Real Time Systems

By

Dr. Rabih Ali Sbera

Department of Computer Engineering Techniques -3rd

2025-2026

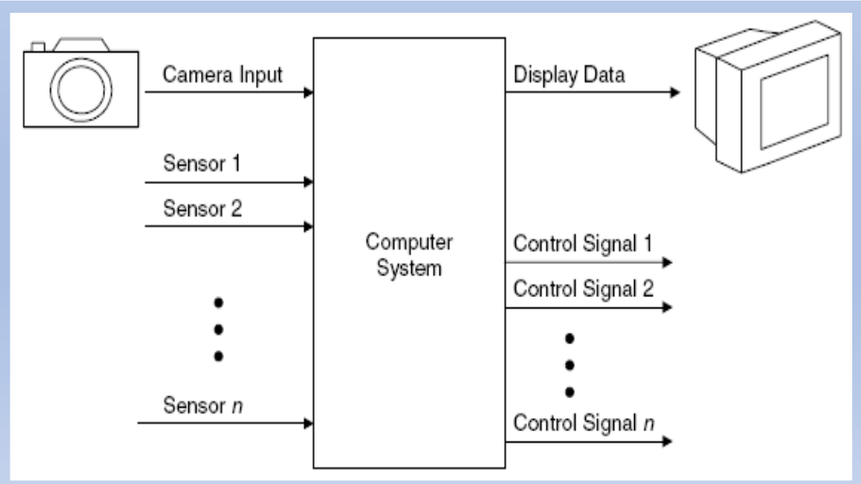


Introduction – definitions
fundamentals

Introduction

Definition of a System

A system can be modeled as any entity in nature that processes a set of inputs and produces a set of outputs. The internal details of the system may not always be important for its modeling, and the system can be expressed as a black box with multiple inputs and outputs. Computer systems consist of hardware (processor, main memory, I/O devices, etc.) and software (sets of instructions executed by the hardware). The software is divided into system software and application software. The operating system mediates between hardware and applications



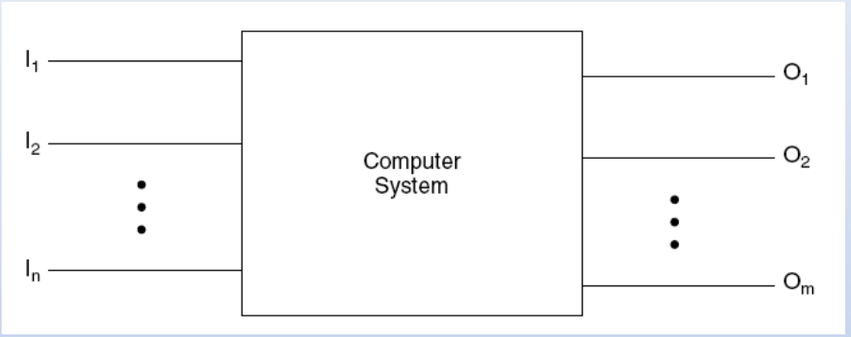




Figure 1.2. A real-time control system including inputs from a camera and multiple sensors, as well as outputs to a display and multiple actuators.

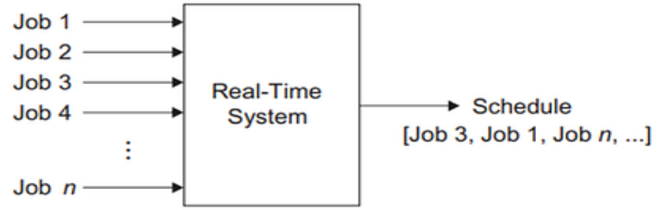


Figure 1.3. A classic representation of a real-time system as a sequence of schedulable jobs.

Introduction

Real-Time Systems are among the most important branches of embedded computer systems and specialized software, distinguished by their ability to respond to events or inputs within a predefined period known as the deadline. They are considered systems that aim not only to produce correct outputs in terms of logical values but also in terms of timing, as delays in response may lead to unacceptable or even catastrophic results.

