Issues in Building Real-Time Applications

郭大維 教授
ktw@csie.ntu.edu.tw
嵌入式系統暨無線網路實驗室
(Embedded Systems and Wireless Networking Laboratory)
國立臺灣大學資訊工程學系

Introduction to Real-Time Systems

- Checklist
  - ⊕ What is a real-time system?
  - ⊕ What is the way usually used to classify real-time tasks?
  - ⊕ What are the issues and research for real-time systems?
  - ⊕ Is there any misconception about real-time computing?
  - ⊕ Is our current software development environments suitable to time-critical systems?
  - ⊕ What kinds of software architectures are adopted or considered in current time-critical systems?

Introduction to Real-Time Systems

- What is a real-time system?

  Any system where a timely response by the computer to external stimuli is vital!

- Examples:
  - multimedia systems, virtual reality, games.
  - avionics, air traffic control, nuclear power plant
  - stock market, trading system, information access, etc.

What is a Real-Time System?

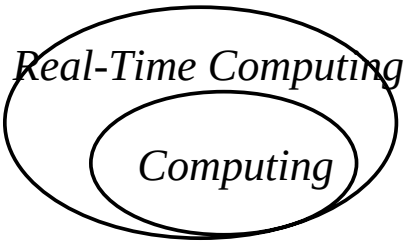
- Does the definition make every computer a real-time computer?

  Yes! It is if we need some response from a computer within a finite time!!

- Category of Real-Time Systems:
  - Hard Real-Time Systems - catastrophic if some deadlines are missed.
  - Soft Real-Time Systems - otherwise.

Issues in Real-time Computing

- The field of real-time computing is especially rich in research problems!

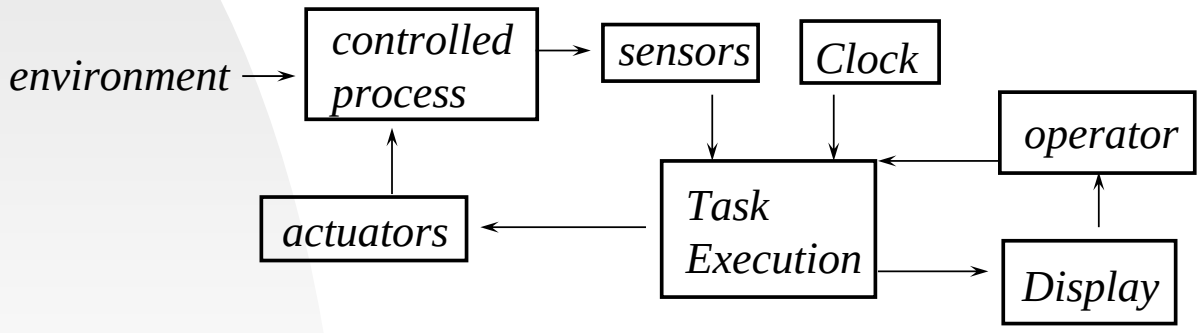


For example, CPU scheduling of tasks with different criticality!

- However, real-time computing systems often differ from their counterparts in two ways:
  - More specific in their applications.
  - More drastic for their failures.

Structure of A Real-Time System - An Example

- A control system



- Rates - sensors & actuators, peripheral, control program
- Phases - takeoff, cruise, and landing, etc.

Task Classes

- Ways to classify real-time tasks:
  - Predictability of their arrivals.
    - Periodic tasks have regular arrival times.
    - Aperiodic tasks have irregular arrival times.
      - bounded inter-arrival time -> Sporadic tasks.
  - Criticality - consequences of non-timely executions.
    - Critical tasks should have timely executions
      - Most of them are hard real-time transactions
    - Non-critical tasks are usually soft real-time tasks
      - minimize miss ratio, minimize response time, maximize values contributing to the system, etc.

