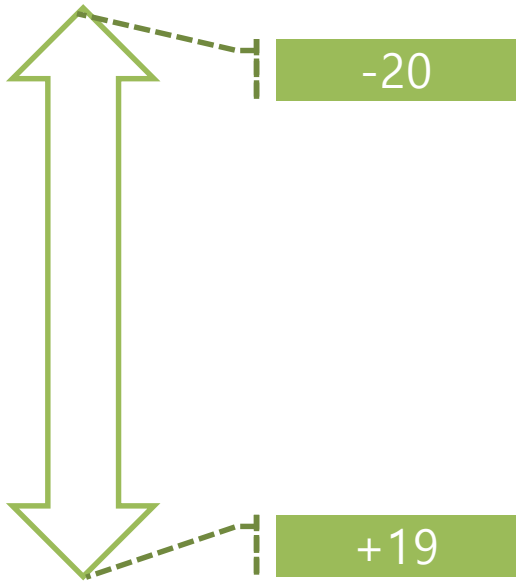
- **I/O-bound** (interactive) **tasks**
  - GUI applications, file server, web server, …
  - Latency sensitive, waste time slice



> A kind of task can turn into the other <

Priority in Unix

- ***nice*** value
  - -20 ~ +19
  - High priority ~ low priority
  - Default: 0
  - No definition of scheduler activity



Priority Space

Scheduler class

- **Generalization of scheduler**
  - More than 1 scheduler in a system
  - Real-time scheduler support

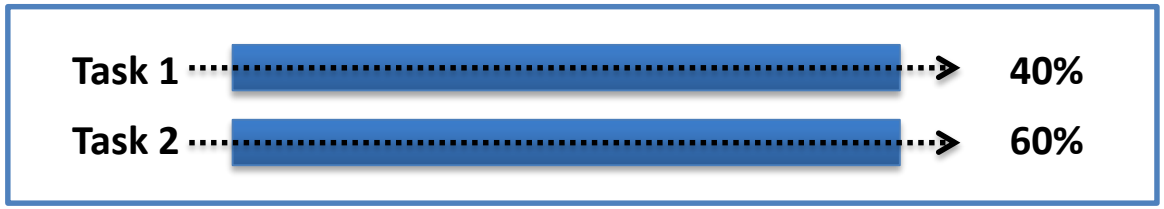
- `struct sched_class` `<kernel/sched/sched.h>`
  - Operations related to each scheduler
  - 4 classes

History of Linux scheduling

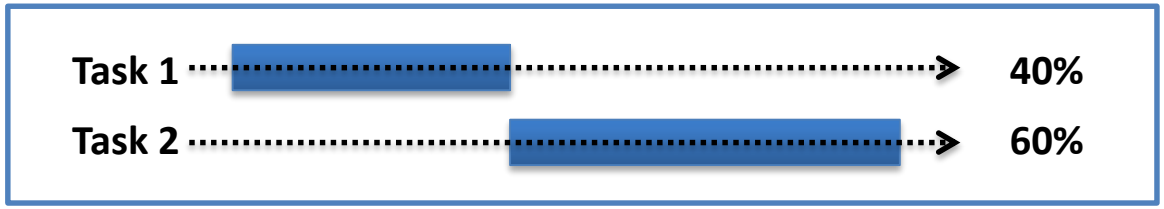
- **O(n) scheduler** < 2.6
  - n = # of tasks
  - Poor scalability
- **O(1) scheduler** < 2.6.23
  - Not good for interactive tasks
- **C**ompletely **F**air **S**cheduler
  - Fairness as a top priority
  - Interactive tasks get boosted

