Assignment 4 Tutorial

Linux Scheduler

Papadogiannakis Manos
papamano@csd.uoc.gr

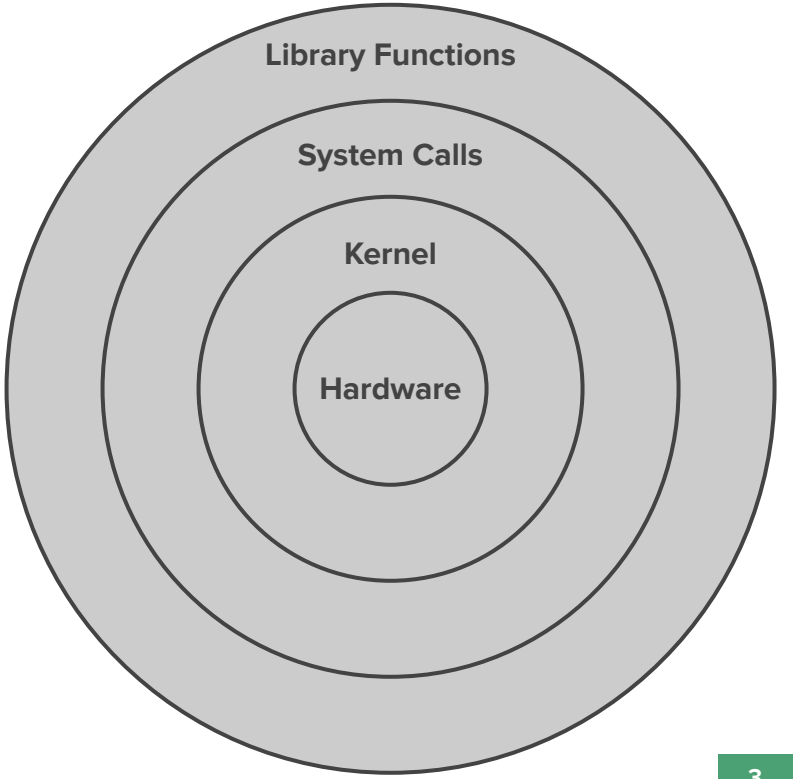
CS-345: Operating Systems
Computer Science Department
University of Crete

Outline

- Linux Scheduler
- Scheduler internals
- History
- Assignment 4

Linux Kernel

- **Heart of the Operating System**
- **Interface between resources and user processes**
- **What the Kernel does**
  - Memory Management
  - **Process Management**
  - Device Drivers
  - System Calls

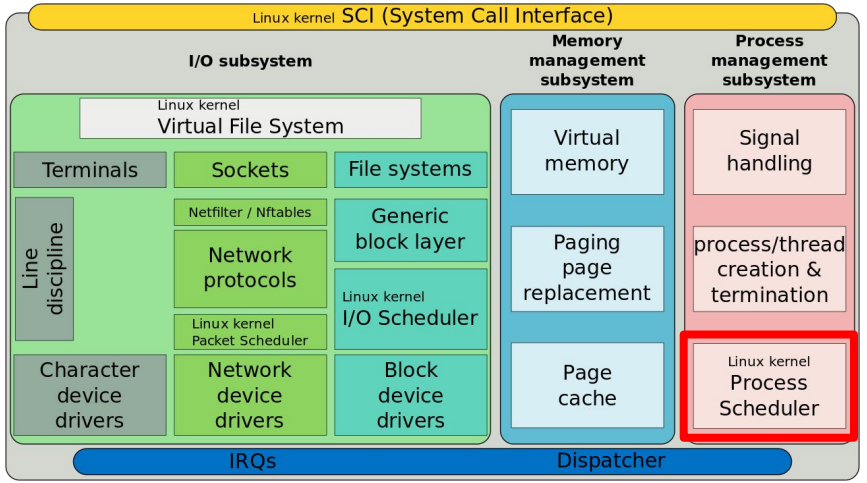


Process Management

- **Multitasking operating systems**
  - Tasks must run in parallel
- **Usually tasks are more than the CPU cores**
- **Need to make it possible to execute tasks at the “same” time**