Issues and Research

- Software engineering
  - System architecture, e.g., event-driven, time-line, time-driven, object-oriented, etc.
  - Network architecture, e.g., topology, predictability, and controllability.
  - Fault-tolerance and reliability evaluation, etc.
  - Tools for prototyping, simulation, code synthesis.
- Operating systems
  - Task assignment and scheduling
  - Communication protocols
  - Failure management and recovery
  - Clock Synchronization, etc.

Issues and Research

- Programming languages
  - Better control over timing
  - Proper interface to special-purpose devices
- Database systems
  - Concurrency Control
  - Failure recovery
  - Availability
  - Query Optimization, etc.
- Specification and verification
  - Expressiveness and complexity

Issues for Programming Environments

- Loop size, timer granularity, imprecise timer, sleep(), multi-programming, etc.
- Sequential programs, parallel programs, timely programs.
- Client-server priority assignments - priority inversion.
- Verification, analysis, and testing.

Issues for Programming Environments

- Potential Timing Hazards:

```
Loop
    ……..
    Sensor();
    ……..
    computation……
    ……..
    t = time();
    SleepTime := ReadyTime + PERIOD - t;
    ReadyTime = ReadyTime + PERIOD;
    Sleep(SleepTime);
EndLoop;
```

???Multiprogramming???

Loop Size?

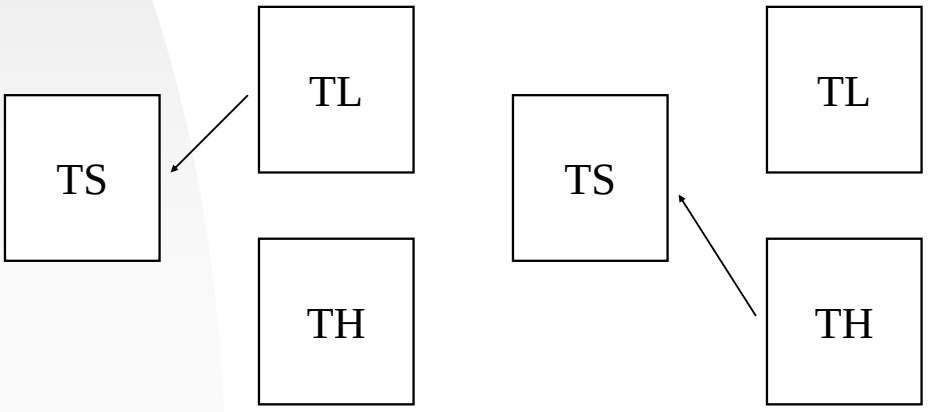
Timer Granularity?

Time Elapsed Here?

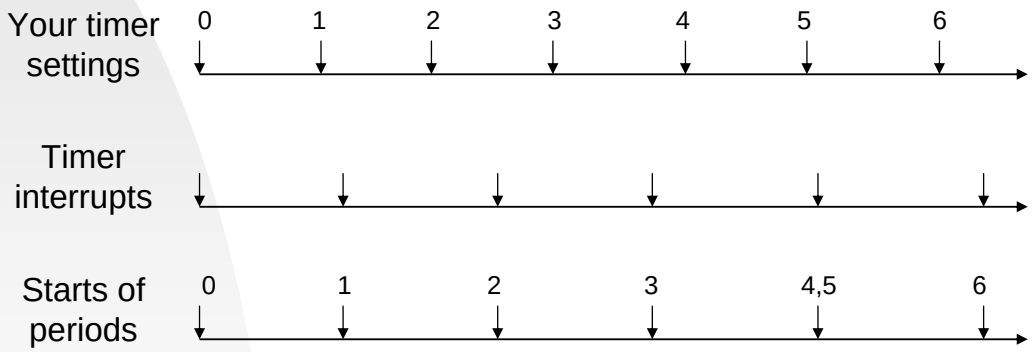
Real Sleep Time?

Issues for Programming Environments

- The priority assignment for a Server TS?
  - Processes TH and TL
  - Priority Inversion



Issues for Timer Drifting



- Set the timer resolution to x.
- Round off all of timeout intervals to integer multiples of x!