Introduction

- **Embedded real-time systems** are extremely diverse and can even be found in **games** and **household devices**. The following table shows some examples of real-time systems and their application areas:

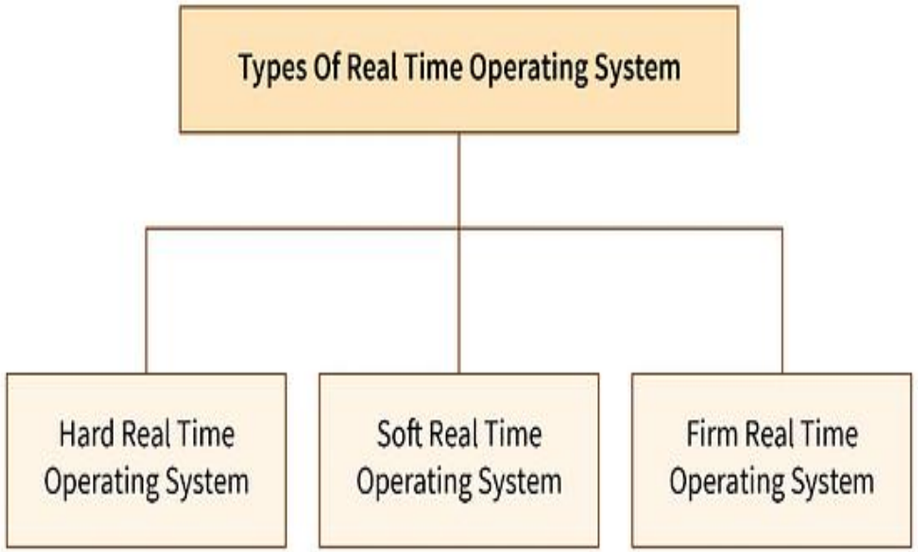
Domain	Applications
Aviation	Navigation, Display screens, Multimedia, Games, Simulation systems
Medical	Robotic surgery, Remote surgery, Medical imaging
Industrial systems	Automated assembly lines, Automated inspection
Civil systems	Elevators, Automobiles

Definitions

A real-time system is a system that must meet explicit constraints (well-defined limits) on response time.
Failure to respect these constraints can lead to serious consequences, or even to the failure of the entire system.

- **What does "system failure" mean?** In some critical cases, such as the failure of a space shuttle or a nuclear reactor, system failure is clearly evident with painful catastrophic results. However, in other cases, such as automatic teller machines (ATMs), the concept of failure is not as obvious.

Type of Real Time System



System	Real-Time System Classification	Description
ATM (Automated Teller Machine)	Soft	Exceeding the system's timing limits does not lead to catastrophic results, but only results in degraded system performance.
Embedded control system in an automated weed-removal machine	Firm	Exceeding the system's timing constraints can cause the machine to go out of control and damage useful crops instead of weeds.
Aircraft weapon control system (e.g., pressing a button to launch an air-to-air missile)	Hard	Failing to launch the missile within the specified time after pressing the button can result in missing the target and catastrophic consequences.