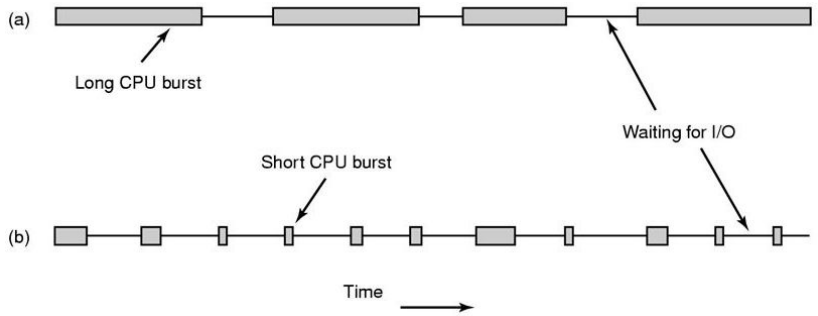
Scheduler

- Coordinates how tasks share the available processor(s)
- Prevents task starvation and preserves fairness
- Take into account system tasks



Task Types

- **Balance between two types of processes:**
  a. Batch processes
  b. I/O Bound tasks

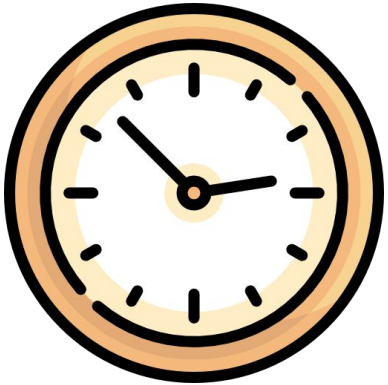


- **Preemption: temporarily evict a running task**

- **Quantum: Variable but keep it as long as possible**

Real-time processes

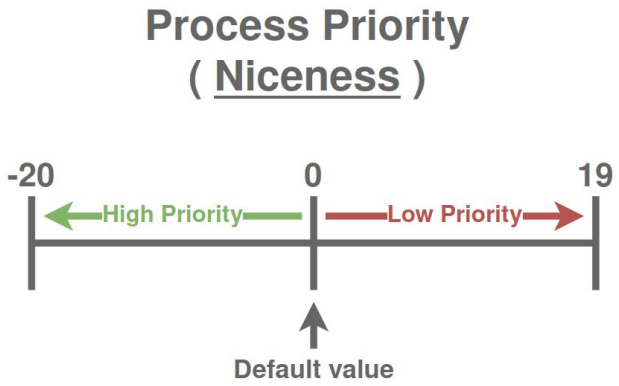
- **Need guarantee about their execution in time boundaries**
- **Soft real-time processes**
  - A task might run a bit late
- **Hard real-time processes**
  - Strict time limits
  - Not supported by default Linux

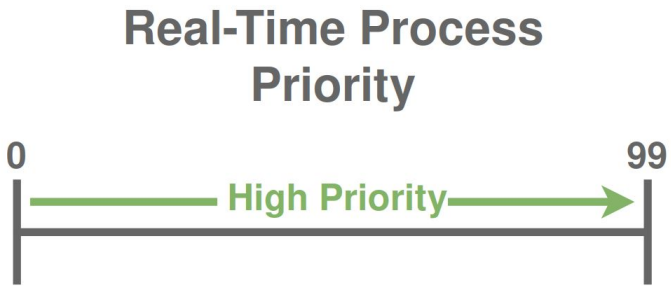


Scheduler Internals

Priority

- Linux provides Priority-based scheduling
- A “number” determines how important a task is

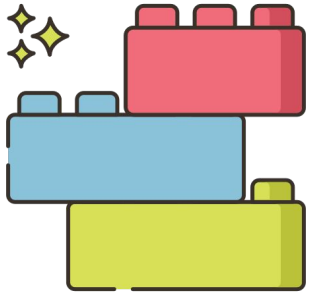




Process Descriptor

- **Scheduler needs information for each process**

- **Useful fields in task_struct:**
  - prio: Process priority
  - sched_class: Scheduling class
  - policy: Scheduling policy



Scheduler Design

- Extensible hierarchy of scheduler modules
- Each module encapsulates a scheduling policy
- Real-time classes:
  - SCHED_FIFO
  - SCHED_RR