Software Architectures and Fault Tolerance Issues for Real-Time Applications

郭大維 教授

ktw@csie.ntu.edu.tw

嵌入式系統暨無線網路實驗室

(Embedded Systems and Wireless Networking Laboratory)

國立臺灣大學資訊工程學系

Source: C. Douglass Locke & Farnam Jahanian, RTCSA'96 Talks Presentation.

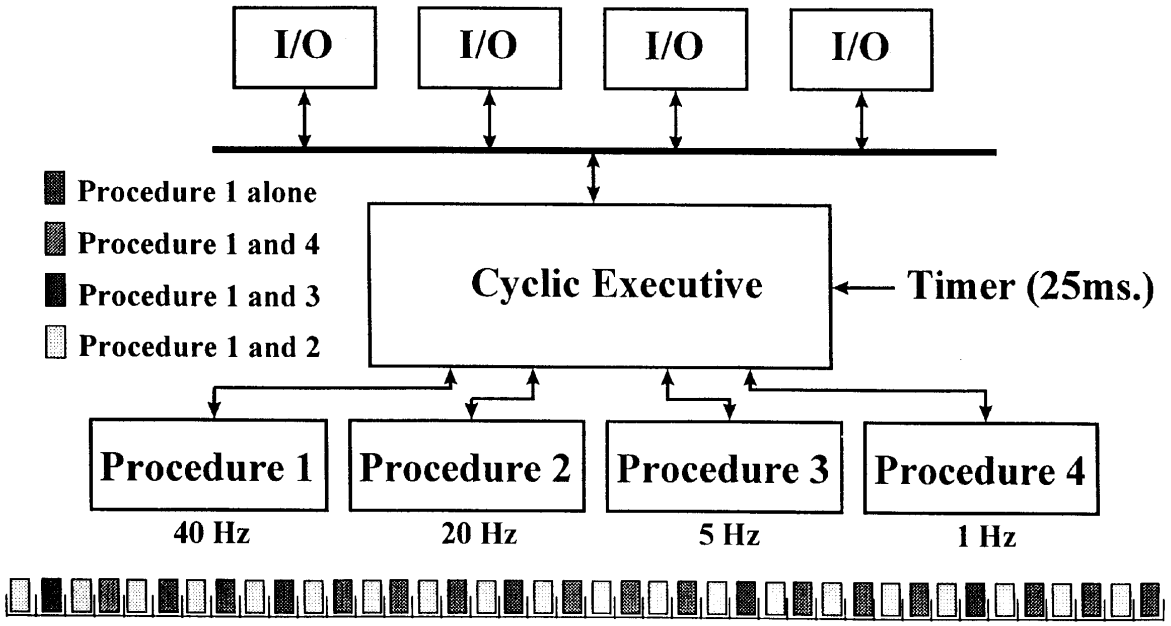
Software Architectures for Real-Time Applications

- Popular architectures:
  - Timeline (i.e., cyclic executive or frame-based)
  - Event-driven (with both periodic and aperiodic activities)
  - Pipelined
  - Client-Server
- Impacts
  - Performance and life-cycle cost
  - Critical design decisions such as synchronization and exceptions.
- No restriction on parallel processing.

Timeline or Cyclic Executive

- A major cycle consists of a non-repeating set of minor cycles
  - Operations are implemented as procedures.
  - The timer calls each procedure in the list.
- No concurrency exists.
- Very high life-cycle cost but very predictable in the run-time behavior!