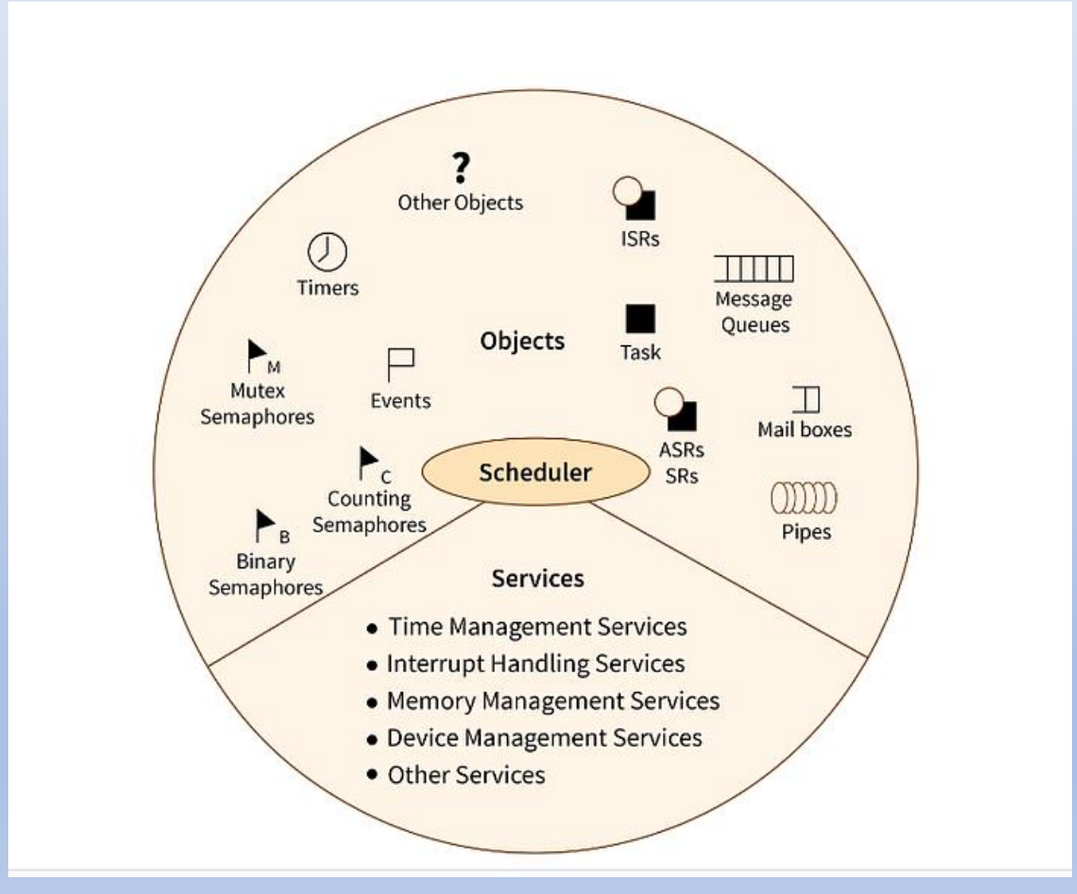
Definitions

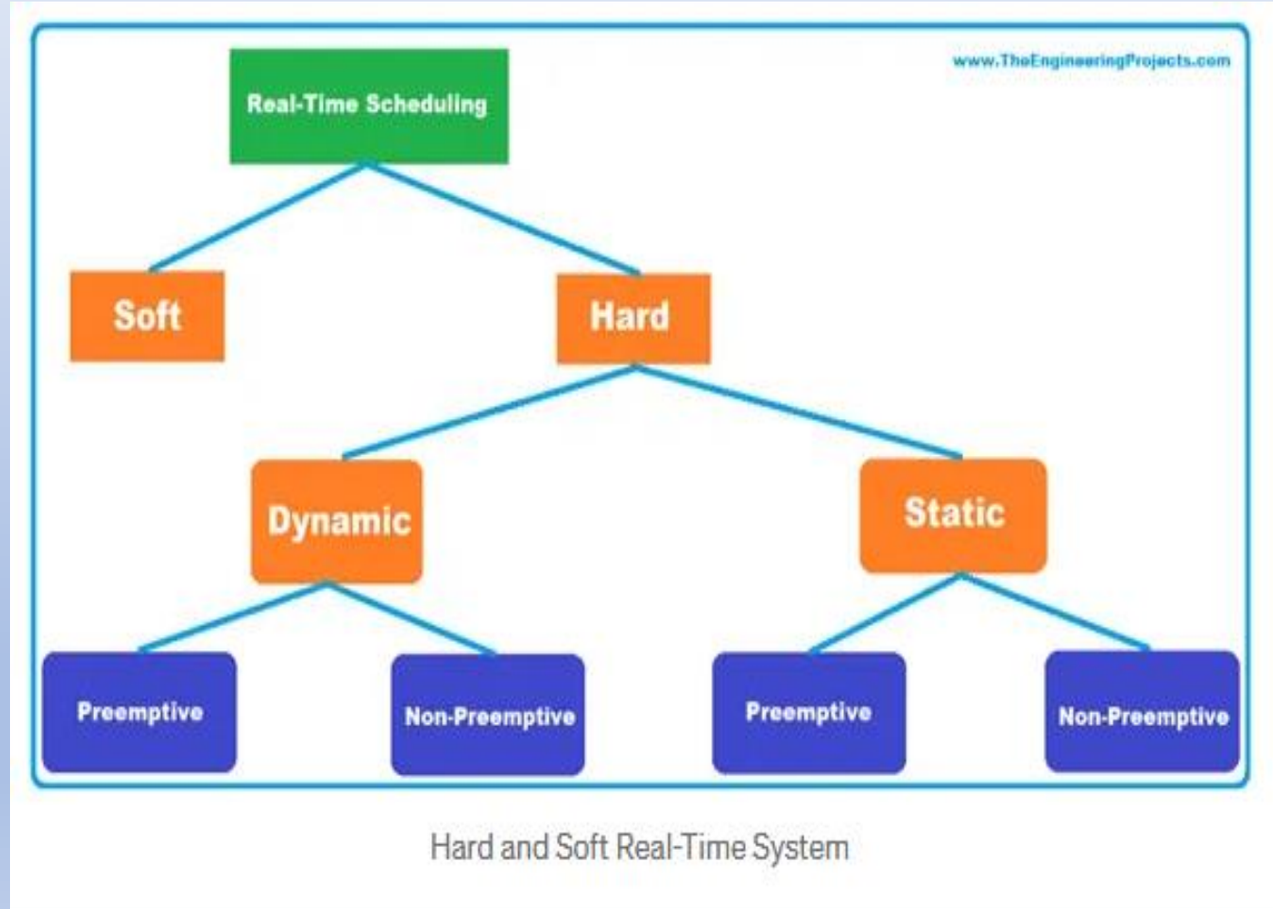
Soft Real-Time System is a system in which the importance of tasks diminishes if they exceed their timing limits, but they do not become completely worthless.