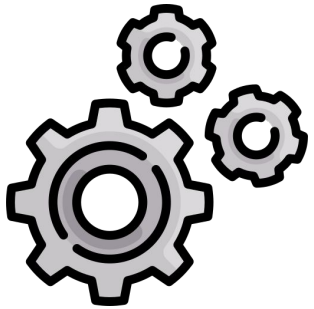
```
static const struct sched_class fair_sched_class = {
  .next                 = &idle_sched_class,
  .enqueue_task         = enqueue_task_fair,
  .dequeue_task         = dequeue_task_fair,
  .yield_task           = yield_task_fair,
  .check_preempt_curr   = check_preempt_wakeup,
  .pick_next_task       = pick_next_task_fair,
  .put_prev_task        = put_prev_task_fair,

...
```

https://elixir.bootlin.com/linux/v2.6.38.1/source/kernel/sched_fair.c

`schedule(void)`

- **Main scheduler function is `schedule( )`**
  - Replace currently executing process with another

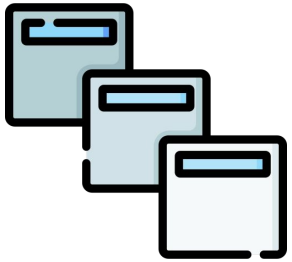
- **Called from different places**
  - Periodic scheduler
  - Current task enters sleep state
  - Sleeping task wakes up



Run queue

- Data structure that manages active processes

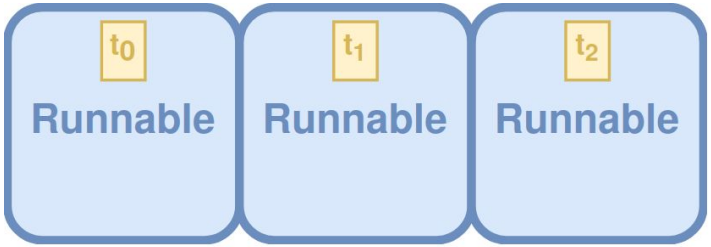
- Holds tasks in the “runnable” state



History

History

- **Genesis**
  - Circular queue
  - Round-robin policy



- **Linux v2.4 - O(n) scheduler**
  - Each task runs a quantum of time in each epoch
  - Epoch advances after all runnable tasks have their quantum
  - At the beginning of each epoch, all tasks get a new quantum

History

- **Linux v2.6 - O(1) Scheduler**
  - Division between real-time and normal tasks
  - One list per priority

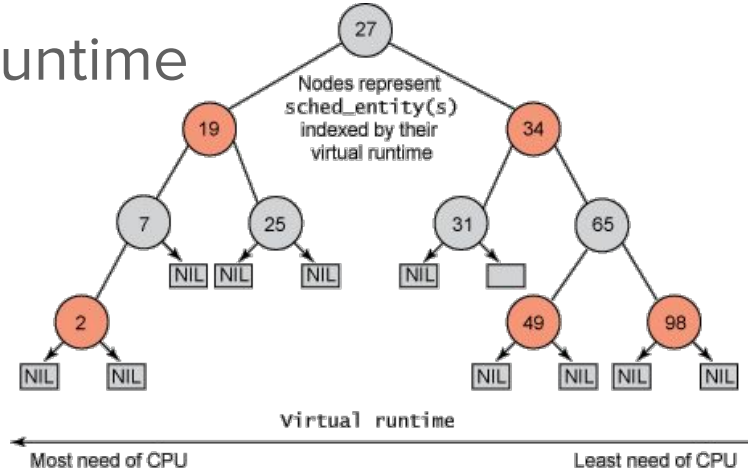
- **Linux v2.6.23 - CFS**
  - Introduced in 2007, Improved in 2016

Completely Fair Scheduler

- **Models an “ideal, precise multitasking CPU”**
- **Ideal scheduling: *n* tasks share 100/*n* percentage of CPU effort each**
- **Fairness:**
  - Tasks get their share of the CPU relative to others
  - A task should run for a period proportional to its priority

