Sistemi Embedded Real Time

Corso di Sistemi RT
Prof. Davide Brugali
Università degli Studi di Bergamo
Embedded systems

- Systems which monitor and control their environment:
  - **Sensors**: Collect data from the system environment
  - **Actuators**: Change (in some way) the system's environment
CRUISE CONTROL

- Regulate the speed of a car by adjusting the throttle:
  Input by the driver → sets a speed and car maintains it.

- Measures the speed through device connected to the motor shaft

- Disturbances: road surface and slope, wind
Real-Time Embedded control

- The responsiveness of the system (Car) can be measured by its \textit{rise time} $R$.
- $R$ is the time it takes to get close to its final state after the input changes.
- A good rule of thumb is the \textit{ratio} $R/T$ of rise time to sampling period is from 10 to 20.
- The faster a system respond to changes, the shorter the sampling period is.
Set timer to interrupt periodically with **Period T**; at each interrupt, do; 
**read** data from Sensor; 
**compute** control output u; 
**write** command to Actuator; 
end do;

### Diagram

- **Desired speed**
- **Sensor**
- **Speed Controller**
- **Actuator**
- **Car**
- **Road slope, surface**
- **Actual speed**

**SPORADICALLY**

**PERIODICALLY**
Real-Time Embedded control

Desired speed

Speed Controller

Actual speed

CPU

read
comp
write

Road slope, surface

Actuator

Sensor

PeriODICALLY

SPORADICALLY

T

R
Real-Time Embedded control

- Embedded PC with OS
- Microcontroller without OS
- Desired speed
- Speed Controller
- Actuator
- Car
- Sensor

Real Time Software

SPORADICALLY

PERIODICALLY

Road slope, surface

+ +

T R

Actual speed
CRUISE CONTROL WITH OBSTACLE DETECTION

- Regulate the speed of a car by adjusting the throttle:
  - Input by the driver → sets a speed and car maintains it.
- Measures the speed through device connected to the motor shaft
- Disturbances: road surface and slope, wind
- Automatically regulates the speed according to the distance to the preceding car
Concurrent Real-Time Embedded control

- Distance Monitor
- Speed Controller
- Actuator
- Sensor
- Environment

Desired speed + + -

Obstacle distance
PERIODICALLY T2
Road slope, surface

Actual speed

CPU

communication

PERIODICALLY T1
Distributed Real-Time Embedded control

CPU 1
Middleware
CPU 2

Distance Monitor
Obstacle distance
Sensor
Environment

PERIODICALLY T2
Road slope, surface

Desired speed

Speed Controller
Actuator
Car

PERIODICALLY T1

Desired speed + + -

Sensor

Network

Actual speed + +
Concurrent Real-time Applications

Real-time Application

<table>
<thead>
<tr>
<th>T1</th>
<th>T2</th>
<th>T3</th>
<th>T4</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>4</td>
<td>8</td>
<td></td>
</tr>
</tbody>
</table>

Interrupt Handler 1
Interrupt Handler 2

Hardware

NO
RT-Operating System
Cyclic Executive & Interrupt Handlers

RT-Operating System
Scheduler

Real-time Application

Task 1
Task 2

Real-time Operating System

Hardware
Distributed Real-time Applications

Real-time Application

\[ T_1 \quad T_3 \quad T_2 \quad T_1 \quad T_4 \]

0 \quad 4 \quad 8

Interrupt Handler 1
Interrupt Handler 2

Hardware

Real-time Application

Task 1
Task 2

Real-time Operating System

Hardware

CAN Transceiver
Distributed Real-time Applications

Real-time Application

Task 1

Task 2

Middleware

Real-time Operating System

Hardware

Data / Events

Ethernet / Ethercat

Real-time Application

Task 1

Task 2

Middleware

Real-time Operating System

Hardware
Fonti di non determinismo

- Architettura dell’eleboratore
  - Cache, pipeline, DMA

- Sistema operativo
  - Memoria virtuale, scheduling, comunicazione

- Linguaggio di programmazione
  - Gestione del tempo

- Metodologia di progetto
  - Assenza di tecniche di analisi e verifica
Real-Time Operating Systems

- Timeliness
  - Achieved through proper scheduling algorithms
    - Core of an RTOS!