Hard Real-Time System is a system in which exceeding timing limits leads to catastrophic results and total system failure, even if it happens only once

Firm Real-Time System is a system in which exceeding its timing constraints does not lead to catastrophic results if it occurs only a few times. However, if it happens more frequently, it can result in catastrophic outcomes and complete system failure

Type of Real Time System





Hard and Soft Real-Time System

Events

Events and Determinism

- The natural way to execute instructions in any software system is **sequential execution**. The processor executing this program contains a special register usually called the **Program Counter Register**, which holds the address of the instruction currently being executed. When the program is executed sequentially, the value of this register is incremented by a fixed amount to obtain the address of the next instruction after the current instruction has been executed.

Anything that occurs and causes a non-sequential change in the program counter (which we consider a change in the program flow) is called an **Event**.

Classification of Events in Real-time systems

Real-time systems usually consist of a number of **tasks or jobs** that execute concurrently, each performing a specific action. These tasks can be **periodic** or **aperiodic** (occurring in response to an event). For example, the task that maintains an airplane's speed in flight is a **periodic task**: it periodically reads the current speed of the airplane, calculates the difference from the desired speed, and sends control commands to the airplane engines accordingly. In contrast, in a fire monitoring system, a rise in temperature triggers a **fire-extinguishing task**, which is **aperiodic**

For example, the task that maintains an airplane's speed in flight is a **periodic task**: it periodically reads the current speed of the airplane, calculates the difference from the desired speed, and sends control commands to the airplane engines accordingly. In contrast, in a fire monitoring system, a rise in temperature triggers a **fire-extinguishing task**, which is **aperiodic**.

Classification of Events in Real-time systems

The **release time** of a task is the moment when the task becomes ready for execution (whether periodic or aperiodic).

The release time is often related to an **interrupt**. This interrupt may be caused by a periodic **timer**, as in periodic tasks, or by specific events, such as a rise in temperature in a fire monitoring system.

- There are two types of events: **synchronous events** and **asynchronous events**. Synchronous events occur at predictable points in the program's execution flow, such as the conditional jump instruction shown in the flowchart above. Although the conditional jump does not always occur, it is predictable. Asynchronous events, on the other hand, occur at unpredictable times during program execution and are caused by sources external to the system.

Classification of Events in Real-time systems

Classification of Events	Examples
Periodic	Program instructions in a loop, scheduled tasks using a timer
Aperiodic	Jump instruction for handling exceptions (runtime errors)
Sporadic	Traditional jump instruction, garbage collection in modern languages
Synchronous	Internally generated timer, predictable exceptions or traps
Asynchronous	Externally generated exceptions, random events, regular interrupts without a fixed period, interrupts generated by an external timer

