Real Time Operating Systems

from Fundamentals of Real Time Systems
Mukul Shirvaikar & Theodore Elbert

Chapter 4

Real Time Systems Design

- Various design approaches implemented by system designers to meet real-time requirements
- Three general approaches to task scheduling:
  - *ad-hoc scheduling*
  - *deterministic scheduling* using a cyclic executive
  - non-deterministic, *priority-driven scheduling* using a multitasking executive
- Evolved over time with successively more sophisticated approaches culminating in a full-fledged real-time operating system

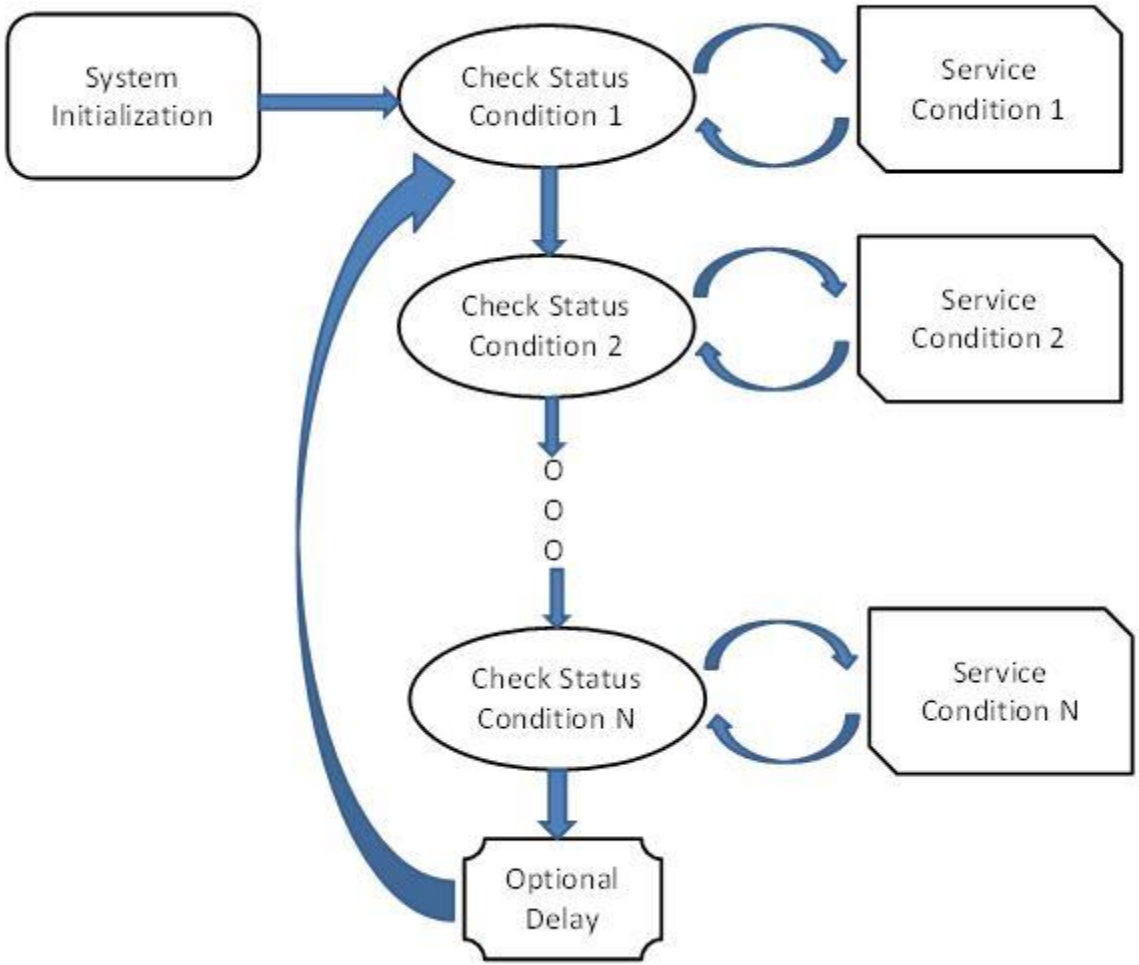
Basic Solutions

- *Ad-hoc scheduling* is the simplest form of task scheduling, if it can be called task scheduling at all
- For straightforward processing with no specific periodic requirements, it provides satisfactory results
  - Tasks are functional program units
  - Dependencies are limited to precedence relationships
  - Not periodic since repetition is not at precise rate
- Program may be essentially written as an endless loop with each task executing to completion in some predetermined sequence

Polled-Loop Systems

- Simplest form of Real-time “kernel”.
- Used for fast reaction to single events
- Do not require interrupts.
- Poll devices attached to the system continuously
- Keeps checking for a service request event, which can be a flag or signal in the software
