Real-Time Systems

- Most programs depend on instructions executing in some sequence, but not when those instructions execute
- A *real-time program* interacts with the external world and is concerned with time
  - When a stimulus appears, the program must respond in a certain way, and before some specified deadline
  - If it delivers the correct answer, but after the deadline, the system has failed
- Example: CPU in CD player
  - CPU must be fast enough to read the CD and produce the music acceptably
- Other examples: aircraft subsystem controllers, scientific experiments, telephone switches, CAT scanners, etc.

Stimulus and Response

- Some external device generates a *stimulus*, and the real-time system must respond before some *deadline*
  - Stimuli may be *periodic* — occurring at regular intervals —a television needs new frame every 1/60th of a second
  - Stimuli may be *aperiodic* — recurrent, but not regular — arrival of an aircraft in an air traffic controller's airspace
  - Stimuli may be *sporadic* — unexpected, such as a device overheating
- System may have many types of events (e.g., video input, audio input, motor drive management), each with its own period and required actions
- Input may come from either an analog or digital device, but if analog, is converted into digital information

Types of Real-Time Systems

- *Soft real-time systems* — it is acceptable to occasionally miss a deadline
  - Telephone switch may be permitted to lose or misroute one call in 100,000
  - Multimedia system may miss delivering some video frames
- *Hard real-time systems* — it is never acceptable to miss a deadline
  - Extreme: missing a deadline may lead to loss of live or an environmental catastrophe
  - Less extreme: missing a deadline may mean in item on a conveyor belt in a factor misses being processed
- The solution is not necessarily faster computers, it's good scheduling that's important

Event-Triggered vs. Time-Triggered Systems

- *Event-triggered systems* — event is detected by a sensor, which causes a CPU interrupt
  - Simple, widely used, works well for soft real-time systems with sufficient computing power
  - ✘ Can fail under conditions of heavy load
    - Pipe ruptures in nuclear reactor, causing temperature alarms, pressure alarms, radioactivity alarms, etc. to all go off at the same time, causing many interrupts
- *Time-triggered systems* — clock interrupt occurs at regular intervals, and at that time selected sensors are sampled
  - In example above, system would notice all alarms at next time interrupt (but would not have to deal with many interrupts)
  - Less chance of failure at high load, but slower response time

Other Design Issues

- Behavior should be predictable
  - System should always be able to meet its deadlines, even at peak load
- Should support fault-tolerance
  - OK to use replication, "hot" backups, etc. but still must not miss deadlines
  - *Fail-safe systems* — can be stopped when a serious failure occurs
- Language support
  - Compiler must be able to determine maximum execution time of loops
    - No **while** loops, only **for** loops with constant parameters
    - No recursion
  - Language must be able to specify minimum and maximum delays

Real-Time Communication

- Predictability and determinism are more important than high performance
  - Stochastic LAN protocols such as Ethernet are unacceptable, because they do not guarantee an upper bound on transmission time
  - However, the token ring has a known upper bound (time to traverse the ring)
- Time Division Multiple Access (TDMA)
  - Traffic organized into fixed-size frames, each of which contains N slots
  - Each slot is assigned to one processor, which may use it to transmit a packet when its time comes
  - Collisions are avoided, delay is bounded, and each processor gets a guaranteed fraction of the bandwidth

Real-Time Scheduling

- System programmed as collection of short tasks (processes or threads), each with a well-defined function and well-bounded execution time
  - Response to a stimulus may require multiple tasks to be run, often with some constraints on their execution order
  - System has to decide which tasks to run on which processor, and when to run each task
- Hard real time vs. soft real time
- Preemptive vs. non-preemptive
- Dynamic (scheduling decisions made during execution) vs. static (scheduling decisions are made before execution)
- Centralized vs. decentralized

Static Scheduling

- Done before system starts operating
- Input: list of tasks, times that each must run (dependencies, time constraints)
- Output: assignment of tasks to processors, starting times for each
- Method:
  - Exhaustive search to find optimal solution, but this method is exponential in number of tasks, so seldom used
  - Should also consider communication
  - Runtime behavior is completely deterministic, and known before the program execution starts
  - System will always meet its real-time constraints, so long as there aren't processor or communication failures

Dynamic Scheduling

- Decides which task to run next as programs execute
  - Same input & output as static scheduling
- Methods (uniprocessor only, no consideration of data dependencies):
  - Rate monotonic algorithm (Liu and Layland, 1973)
    - Preemptive scheduling of periodic tasks
    - Each task is assigned a priority equal to its execution frequency
    - Scheduler selects highest priority task, preempting other tasks if necessary
  - Earliest deadline first (Jackson, <1977)
    - When an event is detected, scheduler adds it to list of waiting tasks
    - List is sorted by deadline, closest first
    - Scheduler selects task at head of the list, preempting other tasks if necessary

Dynamic Scheduling (cont.)

- Methods (uniprocessor only, no consideration of data dependencies):
  - Laxity / slack
    - Scheduler computes for each task the amount of time until its next deadline; this is called the "laxity", or "slack"
    - Scheduler selects the task with the least laxity, since it has the least scheduling freedom
  - These methods are not optimal in a multiprocessor / distributed system, but they are still useful as heuristics
- In general, static scheduling is the best fit for a time-triggered system, while dynamic scheduling is best for an event-triggered system
  - Optimal schedule is possible for stat. sch.
  - Dynamic scheduling must have sufficient resources for even unlikely cases

More Complex Scheduling

- May have to reserve service time for sporadic tasks
- Tasks may be non-preemptive
- Data dependencies modeled using a precedence graph
  - Can't schedule a task until all its predecessors have completed
- May have multiple processors (possibly of different types)
  - These problems are NP-hard
  - May have resource constraints — limit on number of resources of a particular type that may be in use at any point in time
- Tasks may have different (non-unit) execution times