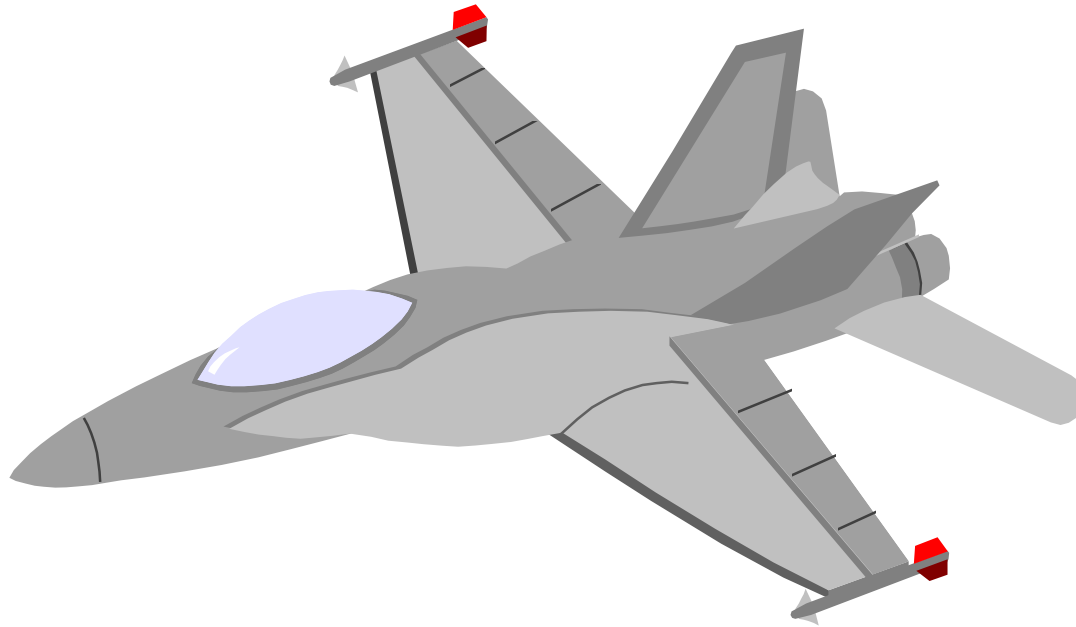


# Real Time Aspects

---



# Real Time - Introduction

---

- ◆ This section contains some considerations on Real Time Systems, their characteristics and their needs.
- ◆ The focus of attention will concentrate on how the C, C++ and Java languages can be properly used for developing Real Time Systems (I.e. what must be done and what must be avoided).

# Real Time - Some Definitions

---

- ◆ A **process** is a program in execution.
- ◆ A **processor** is a (virtual) machine able to execute processes.
- ◆ A **singleprocess** system is a system where only one process at a time is allowed to run.
- ◆ A **multiprocess** (a.k.a. multitasking) system is a system where more than one process are allowed to run at the same time.
- ◆ A **singleprocessor** system is a system with only one processor.
- ◆ A **multiprocessor** system is a system with more than one processor.

# Real Time - Some Definitions

(cont)

- ◆ Being a program in execution the process is characterised by:
  - the **code**
  - the **data** (static memory, free store and the stack)
  - the **environment** (I.e. the open files, the working directory, etc..)
  - the pointer to the instruction currently under execution.
- ◆ In a **singlethread** system each process has a single **thread** of execution.
- ◆ In a **multithread** system each process may have one or more threads of execution; in this case the threads share the environment, the code, the static memory and the free store but have different stacks and instruction pointers.

# Real Time - Some Definitions

(cont)

- ◆ Quite often **task** and **thread** refer to the same concepts, so **multitasking** is a synonym of **multithread**.
- ◆ **Multitasking** is the process of **scheduling** and **switching** the processor(s) between several tasks.
- ◆ The **kernel** is the part of a multitasking system responsible for the **management** of tasks and **communication** between tasks.
- ◆ The **scheduler** is the part of the kernel responsible for determining **which task will run next**. Most scheduler are priority based.

# Real Time - Some Definitions

(cont)

- ◆ A **Hard Real Time System** is a system where all threads must **always meet their deadlines**.
- ◆ A **(Fuzzy) Real Time System** is a system where all threads must **meet their deadlines most of the times** (I.e. with a given probability).

NOTE: no reference to the application typical time constants has been made in the above two definitions.