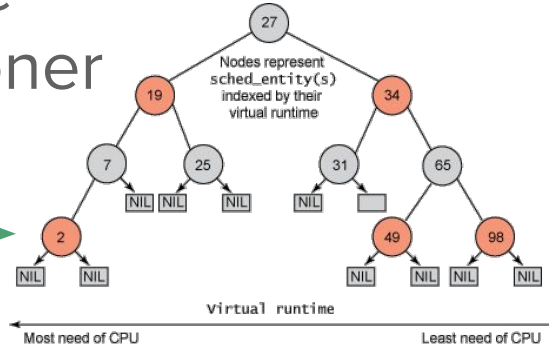
Completely Fair Scheduler

- **Time-ordered red-black tree**
  - Runnable tasks are sorted by vruntime
- **When a task is executing its vruntime increases**
  - Moves to the right of the tree
- **Scheduler always selects leftmost leaf**
  - Task with smallest vruntime



Completely Fair Scheduler - Improvements

- **Virtual clock ticks slowly for important tasks**
  - Move slower to the right of the tree
  - Chance to be scheduled again sooner
- **Leftmost node is cached**
  - O(1) access



- **Reinsertion of preempted tasks takes `O(logn)`**

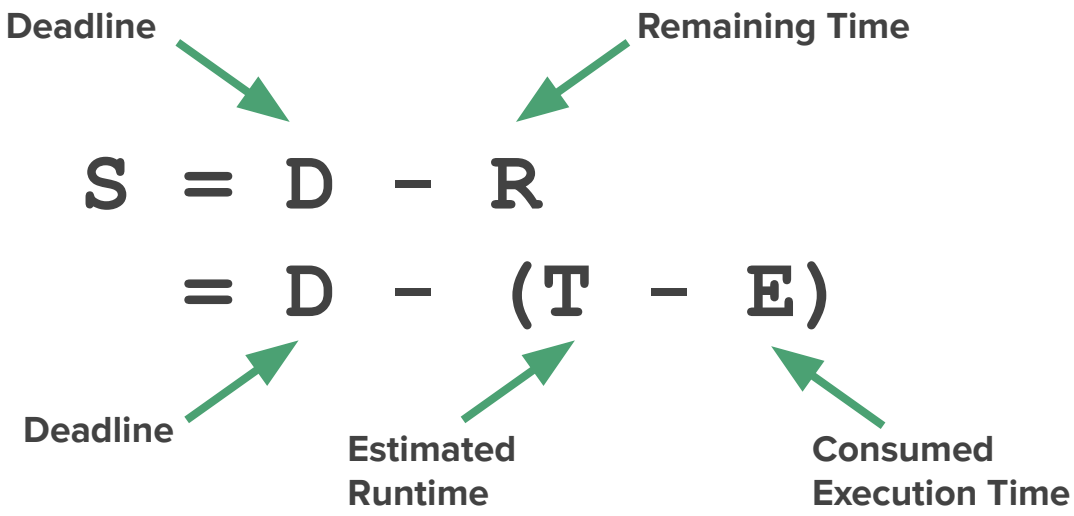
Assignment 4

Assignment 4 - Shortest Task First

- **Each process is defined by:**
  - Deadline
  - (Estimated) Runtime

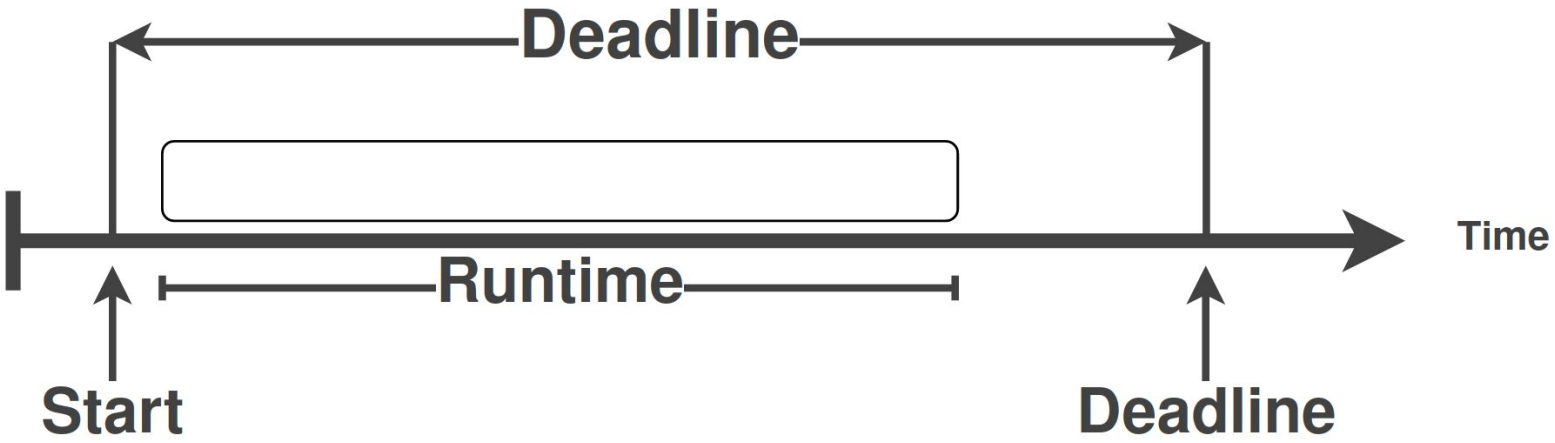
- **“*The process with the shortest execution margin should go first*”**

