Operating Systems 20/21
Task 2: Process Scheduling

Tom Spink

tspink@inf.ed.ac.uk

Overview

Your second task is to implement two scheduling algorithms:

Round Robin & FIFO

Deadline

The **STRICT** deadline is: **Week 6 25/02/2021**

Thursday at **4pm GMT**

Specification Document and Submission

Available on the course website, under the coursework section:

https://course.inf.ed.ac.uk/os

The InfOS Process Scheduler

- InfOS has an interface for scheduling called the scheduling algorithm.
- Your job is to write two implementations of this interface, by creating a:
  - Round-robin scheduler, and a
  - FIFO scheduler
- You don’t need to worry about the time quantum for a process - this is handled by the scheduler core.
- Look at sched-cfs.cpp for inspiration.

Task 1: Round-robin Process Scheduler

- Provided skeleton is: `sched-rr.cpp`
- Implement these three methods, as per the specification:
  - `add_to_runqueue`
  - `remove_from_runqueue`
  - `pick_next_task`

Task 1: Round-robin Process Scheduler

- Test by using the `build-and-run.sh` script
- If your implementation is broken, it's likely the system will hang.
- When you get to the shell, try running the scheduler tests:
  - `/usr/sched-test1`
  - `/usr/sched-test2`
- Also run some more programs to make sure starting and stopping processes works.
- Don't worry about output ordering (this is not important), just make sure multiple threads are starting and stopping.

Task 1: FIFO Process Scheduler

- Provided skeleton is: `sched-fifo.cpp`
- Implement these three methods, as per the specification:
    - `add_to_runqueue`
    - `remove_from_runqueue`
    - `pick_next_task`

Task 1: FIFO Process Scheduler

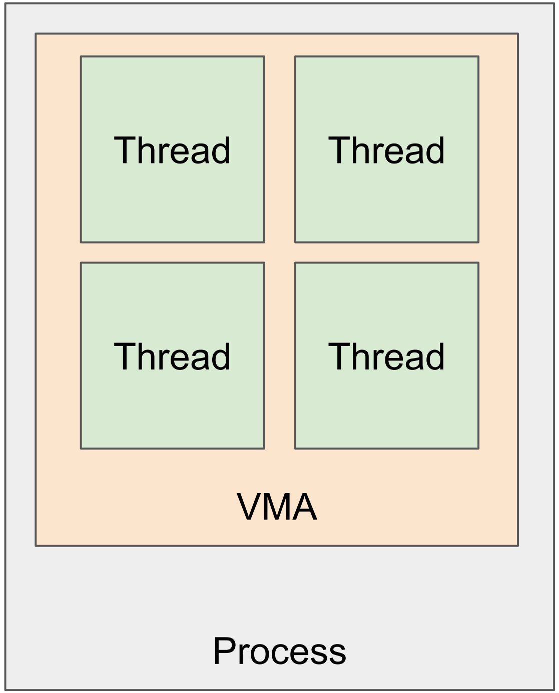
- Test by using the `build-and-run.sh` script
- If your implementation is broken, it's likely the system will hang.
- When you get to the shell, try running the scheduler tests:
  - `/usr/sched-test1`
  - `/usr/sched-test2`
- **IMPORTANT:** `sched-test2` will stop further threads from being scheduled - which means you'll have to kill the OS with Ctrl+C in the debug terminal.
  - Bonus mark: Why is this?

InfOS Scheduler Core

- The scheduler core is contained within `kernel/sched.cpp`
- The default (example) scheduler is in `kernel/sched-cfs.cpp`

Process and Thread Model

- A process has a virtual memory space (VMA)
- A process is a collection of threads
- A thread shares the same VMA with other threads in the same process
- A thread is a schedulable entity
- A schedulable entity is something that can run on a processor core