# Real Time - Some Definitions

(cont)

- ◆ **Non-preemptive** kernel require that each task does something to explicitly give up control to the processor. To maintain the illusion of concurrency, this process must be done frequently. Non-preemptive scheduling is also called **cooperative multitasking**; tasks cooperate wit each other to relinquish control of the processor. Cooperative multitasking is generally non deterministic.
- ◆ In one particular type of cooperative multitasking, the **cyclic multitasking**, all tasks are deterministic; all tasks are executed in a cycle and each task runs through completion.
- ◆ In a **preemptive kernel** the highest priority task ready to run is always given control of the processor. If a task does not relinquish the control it will be suspended anyhow from the scheduler. With a preemptive kernel, execution of the highest priority task is deterministic.

# Real Time - Some Definitions

(cont)

- ◆ A **deadlock**, also called deadly embrace, is a situation in which two (or more) tasks are unknowingly waiting for resources that are held by each other (see C and Java examples).
- ◆ **Priority inversion** is any situation in which a low priority task holds a resource while one or more higher priority tasks are ready to use it. In this situation the low priority task prevents the higher priority tasks from executing until it releases the resource. To correct this problem, the priority of the low priority task can be raised while accessing the resource and restored to its initial value when done.

# Real Time - Reentrancy

---

- ◆ A **reentrant function** is a function that can be safely used in a multithread system. Because each thread has its own stack, a reentrant function must allocate objects only in the stack (I.e. must have only automatic variables).
- ◆ A reentrant library is a library where all functions are reentrant.
- ◆ The **standard C/C++ libraries** are normally **not reentrant**.
- ◆ Newlib (from Cygnus) is a freely available reentrant standard C library.
- ◆ **Cyclic multithreading does not require reentrant libraries** (Why?).

# Real Time - Static vrs. Dynamic Memory Allocation

---

- ◆ Dynamic memory allocation is not recommended for real time applications because:
  - it may cause memory fragmentation problems;
  - it may induce a non deterministic behaviour in the application (I.e. sometimes the memory allocations may succeed, some other times they may fail).
- ◆ Only static and automatic memory should be used.

# Real Time - Static vrs. Dynamic Binding

---

- ◆ Dynamic binding (I.e. virtual functions) introduces some inefficiencies at function invocation time. Instead of a direct call to the function, first virtual tables are used to check which function needs to be called and then the function is called indirectly via a pointer to it.
- ◆ It has to be pointed out that such inefficiencies are very small, bigger inefficiencies may be introduced by the selection of the wrong algorithms.

# Real Time - Efficiency Considerations

---

- ◆ When efficiency is a real concern, it cannot be achieved via some optimisations techniques/tools. On the contrary, it has to be “injected” in the design. **Applications must be designed for efficiency.**
- ◆ More and more powerful processors are (going to be) introduced even in embedded applications (e.g. Java Virtual Machines embedded in Smart Cards). Because of these trends, efficiency considerations may have not the same importance they had in the past.

# Real Time - Exception Handling

---

- ◆ There are two different approaches to exceptions handling:
  - **traditional** - when an exception is raised either is handled by some portion of the code or the entire application stops (the application is reliable but not safe)
  - **alternative** - when a situation which may raise an exception occurs, recovery actions (e.g. saturated arithmetic) are taken so that the exception does not show up and the application continues its normal behaviour (the application is not reliable but safe).

# Real Time - Exception Handling (cont)

---

- ◆ The selection of the correct approach depends only on the particular application domain.
- ◆ Interrupt and exception handling is very complex in the case of multithread systems.

# Real Time - Summary

---

- ◆ Some Definitions
- ◆ Reentrancy
- ◆ Static vrs. Dynamic Memory Allocation
- ◆ Static vrs. Dynamic Binding
- ◆ Efficiency Considerations
- ◆ Exception Handling

# Real Time - Exercises

---

- ◆ 1. Have a look to the example `tcdemo.c` and try to understand what it does. Which type of multithread system is implemented by the example?
- ◆ 2. Make the example `deadlock.c` run.
- ◆ 3. Modify the example `DeadLock` by introducing different priorities (this is the Java version, but the same modifications can be applied to the C file mentioned in the previous exercise); check and comment the new behaviour.

# Real Time - Further Reading

---

- ◆ Jean J. Labrosse, “uC/OS The Real Time Kernel”, R & D Publications, 1992, ISBN 0-13-242967-5.