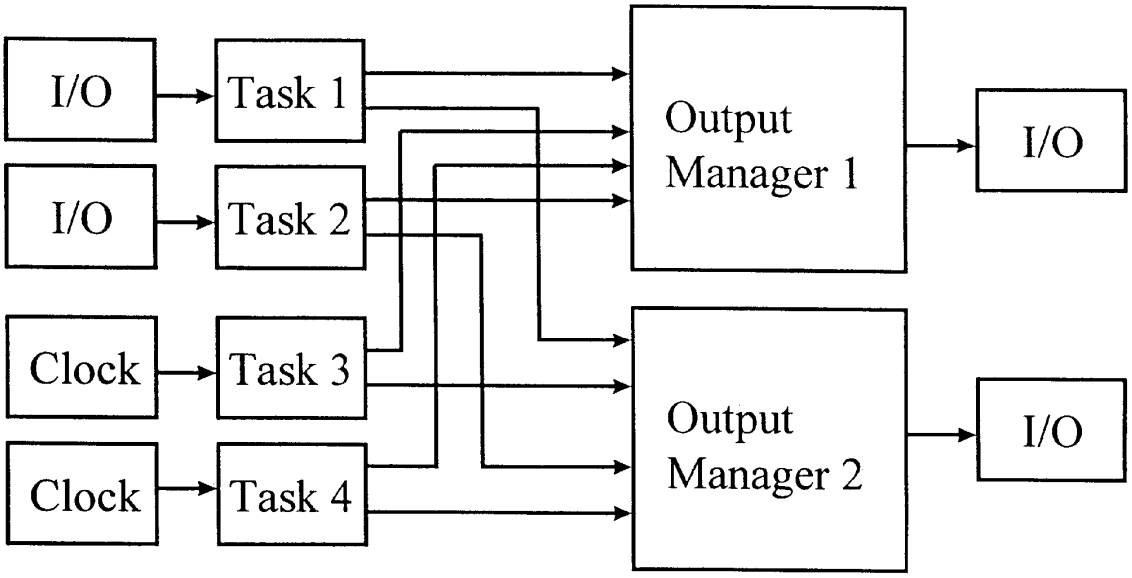
Timeline or Cyclic Executive



Event-Driven

- Characteristics:
  - Trigger schedulable tasks by I/O completion and timer events.
- Task Priority:
  - Determined by timing constraints, e.g., RMS, or by semantic importance.
- Ways to avoid synchronization is needed for predictable response.
- Processor utilization is preserved for aperiodic events for response predictability.
- Prone to event shower! Good for systems with spare computation power!

Event-Driven

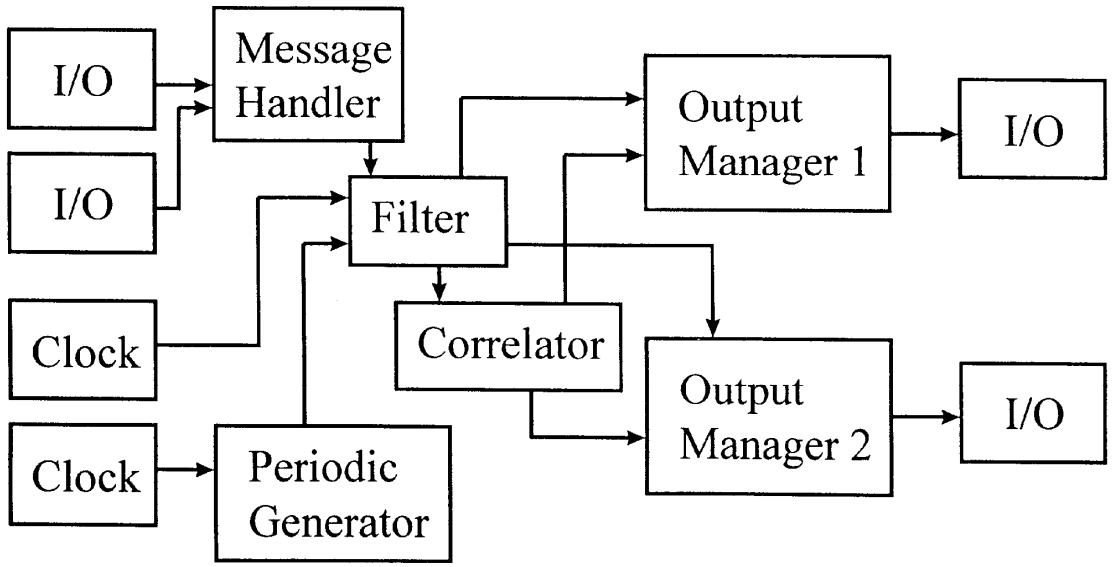


Persistent data shared within partitions

Pipelined

- Characteristics:
  - Trigger schedulable tasks by I/O completion, timer events, and inter-task messages.
  - The system can be described as a set of pipelines of task invocations.
- Task priority
  - Increasing task priorities in a unidirectional pipeline will minimize the message queue buildup.
  - Equal task priority setup is normal for bi-directional pipelines.
- Prone to event shower! Good for systems with spare computation power!