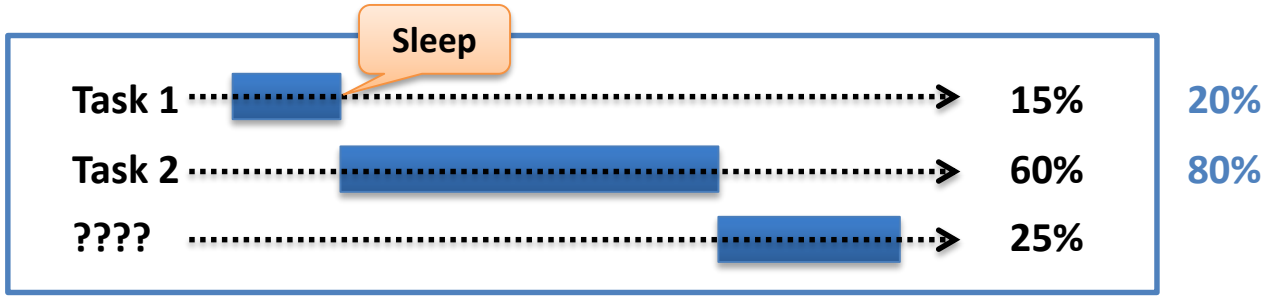
Fairness – *time slice*?

- Ideal



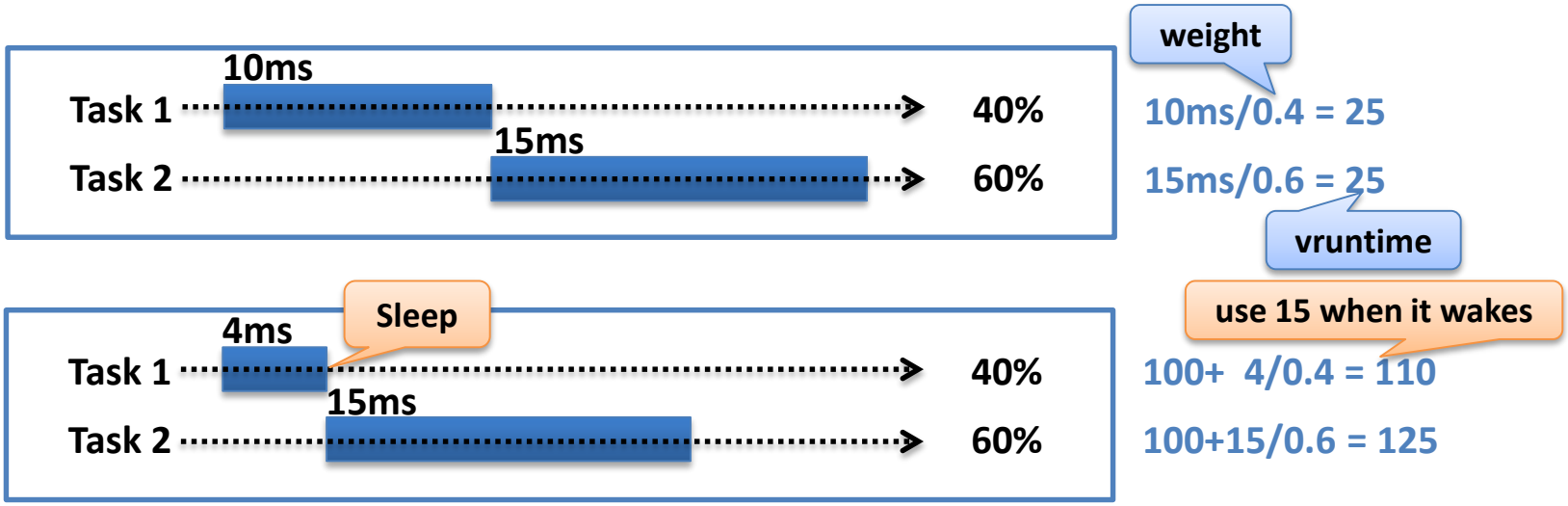
- Practical





Completely Fair Scheduler

- ***Weight***-based fairness
  - Weighted clock per each process



- Schedule task with minimum `vruntime`

CFS (2)