- Event flag is typically cleared upon servicing so that the system is ready for the next event or “*burst*”

Polled-Loop Systems



Capture characters

Process User Commands

Process Recipe

Polled-Loop Systems

- Simple while loop that executes multiple ‘tasks’.
- Tasks must not block.
- Tasks must have a known maximum execution time.

```
loop
{                       /* do forever      */
  if (packet_here)      /* check flag      */
  {
    packet_here = 0;/* reset flag       */
    process_data(); /* process data */
  }
  if (timer_event)      /* check flag      */
  {
    readCCR();          /* resets timer event flag */
    process_event();/* process event */
  }
}
```

Polled-Loop Systems

- Dedicated to the status loop, thereby making it impossible to do other tasks or even enter a "*power-saving*" or sleep mode

- Not possible to guarantee that the peripherals will be serviced in the correct order or priority level

Finite State Machines

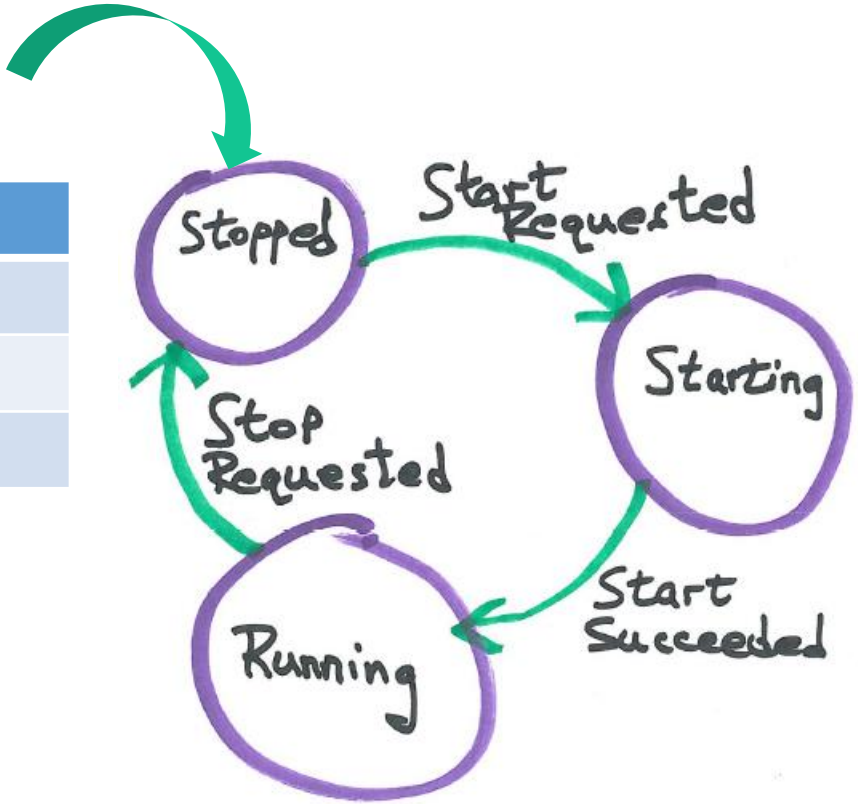
- Allows the system to transition between different states
- System tasks divided into sub-tasks and then a state associated with each one
- Once a sub-task is completed, the program changes the state
- Relinquishes control to the main loop which then decides the next sub-task to perform based upon a switch statement associated with the designated state

