بسم الله الرحمن الرحيم

ECP-622

Embedded and Real-Time Operating Systems

Dr. Hany M. Elsayed

Lectures: Saturday 11:30 PM – 2:30 PM
HTC Room A05

Course material available at:

http://elearn.eng.cu.edu.eg

e-mail: helsayed@ieee.org

Grading:

Assignments	25%
Mid-term Exams	25%
Final Exam	50%

What is an Operating System?

An Operating System (OS) is a set of programs that:

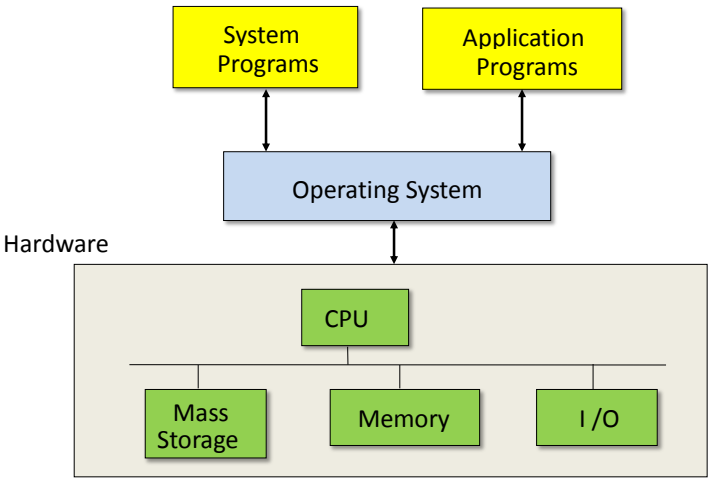
- Manages the computer hardware resources efficiently.

  and

- Provides a convenient user interface to the computer.

The operating system has a major influence on the overall function and performance of any computing system.

What is an Operating System?



The Functions of an Operating Systems

For study purposes, the functions of an operating system are usually classified into functions of:

- ❑ Process Management
- ❑ Memory Management
- ❑ I/O Management
- ❑ File System Management

In actual implementation, each of these functions is not performed separately.

Types of Operating Systems

- Embedded Systems vs. General-Purpose Systems

In an embedded system the computer is a part of a larger non-computing system. Typically system is dedicated to a particular set of tasks with timing constraints. In many cases, user has no direct access to the computer.

Embedded systems have cost, power, and performance constraints.

Since requirements of applications vary widely, a high degree of configurability is needed.

Types of Operating Systems

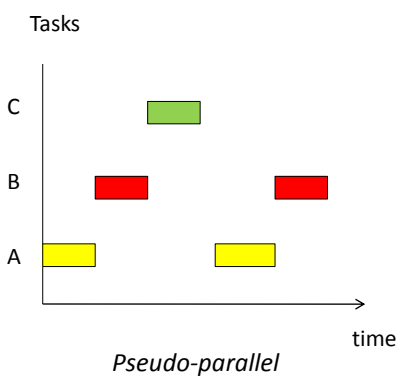
- Real-Time vs. Non-Real-Time Systems

Real-time operating systems are used to run applications with real-time requirements, i.e. with timing constraints on operation. RTOS must ensure that these constraints are met under all operating conditions.

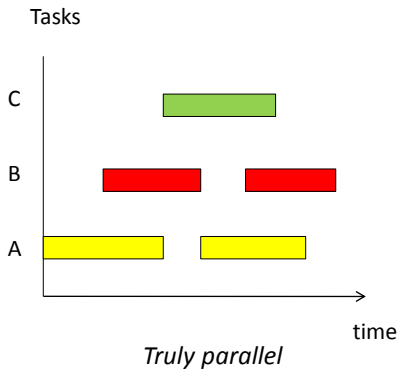
In systems with no real-time constraints, we are rather interested in the average response time and throughput of the system.

Multitasking

In a multitasking system, OS allows having many programs active at the same time.