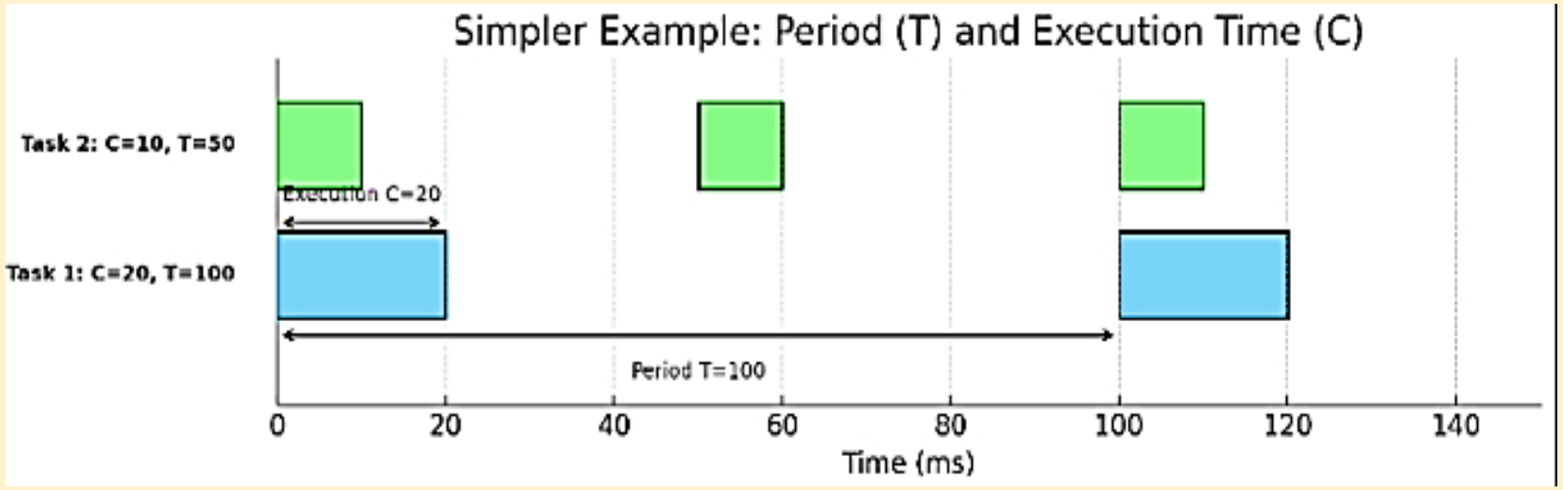
Utilization of processor

is the proportion of time during which the processor is not idle.

A system is considered **overloaded** if the value of U exceeds 100%. It is generally undesirable for a system to have high utilization, because any changes or addition of new tasks may lead to overload.

Similarly, systems with low utilization are not necessarily good, as this may indicate poor engineering design, suggesting that the system could have been implemented at a lower cost using less powerful hardware.

Typically, utilization in modern systems is around 50%, while some systems with no expected growth may have utilization close to 80%. In any case, the **optimal utilization** for real-time systems with independent, periodic tasks is considered to be 70%.



Utilization of processor

The following table shows some possible utilization values and the corresponding typical cases.

Utilization (%)	Operating Zone / Description	Typical Applications
0–25	Very underutilized – processor much stronger than needed	Multiple
26–50	Very safe	Multiple
51–68	Safe	Multiple
69	Theoretical limit	Embedded systems
70–82	Near risky	Embedded systems
83–99	Dangerous	Embedded systems
100+	Overloaded	Stressed systems

processor utilization (U) can be calculated by summing the utilization factors of the participating **periodic and non-periodic tasks**.

Utilization of processor

- Assume the system contains **n ≥ 1 periodic tasks**, each with a **period p_i**
- so that its execution rate is

$$f_i = \frac{1}{p_i}$$

- If the **estimated maximum execution time (worst-case execution time) for task i** the **utilization factor** is calculated as follows:

$$u_i = \frac{e_i}{p_i}$$

- The total utilization of the system U is equal to:

$$U = \sum_{i=1}^{n} u_i = \sum_{i=1}^{n} \frac{e_i}{p_i}$$

Example: Calculating CPU Utilization for a Periodic System

- Assume we have a system with **three periodic tasks** as follows:

Task (i)	Period pi [ms]	Worst-Case Execution Time ei[ms]
1	50	10
2	100	20
3	200	50

- **Step 1: Calculate the execution rate of each task** $f_i = \frac{1}{p_i}$

Task	fi=1/pif_
1	1/50 = 0.02
2	1/100 = 0.01
3	1/200 = 0.005

Step 2:

Task	ui
1	10 × 0.02 = 0.2 (20%)
2	20 × 0.01 = 0.2 (20%)
3	50 × 0.005 = 0.25 (25%)

Step 3: Calculate total CPU utilization

Result:

- **U = 0.65 or 65%**
- This means the processor is busy 65% of the time and idle 35% of the time. Since 65% is below the 70–82% “near risky” range, the system is in a **safe operating zone** and works efficiently without the risk of overload.