Finite State Machines

- States: a set of condition for a machine that exists for a non-trivial amount of time. The meaning of non-trivial is project dependent.
- Events: a set of actions that occur effectively instantly in the context of the project
- FSMs can be used in conjunction with interrupts or polling to form fairly sophisticated systems

Finite State Machines Example

States	Events
Stopped	Start button pressed
Starting	Start succeeded
Running	Stop button pressed



What event(s) are missing from this example?

The Multitasking Executive (RTOS)

- Embedded systems have a singular purpose.
- To accomplish this purpose, they may require running multiple tasks simultaneously.
- Tasks must not block, they are called upon repeatedly, they must remember their state.
- Some tasks are more important than others.
- Tasks may need to communicate with each other.
- An RTOS provides the framework to make this all happen.

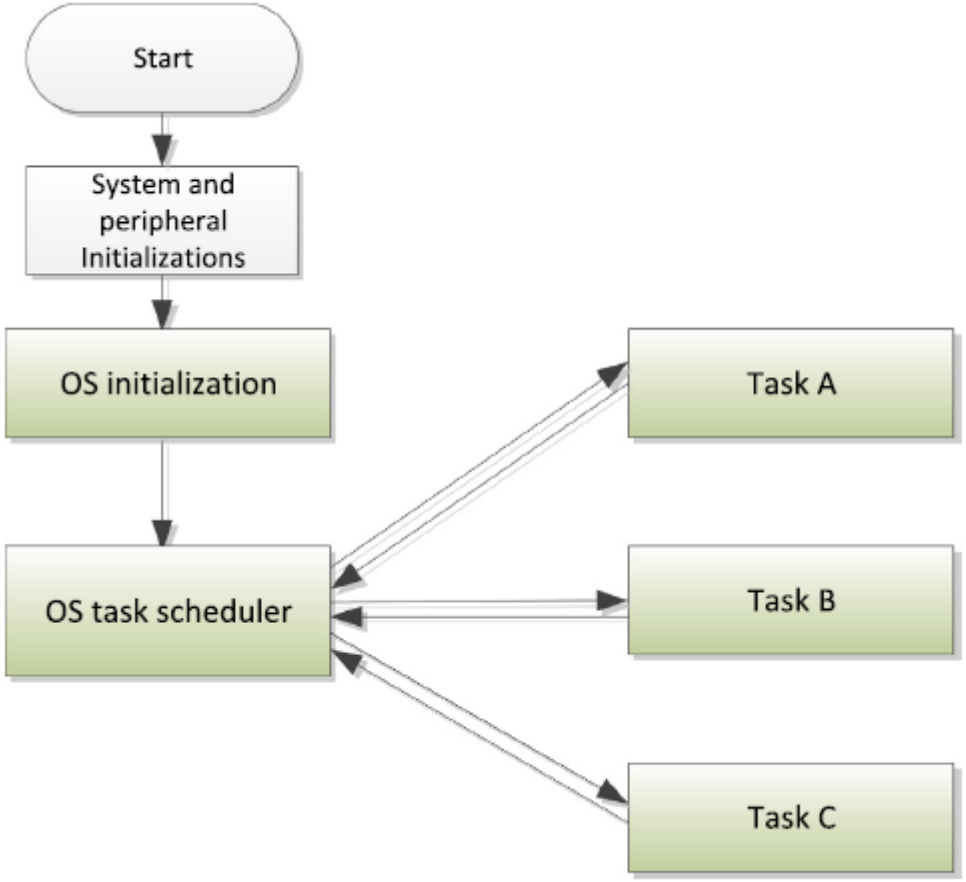
Process vs. Task vs. Thread

- In contemporary operating systems,
  - **Application** is a separately loadable program.
  - A task or process is a running application.
    (Windows task manager lets you manage processes)
- In an embedded system
  - **Process** is an executing program. A process has its own dedicated memory space.
  - **Threads** are semi-independent tasks that operate under a process. Threads share the memory space of the process. Threads can share data structures and variables.
  - Task ≈ Thread