Pseudo-parallel

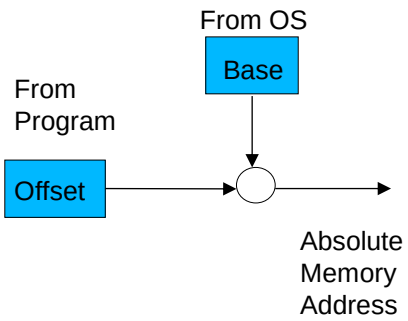


Truly parallel

Some Basic Requirements for Multitasking

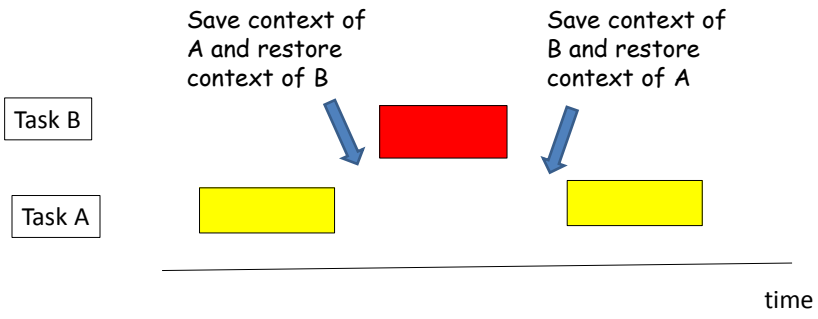
For efficient operation, the multitasking system should keep several programs in memory and switch CPU between them. All programs must thus be relocatable: i.e. can run from any available part in memory.

Thus, programs must use relative rather than absolute addresses.



Some Basic Requirements for Multitasking

While running, each program will have a *context*. This refers to all the information needed for its operation: register contents, memory pointers, …etc. Multitaskig requires continuous switching between task contexts.



Kernel Mode vs. User Mode

For proper operation, user programs should not be allowed to perform some operations:

e.g. access memory of other programs.
stop program switching.
halt the processor.

Most advanced processors have two modes of operation:

Kernel mode: all instructions can be executed, used by OS.

User mode: some instructions are prohibited.

Several privilege levels may be available, e.g. Intel processors have four levels and ARMv8 processors have seven levels.

What is a Real-Time System?

A computing system is said to be a real-time system if its correctness depends not only on the logical results of its computations, but also on the time at which these results are produced.

A real-time systems is a computing system that must perform computation within given timing constraints.

What is a Real-Time System?

- "Real-time" is not synonymous with "fast ".

  Fast computing (a relative term anyway) usually implies minimizing the average response time for a given set of tasks.

  The objective of real-time computing is to meet the individual timing requirements of each task, even under worst-case conditions.

Predictability is the most important aspect in real-time computing.

What is a Real-Time System?

Real-time systems are often "reactive" and/or "embedded" systems.

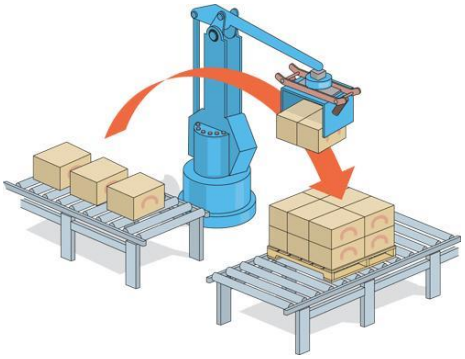
A reactive system is one that maintains constant interaction with its environment, through sensor readings, external interrupts, outputs to actuators, ...etc.

Timing constraints are thus imposed by the requirements of the external environment.

Examples of Real-Time Systems

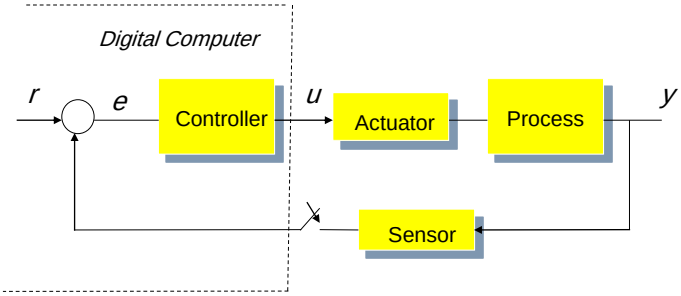
Example 1: Robot Arm

Robot arm should pick up an object from a moving conveyor belt. If the arm goes to the correct place but is late, the object won't be there anymore. If the arm arrives early, the object won't be there yet.



