CMPE 117

Lecture 2

© Luca de Alfaro, 2002

Desirable features in real–time systems

- Timeliness.
- Predictability.
- Fault tolerance

And also, of course, performance:

- Ability to schedule time precisely
- Fast response time

Achieving predictability

- Many causes of timing uncertainty:
  - DMA
  - Cache
  - Pipeline behavior
  - Interrupts
  - Resource interaction (shared resources with access control)
  - Memory management
  - Programming languages

DMA

- Problem: cycle stealing.
- Solutions: pre–reserved cycles, or profiling.

Cache

- Problems:
  - Difficult to model
  - Strongly depends on the past (interrupt processing, which jobs executed in the meantime)
- Solutions:
  - Experimental. This does not provide bounds on the Worst Case Execution Time (WCET).
  - Accurate modeling of pipeline architecture, and cache. Precision of WCET computation: now 10% [USES group, University of the Saarlands, Germany]

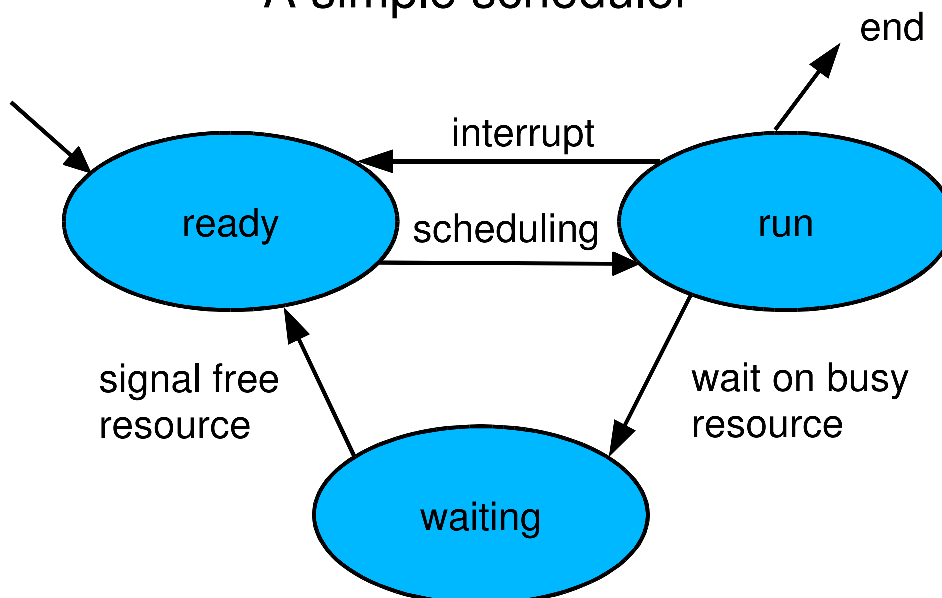
Programming Languages

- Predictability:
  - Garbage collection? (There are real–time algorithms, but mostly, static allocation is used)
  - Recursion?
  - Loops? (Time–bounded loops?)
  - Dynamic/general data structures (what about access time – can it be bounded?)

Programming Languages

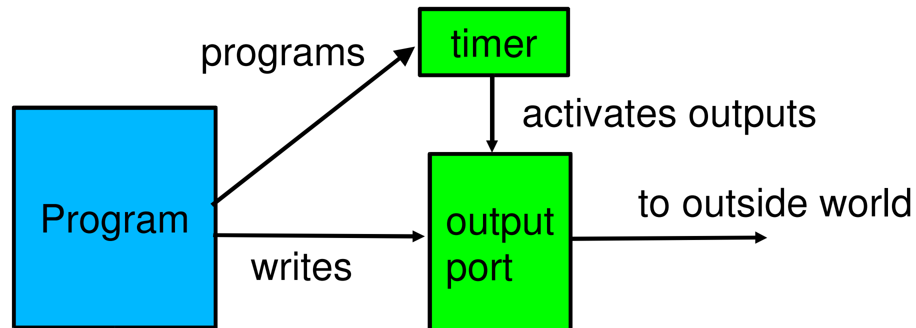
- Specification of real–time behavior:
  - Schedule alarms
  - Specify timeouts
  - Specify priorities
- What is missing:
  - Specification of deadlines (to help the scheduler)
  - Specification of structured activity (precedence constraints, structured loops such as periodic tasks)
  - Specification of imprecision (e.g., jitter)

A simple scheduler



Software – World interaction

- A general problem: how to enforce precise timing?
- Outputs: use time–triggered outputs



Inputs?

Two extremes:

- Asynchronous: interrupts. When an event happens, an interrupt occurs.
- Synchronous: The program polls the outside world, when it needs to read an event.

Heavvyweight interrupts

- An external event triggers an interrupt, and the interrupt service routine (ISR) does most of the work, reading the input, and activating the responding process.
- Used in non–real–time OSs.
- Problems:
  - Unpredictable timing, due to interruptions by ISR
  - Nested interrupts are difficult to handle

No interrupts

- Input events set a bit in a peripheal device. The bit can then be read (polled) under software control.
- Benefit: no interrupts (except the timer).
- Drawbacks:
  - Code for device driver tightly coupled with application code.
  - If busy waiting is required, this can lead to low processor efficiency.

Lightweight interrupts

- ISR read the state of the device that caused the interrupt, and do a minimal amount of processing: whatever is indispensible (flushing buffers, setting other timers, etc), plus activating the user routine that will then respond to the interrupt.
- Benetifs:
  - Flexible: all interrupts are serviced quickly, and we can choose independently the priority of the tasks they activate.
  - A bit unpredictable, but more efficient than polling.

Scheduling

Types of tasks

- Hard real–time (strict deadlines: process control, vehicle control, music, ...)
- Soft real–time (non–critical multimedia, applications where the average, rather than maximum, response time matters...).

Scheduling: vocabulary

- a: arrival time
- C: computation time
- d: deadline
- s: starting time
- f: finishing time
- $L = f - d$: lateness
- $E = \max(0,L)$: tardiness
- $X = d - a - C$: slack