Pipelined

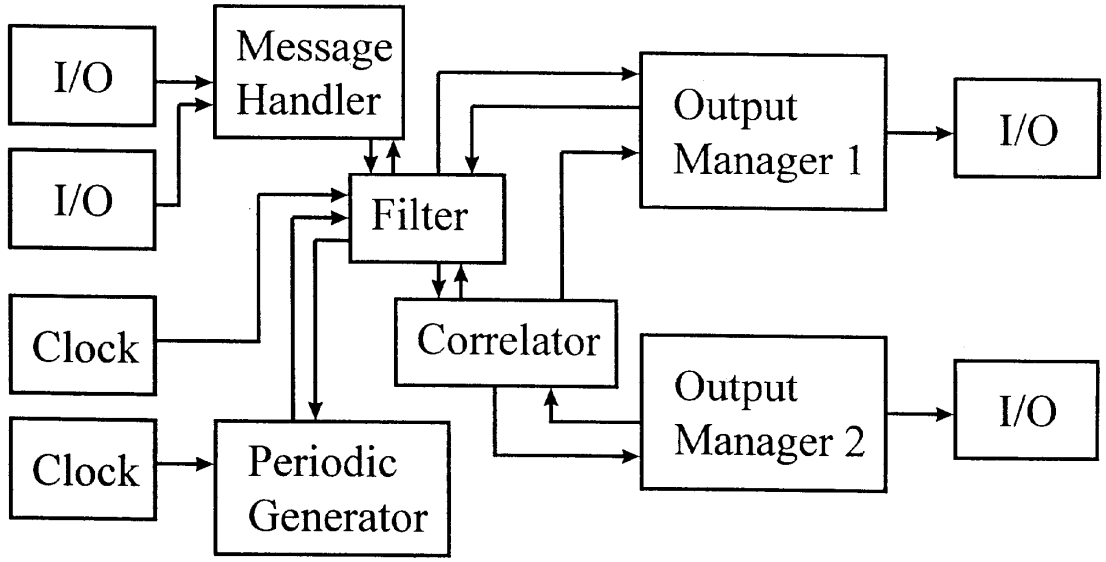


No shared data -- persistent data in Abstract Data Objects

Client-Server

- Characteristics:
  - Trigger schedulable tasks by I/O completion, timer events, and inter-task messages.
  - Control flow for an event stays at a node while data flow is distributed.
- Task priority
  - Priority inheritance is used ideally. Practically task priorities are set equally, and message priorities are used instead to avoid bottlenecks.
- More messages are exchanged but are significantly easier in debugging than pipelined systems.

Client-Sever



No shared data--persistent data in Abstract Data Objects

Fault Tolerance

- Definition:
  - A real-time fault-tolerance system is a system that can deliver its service even in the presence of faults.
- Timeliness versus Fault Tolerance
  - Possible Faults: Hardware/software errors, violation of timing constraints because of the “environment”.

Fault Tolerance

- Use redundancy to detect errors and mask failures
  - Space Redundancy: replication of physical devices.
  - Time Redundancy: repetition of a computation or communication.
  - Information Redundancy: specific encoding scheme, e.g., parity bit.

Fault Tolerance

- Real-time systems
  - Time is scare -> methods should trade space/information redundancy for time.
- Possible Structures:
  - Active replicas:
    - Each request is processed by all replicas, and their results are “combined” to mask faults.
  - Passive replicas:
    - One primary and several backups.
    - Once the primary fails, a backup takes over.
  - Cooperating replicas/objects:
    - A client makes a request through a “broker” mechanism.