Examples of Real-Time Systems

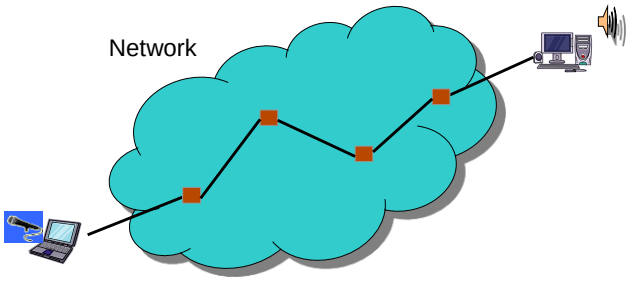
Example 2: Process Controller



If the output samples are delayed beyond a certain limit, the loop performance is degraded and system may even become unstable.

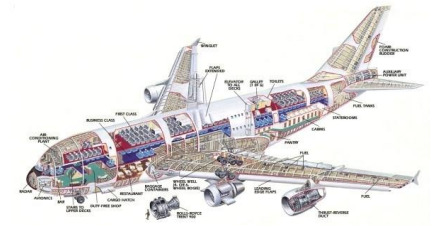
Examples of Real-Time Systems

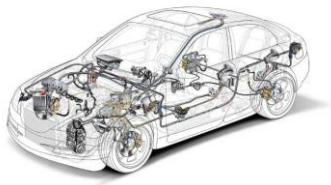
Example 3: Audio/Video Communication

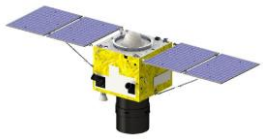


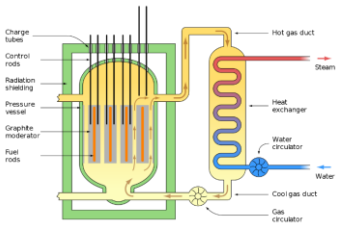
There are constraints on packet delay and/or delay variability (Quality of Service requirements).

Examples of Real-Time Systems









Hard Real-Time vs. Soft Real-Time Systems

- In a hard real-time system, failure to satisfy timing constraints causes damage to the environment and cannot be tolerated.

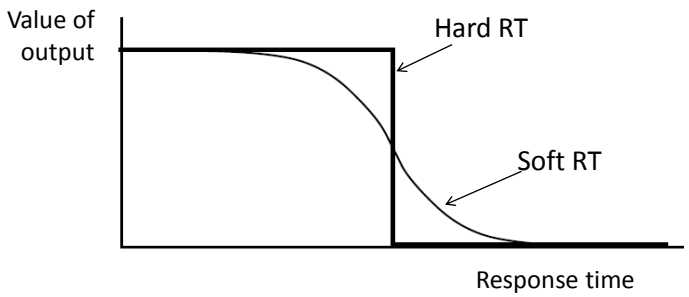
  Satisfaction of timing constraints must be guaranteed under all conditions whatever the cost may be.

- In a soft real-time system, failure to satisfy timing constraints does not cause any damage, it merely degrades performance.

  We may accept "low probability of missing a deadline".

Hard Real-Time vs. Soft Real-Time Systems

An alternative way to indicate that it is "acceptable to occasionally miss the deadline" is to define a value function:



How to Achieve Predictability?

To build a predictable system, all its components (hardware and software) should allow realizing this requirement.

- Computer Architecture

  What are the effects of such architectural features as pipelining, caching, paging, ...etc.?

- Operating Systems

  A Real-time operating system (RTOS) should manage resources such that system behaviour is predictable under all load conditions.

How to Achieve Predictability?

- Programming Languages

  An efficient real-time programming language simplifies the development of large, complex systems.