- Predictability
  - Affected by several issues
    - Characteristics of the processor (pipelining, cache, DMA, ...)
    - I/O & interrupts
    - Synchronization & IPC
    - Architecture
    - Memory management
    - Applications
    - Scheduling!
Achieving Predictability: DMA

- **Direct Memory Access**
  - To transfer data between a device and the main memory
  - Problem: I/O device and CPU share the same bus

2 possible solutions:

- **Cycle stealing**
  - The DMA steals a CPU memory cycle to execute a data transfer
  - The CPU waits until the transfer is completed
  - Source of non-determinism!

- **Time-slice method**
  - Each memory cycle is split in two adjacent time slots
    - One for the CPU
    - One for the DMA
  - More costly, but more predictable!
Achieving Predictability: Cache

To obtain a high predictability it is better to have processors **without cache**

Source of non-determinism

• cache miss vs. cache hit
• writing vs. reading
Achieving Predictability: Interrupts

One of the biggest problem for predictability

• Typical device driver
  <enable device interrupt>
  <wait for interrupt>
  <transfer data>

• In most OS
  • interrupts served with respect to fixed priority scheme
  • interrupts have higher priorities than processes
  • How much is the delay introduced by interrupts?
    • How many interrupts occur during a task?

• problem in real-time systems
  • processes may be of higher importance than I/O operation!
Interrupts: First Solution Attempt

Disable all interrupts, but timer interrupts

 Advantages
• All peripheral devices have to be handled by tasks
• Data transfer by polling
• Great flexibility, time for data transfers can be estimated precisely
• No change of kernel needed when adding devices

 Problems
• Degradation of processor performance (busy wait)
• Task must know low level details of the drive
Interrupts: Second Solution Attempt

- Disable all interrupts but timer interrupts, and handle devices by special, timer-activated kernel routines

**Advantages**
- unbounded delays due to interrupt driver eliminated
- periodic device routines can be estimated in advance
- hardware details encapsulated in dedicated routines

**Problems**
- degradation of processor performance (still busy waiting within I/O routines)
- more inter-process communication than first solution
- kernel has to be modified when adding devices
Interrupts: Third Solution Attempt

Enable external interrupts and reduce the drivers to the least possible size

- Driver only activates proper task to take care of device
- The task executes under direct control of OS, just like any other task
- User tasks may have higher priority than device tasks
Interrupts: Third Solution Attempt (2)

Advantages

• busy wait eliminated
• unbounded delays due to unexpected device handling dramatically reduced (not eliminated!)
• remaining unbounded overhead may be estimated relatively precisely

State of the art!
Achieving predictability: System Calls

- All system calls have to be characterized by bounded execution time
  - each kernel primitive should be preemptable!
  - non-preemptable calls could delay the execution of critical activities → system may miss hard deadline
Achieving predictability: Semaphore

- Usual semaphore mechanism not suited for real-time applications
- Priority inversion problem
- High priority task is blocked by low priority task for unbounded time

- Solution: use special protocols
  - Priority Inheritance
  - Priority ceiling
Achieving predictability: Memory Management

• Avoid non-deterministic delays
• No conventional demand paging (page fault handling!)
  • Page fault & page replacement may cause unpredictable delays
  • May use selective page locking to increase determinism
• Typically used
  • Memory segmentation
  • Static partitioning
    • if applications require similar amounts of memory
• Problems
  • flexibility reduced in dynamic environment
    • careful balancing required between predictability and flexibility
Need for Special Network Technologies

- General-purpose networks, such as Ethernet, exhibit random and nondeterministic behaviour due to the conflict arbitration methods used (e.g., CSMA/CD)
  - Results in *nondeterministic delays*, including arbitrarily long delays in the worst-case
  - Results in *packet dropping* during periods of heavy network activity

- **Fieldbus**, used in process and factory automation
  - **PROFIBUS**, used in factory and process automation
  - **CANBUS**, used in automotive
  - **EtherCAT**, Ethernet for Control Automation Technology