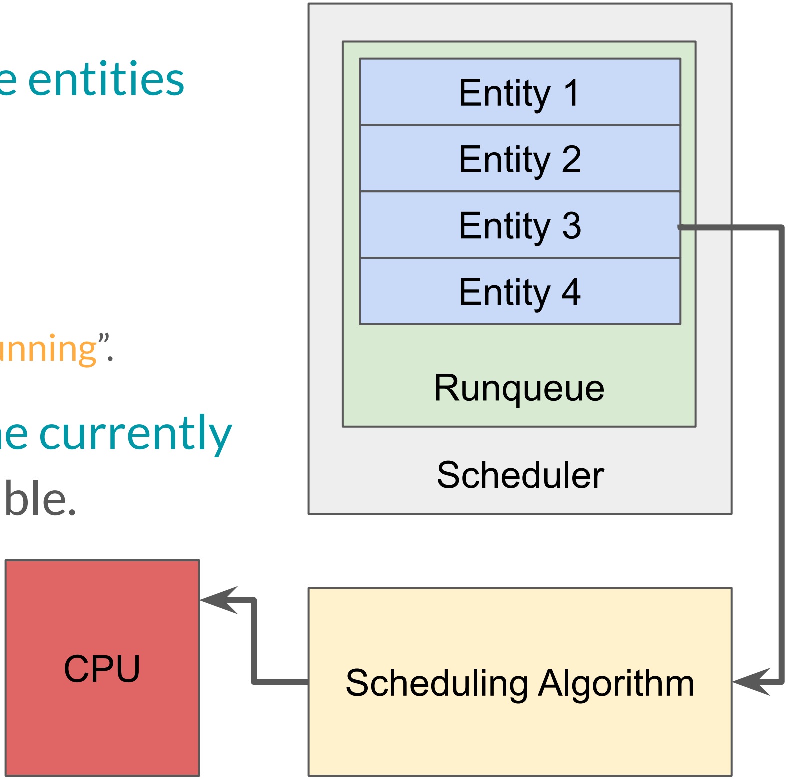


Schedulable Entity

- A schedulable entity is something that can be scheduled to run on a processor core
- It is essentially a thread, although provision is made for future expansion of this concept
- A scheduler schedules schedulable entities

Scheduler

- The scheduler maintains a list of schedulable entities that can run on a processor core.
    - The runqueue
- These entities are said to be “runnable”.
    - The entity that is currently running is said to be “running”.
- The scheduler is responsible for changing the currently running task, to the next entity that is runnable.
    - This is called a context-switch.
- The scheduling algorithm decides on the next eligible entity.



The Timer Interrupt

- InfOS has a timer interrupt that fires every 10ms
- This timer is the LAPIC timer (`drivers/timer/lapic-timer.cpp`)
- The timer interrupt updates the system runtime
  - This is why the clocks drift between the system and the hardware time!
- It also updates the currently executing thread's accounting
- It also (possibly) triggers a context switch

Context Switching

During a timer interrupt, the scheduler is asked to “schedule”.

The scheduler will ask the algorithm to pick the next entity

A few things might happen:

1. The algorithm returns NULL because nothing is runnable
2. The algorithm returns the SAME entity that is currently running
3. The algorithm returns a DIFFERENT entity than is currently running

Returning NULL

The scheduler picks the idle entity

Returning the SAME running entity

The scheduler does nothing

Returning a DIFFERENT entity

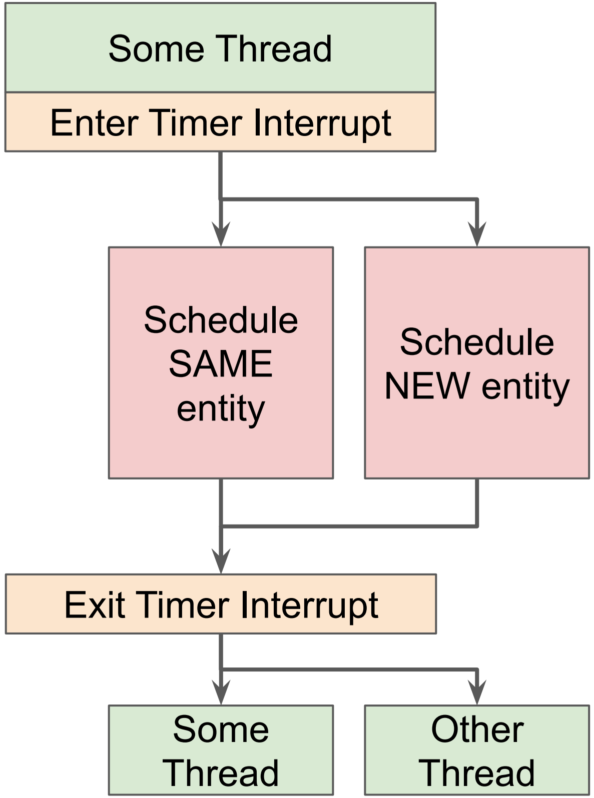
The scheduler will “activate” the new entity - which performs a context switch

Returning from the Timer interrupt

When the interrupt returns, the context to return into is loaded from the current thread object.

If the scheduler changed the "current thread" a context switch will have occured, as the return will be into a different context.

If the current thread wasn't changed, the original context will be loaded.


Some Thread
Enter Timer Interrupt
Schedule SAME entity
Schedule NEW entity
Exit Timer Interrupt
Some Thread
Other Thread