- Communication Networks

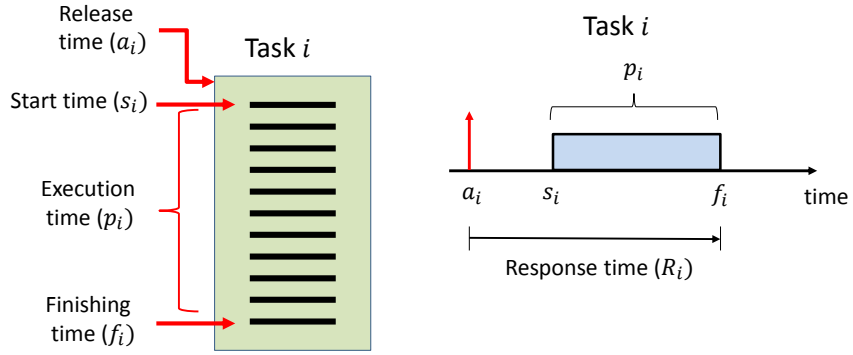
  How should protocols at different layers be modified to handle real-time traffic?

- Fault-Tolerance Methods

  Hard real-time systems are often safety critical systems that must be as reliable as possible.

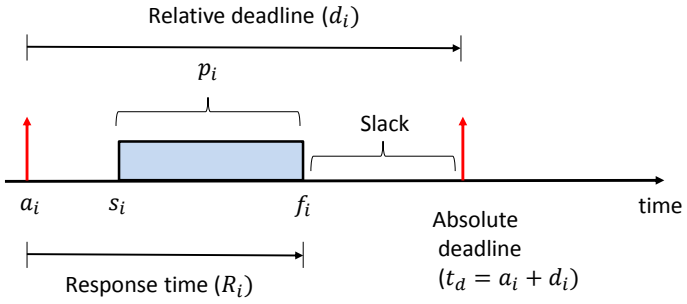
Real-Time Task Models

Tasks are the building blocks of real-time applications. Typically, many tasks will need to be active concurrently.



Real-Time Task Models

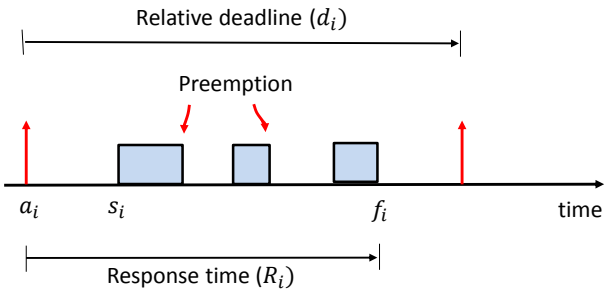
Task will have timing constraints on start time, response time, … etc. A common form is an upper bound on the response time (deadline).



Real-Time Task Models

There may also be some precedence constraints: e.g. Task B can only be released after Task A is finished.

Preemption is a mechanism allowing the operating system to temporarily suspend the execution of a running task in order to allow another task to run.



Real-Time Task Models

Preemption allows providing better response time to higher priority tasks. Its disadvantages include time wasted in context switching, cache and pipeline related timing costs, and the need to handle possible resource conflicts.

Resource constraints result when several tasks need to access some resource that can only be used by one task at a time (data, I/O device, bus, …). If a task is preempted while accessing the resource, using the resource by the preempting task can result in errors and inconsistencies.

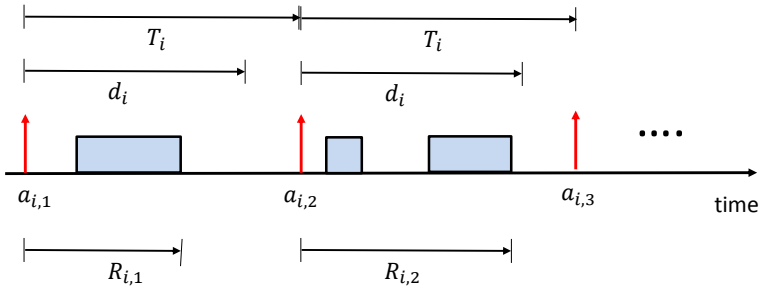
Repetitive Task Models

In most applications the same task runs several times on different inputs. The release of a new instance of the task may be either:

Event driven (aperiodic): Task is triggered by the occurrence of some event at an arbitrary time. Aperiodic task with associated deadline is referred to as a sporadic task.
Minimum time between release times is typically specified.

Time driven (periodic): An instance of task is released each period T_i (e.g. with sampling or recurrent polling of events).

Periodic Real-Time Tasks



First release time $a_{i,1} = \Phi_i$ is termed the task phase and the ratio $u_i = p_i/T_i$ is referred to as the task utilization. In the above figure there is a jitter in both start and finish times.