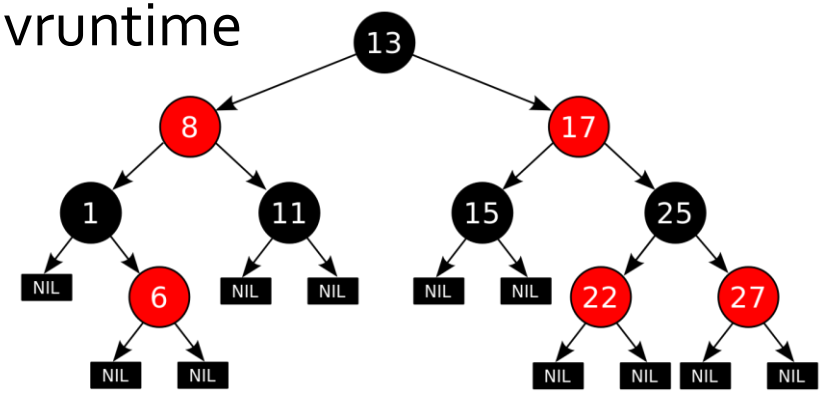
- **Schedule entity**
  - `struct sched_entity`
  - Runtime accounting

    vruntime += (now-exec_start)/weight
  - `update_curr()`

```
unsigned long weight;
unsigned long inv_weight;

struct load_weight      load;
struct rb_node          run_node;
struct list_head        group_node;
unsigned int            on_rq;
u64                     exec_start;
u64                     sum_exec_runtime;
u64                     vruntime;
u64                     prev_sum_exec_runtime;
u64                     nr_migrations;
```

- **Red-black tree**
  - To find a task with minimum vruntime
  - Balanced binary search tree
    - `__pick_next_entity()`
    - `enqueue_entity()`

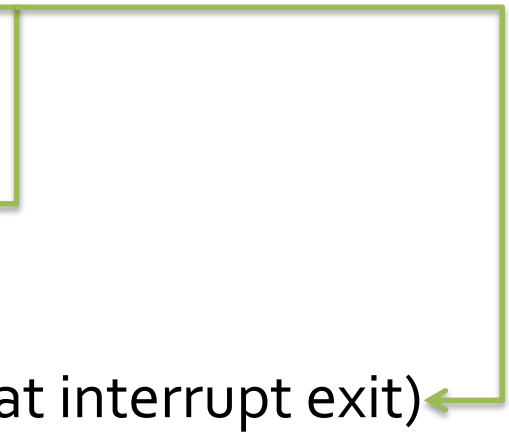


Real-time class

- **Priorities**
  - 0~99 ➔ 100 queues, one for each priority
  - Run tasks in a priority queue with lower value first
- **Policies**
  - SCHED_FIFO: non-preemptive
  - SCHED_RR: preemptive with time-slice

Preemption

- **Setting preemption flag**
  - `thread_info.flags & TIF_NEED_RESCHED`
    - `set/clear_tsk_need_resched()`
    - `need_resched()`
- **User preemption**
  - On system call and interrupt exit
- **Kernel preemption**
  - Yield to higher priority kernel tasks (at interrupt exit)
  - Cannot sleep with kernel lock
    - `thread_info.preempt_count == 0`
  - Explicit call of `schedule()`



Context switching

- **schedule()** ➔ __schedule() <kernel/sched/core.c>
  - Yield CPU to other task
  - **pick_next_task()**
    - Prefer CFS if (no RT task)
    - STOP ➔ RT ➔ CFS ➔ IDLE
  - **context_switch()**
    - arch_start_context_switch()
    - switch_mm()
    - switch_to()

Wait queues

- **Tasks go sleep inside kernel**
  - Run queue → wait queue
  - Wait queue can be created for each purpose
- **Example**

```
DEFINE_WAIT(wait);

add_wait_queue(q, &wait);
while (!condition) {
        prepare_to_wait(&q, &wait, TASK_INTERRUPTIBLE);
        process_something_before_wait();
        schedule();
}
finish_wait();
```

```
wake_up(&q);
```

```
wake_up_process(task);
```