Assignment 4 - Shortest Task First

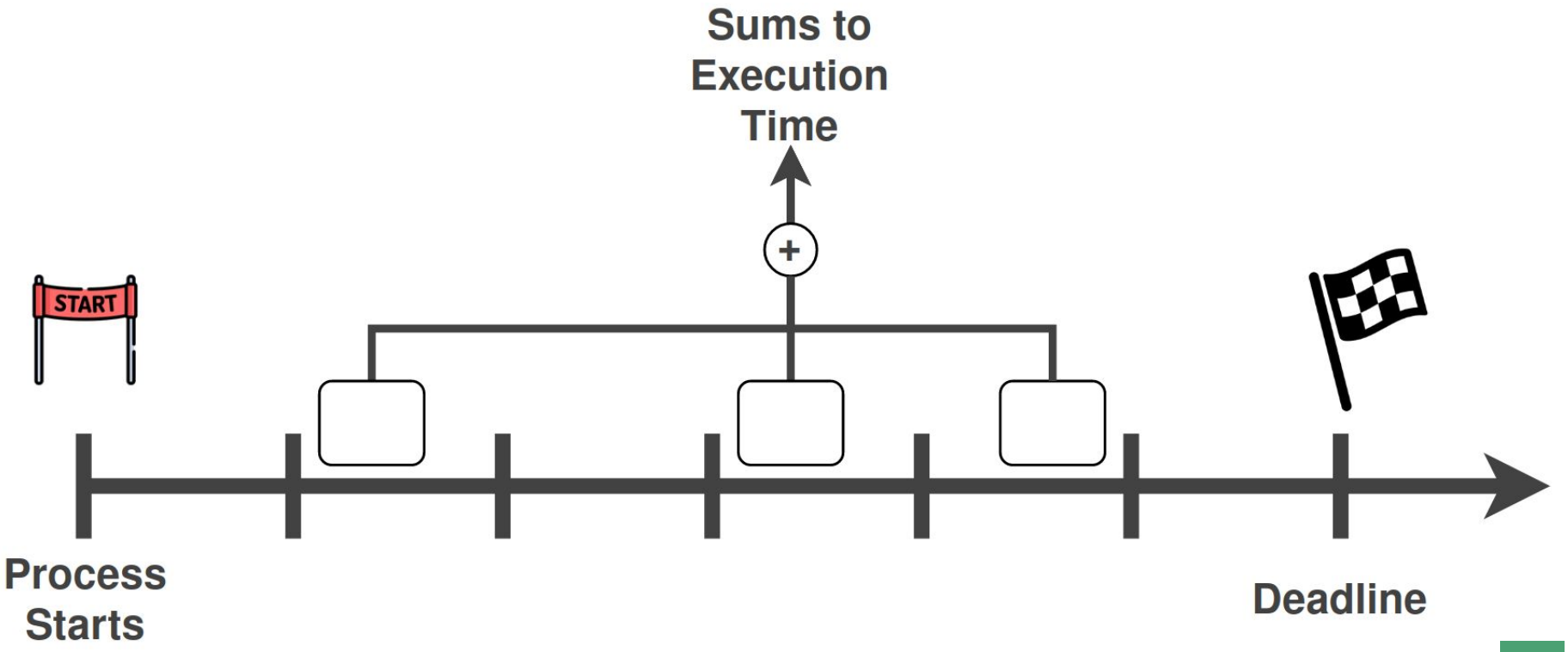


“Smallest *S* goes first”