Worst-Case Execution Time (WCET)

To guarantee that all timing constraints are satisfied, we should be able to estimate the execution time of a given program code on a given hardware.

In general, this running time will be variable. This results from the dependence of control flow on input data, and as a result of hardware related factors.

Average execution time is an important performance measure for ordinary systems. For a real-time system the Worst-Case Execution Time (WCET) is the key measure.

For efficient system design, the estimates of the bounds on execution time should be safe and as tight as possible.

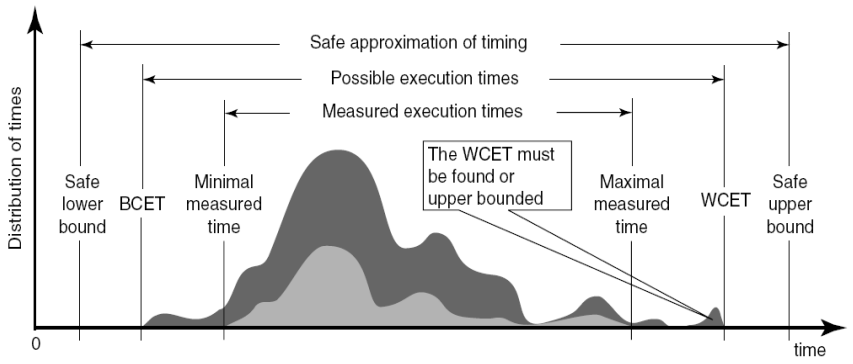
Timing by Measurement

Traditionally, estimates of execution time are obtained by direct measurements. Thus, code is run many times with different data sets, and the response times are measured (e.g. with a logic analyzer).

However, usually the number of possible execution paths is too large to allow an exhaustive testing. Without analysis there is no guarantee that worst case will be accounted for.

Analysis methods not based on actually running the code (static timing analysis) are thus preferable. To handle large applications, an automated approach will be necessary.

Worst-Case Execution Time (WCET)



Source: Ermedahl & Engblom 2008

Static WCET Analysis

The basic approach for WCET estimation requires analysis on two levels:

1) Low Level Analysis: to determine the bounds on the execution time of each basic block (branch free sequence of instructions) in the program.

Note that this analysis has to be performed on the compiled object code of a program.

This analysis should consider all the hardware details and will be complicated by advanced architectural features.

Low Level Analysis

```
z = 2*x - x*y - x^2
```


ldi r16,2	1 clock cycle
mul r16,r2	2 clock cycles
movw r5,r0	1 clock cycle
mul r2,r3	2 clock cycles
sub r5,r0	1 clock cycle
sbc r6,r1	1 clock cycle
mul r2,r2	2 clock cycles
sub r5,r0	1 clock cycle
sbc r6,r1	1 clock cycle

With a clock frequency of 8 MHz, these instructions take 1.5 μs

Static WCET Analysis

2) Program Path Analysis: Program control flow is studied to determine the possible paths which the program execution may follow, and identify the time of the worst-case path.

> Using appropriate formulas for each control structure, the execution times of individual basic blocks are combined to find time bounds of a complete task.

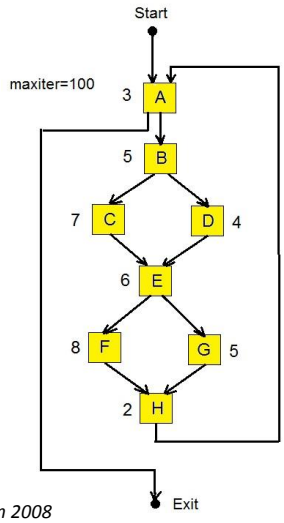
In the development of automated timing analysis tools, the two levels of analysis may be combined in different ways.

Program Path Analysis

Program structure can be described by a Control Flow Graph.

Each node represents a basic block.

An edge connects block to a block that can be executed immediately after it.



Source: Wilhelm 2008

References

- A. Silberschatz, P.B. Galvin and G. Gagne, *Operating System Concepts* , 10th Ed., John Wiley, 2018.

- I.C. Bertolotti and G. Manduchi, *Real-Time Embedded Systems: Open-Source Operating Systems Perspective*, CRC Press, 2012.