Changes compared to textbook

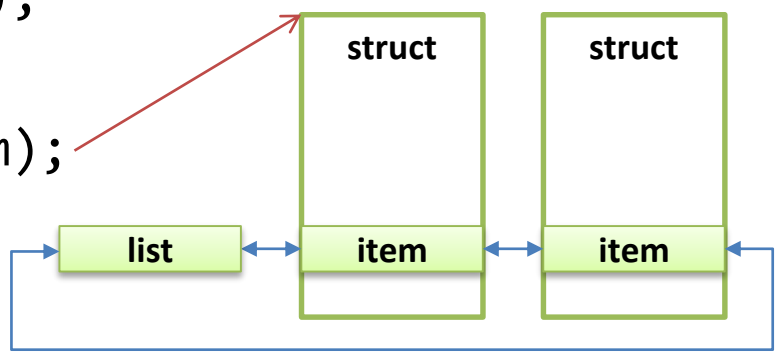
- `kernel/sched/` directory
  - kernel/sched.c ➔ kernel/sched/core.c
  - kernel/sched_fair.c ➔ kernel/sched/fair.c
  - kernel/sched_rt.c ➔ kernel/sched/rt.c

Kernel data structures

<Chapter 6 Kernel Data Structures>

Linked list

- **Generic list implementation**
  - `struct list_head { struct list_head *prev, *next; }`
  - `list_add(struct->item, list);`
  - `list_del(struct->item);`
  - `list_entry(ptr, struct, item);`



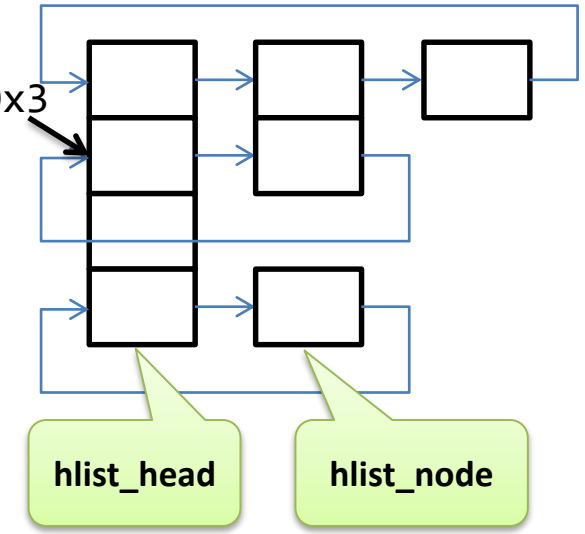
- **Traversing linked list**
  - `list_for_each_entry(p, list, member){`
    `do_something_with_p;`
    `}`

- Defined in `<include/linux/list.h>`

Hash table

- Handles conflicts with **chaining**: linked list!
  - 2 pointers for each hash queue?

```
struct hlist_head {
        struct hlist_node *first;
};
struct hlist_node {
        struct hlist_node *next, **pprev;
};
```



- **Functions**
  - hlist_add_head()
  - hlist_del()
  - hlist_for_each_entry()

Queue: kfifo

- Defined in `<include/linux/kfifo.h>`

- **Functions**
  - `kfifo_alloc()`
  - `DECLARE_KFIFO(), INIT_KFIFO()`

  - `kfifo_in()`
  - `kfifo_out()` ← `kfifo_out_peek()`

  - `kfifo_is_empty(), kfifo_is_full()`

Map

- ID to pointer translation service
- **Functions**
  - **Initialization**
    - `void idr_init(struct idr *idp)`
  - **Adding new ID**
    - `int idr_pre_get(struct idr *idp, gfp_t gfp_mask)`
    - `int idr_get_new(struct idr *idp, void *ptr, int *id)`
  - **Looking up**
    - `void *idr_find(struct idr *idp, int id)`
  - **Removing**
    - `void idr_remove(struct idr *idp, int id)`

Memory allocation

Bitmap

- Defined in `<arch/(arch)/include/asm/bitops.h>`
- **Functions**
  - `set_bit(long nr, volatile unsigned long *addr)`
  - `clear_bit(…)`
  - `test_bit(…)`
  - `test_and_set_bit(…)`
- Initialize by allocation

Red-black tree

- Defined in `<lib/rbtree.c>` and `<include/linux/rbtree.h>`
- **Data structures**
  - `struct rb_root`
    - Use `RB_ROOT` for initialization
  - `struct rb_node`
    - Embedded in a structure
- **Functions**
  - `rb_entry()`
  - Look up: tree traversal with `rb_node`
  - Insert: `rb_link_node()`
  - Rebalance: `rb_insert_color()`