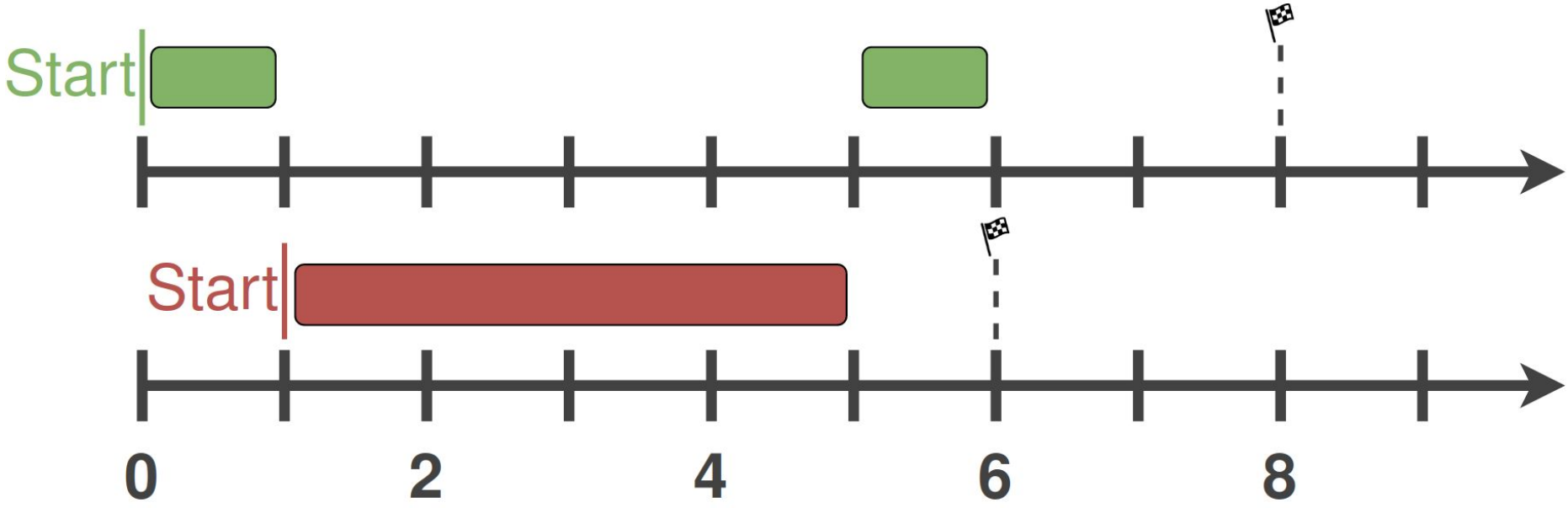
Shortest Job Definition



Scheduling Definition

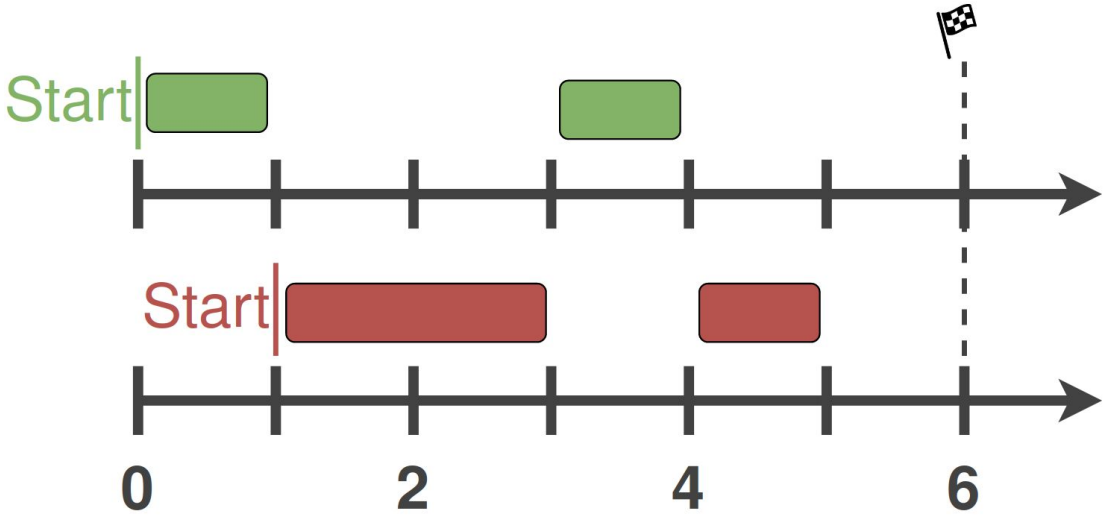


Shortest Job First - Preemption



P_1: Deadline 8s - Runtime: 2s P_2: Deadline 6s - Runtime: 4s

Shortest Job First - Example



P_1: Deadline 6s - Runtime: 2s

P_2: Deadline 6s - Runtime: 3s

Time	P1	P2
0	6-(2-0)=4	-
1	6-(2-1)=5	6-(3-0)=3
2	6-(2-1)=5	6-(3-1)=4
3	6-(2-1)=5	6-(3-2)=5
4	-	6-(3-2)=5
5	-	-

Implementation

- **Use your code from assignment 3**
  - System calls related to period processes
- **Linux kernel compilation process**
  - Instructions in assignment 3
- **Might need to make changes to `task_struct`**

Testing

- **Create simple demo processes**
  - Each initially sets its parameters
- **Each process should spin for some time**
  - Infinite loop, not sleep
- **Scheduler should print:**
  - PID of the task it selected
  - Its parameters


```
[Timestamp: 0] Selected process P1 with: Deadline 8s, Runtime 2s, Consumed Time 0s
[Timestamp: 1] Selected process P2 with: Deadline 6s, Runtime 4s, Consumed Time 0s
[Timestamp: 2] Selected process P2 with: Deadline 6s, Runtime 4s, Consumed Time 1s
[Timestamp: 3] Selected process P2 with: Deadline 6s, Runtime 4s, Consumed Time 2s
[Timestamp: 4] Selected process P2 with: Deadline 6s, Runtime 4s, Consumed Time 3s
[Timestamp: 5] Selected process P1 with: Deadline 8s, Runtime 2s, Consumed Time 1s
[Timestamp: 6] No process to select
```

Notes

Files

- **Actual context switch**
  - kernel/sched.c
- **Completely Fair Scheduler**
  - kernel/sched_fair.c
- **Scheduling structs**
  - include/linux/sched.h
