Scheduling

Multiple Tasks

PeriodicAction:

- Allows us to describe having several different tasks that must be performed by our processor
- Tasks are mostly independent of one another
- Tasks don’t take long to execute, but must be executed at some regular period

Tasks and Communication

In our set-up:

- Each task is implemented as a PeriodicAction
- Each has its own period and function to be called
- For the most part, tasks are independent
- But, in some cases, we need to communicate information between them
  - Many models for doing this

Tasks and Communication

For our implementation, all communication between tasks is through global variables (!)

- Typically, one task will write to the global variable
- And: one or more tasks will read from the variable
- Be careful with your use of global variables – in complicated code, it can be hard to track which tasks is writing to which variables

Hovercraft example

Tasks

One way to talk about a task:

- How often to execute it (the period)
- How long will it take to execute?
  - A possibility: just worry about the longest possible time

  This is called ***Worst Case Execution Time (WCET)***

(periodic action: timing diagram)

Task States

A task is only in one of three states:

- Ready
- Executing
- Waiting

Task States

With PeriodicAction:

- Ready: Time to begin execution
- Executing: Executing
- Waiting: Waiting for the next time period to execute

Task Deadlines

When must a task complete execution?

Task Deadlines

When must a task complete execution?

- Can be specified as part of the task
- Often, we make the choice that the task must complete execution before it’s next period of execution

With PeriodicAction:

- We don’t have a specific way to express/enforce deadlines
- But: we can detect when a deadline is missed (see the implementation)

Multiple Tasks

In general: we have multiple tasks

- Each have their own timing requirements
- Need to address all of them
- And not allow one to interfere with the others

(two task example)

Multiple Tasks

- Only one task may be executing at once
  - Other tasks must be waiting or ready
- When a task completes execution, we must choose which task to move from Ready to Executing
  - This is the job of the scheduler
  - Many different choices here
- If one task is executing and another becomes ready to run, it must wait
  - This is ***non-preemptive*** scheduling

Multiple Tasks

Non-preemptive scheduling implications

- Tasks don’t necessarily execute as soon as they are ready
- The delay in execution can vary from period to period
  - This variation is called ***scheduling jitter***

Scheduler Type: Static Priority

- When a task is created, we also assign a fixed priority (an integer)
- If multiple tasks are ready to execute at once, the scheduler picks the one with the highest priority

- Idea: some tasks are more important than others
    - Want to address them first
    - Want to minimize jitter

Special Cases of a Static Scheduler

Rate Monotonic Scheduling

- Highest frequency task receives the highest priority

Shortest Job First:

- Shortest Worst Case Execution Time is highest priority
- Hard to anticipate what WCET will be
- But: if we do know WCET, then can prove that this minimizes time in the waiting state

Scheduling example 1

Scheduling example 2

Scheduling example 3

Scheduling example 4