- **Process descriptor**
  - include/linux/sched.h
- **Real-time scheduling**
  - kernel/sched_rt.c

sched.c

```
asmlinkage void __sched schedule(void) {

struct task_struct *prev, *next;
 ...
struct rq * rq;
 ...
preempt_disable();
 ...
prev = rq->curr;
 ...
pur_prev_task(rq, prev);

next = pick_next_task(rq);

 ...
if (likely(prev != next)) {
 ...
    context_switch(rq, prev, next);
```

Previous and next tasks

The processors runqueue (1 in this assignment)

Disable preemption (avoid schedule inside schedule)

Previous is the current task runnin

Put prev task in the runqueue

The appropriate pick function is called depending on the scheduling class

Actual context switch

Notes

- **Use Bootlin to find functions, structs, etc...**
  - https://elixir.bootlin.com/linux/v2.6.38.1/source
- **You can also map source code using ctags**
  - http://www.tutorialspoint.com/unix_commands/ctags.htm
- **Understand how the scheduler works**
  - Use **printk** to observe kernel behavior
  - Follow the call to find out how the next tasked is picked

Notes

- **Reuse existing code snippets within the kernel**
  - E.g. traversing data structures
- **Compile often with small changes**
  - Massively helps debugging
- **Submit anything you can to show your effort!!!**
  - A **README** file goes a long way

Turnin

What to submit:

1. bzImage
2. Modified or created source files
3. Test programs and headers in Guest OS
4. README

Credit

- Icons from FlatIcon, made by:
  - DinosoftLabs
  - surang
  - Flat Icons
  - Freepik
  - Smashicons

Thank You!

papamano@csd.uoc.gr

Questions?