The Multitasking Executive (RTOS)

- Sophisticated approach to the task scheduling problem is a *multitasking executive or RTOS*
- An RTOS schedules tasks in real-time based on their current priority chosen based on a scheduling strategy or metric
- The primary function of a multitasking executive is to schedule the tasks for execution in a manner that ensures the meeting of system operational requirements

The Multitasking Executive (RTOS)



The Multitasking Executive (RTOS)

- Task (or tasks if there are multiple processors) with the highest priority will be scheduled for execution unless the execution of the task is blocked
- A task is said to be *blocked* if the current state of program execution requires that it not be allowed to execute. For example, the task may be waiting for some lower priority task to provide it with needed data

The Multitasking Executive (RTOS)

- Tasks can be added or deleted or the priorities of tasks can be changed without the necessity to change the system or redesign it - great amount of flexibility
- Appropriate for systems with
  - Synchronous requirements
  - Non-deterministic task execution times (times may vary over successive executions)
  - Large asynchronous components

The Multitasking Executive (RTOS)

- The core functional component is a software component called a *multitasking executive*
- In addition to task scheduling it provides other services
  - task synchronization (primitive operations, such as semaphores, and queues)
  - inter-task communication (signals and mailboxes)
  - memory management
  - Modules: device drivers, file system support, networking, protocols

The Multitasking Executive (RTOS)

- Task synchronization involves suspension and resumption of a task in accordance with the status of other tasks – known as *context switching*
- The *context* is the minimal information about a task that must be saved before suspension of the task so that it may be resumed at a later instant (register contents, I/O, memory management, etc.)

The Multitasking Executive (RTOS)

- Stored in protected memory in a data structure called the Task Control Block (TCB) or Thread Definition Structure (TDS)
- Context switching takes time that is overhead from the viewpoint of the application program. The amount of time required for context switching is one of important benchmarks used to judge the efficiency of a multitasking executive

The Multitasking Executive (RTOS)

- *Concurrent execution* - only an impression provided to the user that multiple tasks are running at the same time, because a single processor can only implement one task at any instant of time

- *Parallel execution* - multi-core or multi-processor configurations and can actually execute tasks in parallel (still concurrent on each individual processor)

The Multitasking Executive (RTOS)

- *Concurrent execution* - Multiple tasks can be "*active*" at any instant of time and execute sequentially using a simple *time-slicing* scheme (also known as *round-robin* scheduling)

- Alternatively, such context switching can also be based on preemption due to priorities or resources.

The Concurrently Executing Task

- RTOS must provide for the creation, deletion, preemption and monitoring of tasks or threads

- The term "*thread*" is increasingly used instead of the term "*task*" to describe a distinct sequence of operations, since the use of multicore processors has become commonplace over the last few years

The Concurrently Executing Task

- Storage space requirements for the context of a task must be specified and provision must be made for multiple instances of the same task
- Other properties such as the task identifier and the task priority must also be maintained
- It is convenient to maintain such information about a task or thread in a data structure or object - *task control block*, or TCB or Thread Definition Structure (TDS).

The Concurrently Executing Task

TCB - Task Control Block

```
struct osTaskControlBlock {
  string TASK_NAME;
  uint32_t priorty;
  char *register_save;
  uint32_t STATE;
  time_t ACTIVATION_TIME;
}
```

TASK_NAME – the name of the task that acts as an identifier for starting the task, suspending the task, or performing some other operation affecting the task

ACTIVATION_TIME – the time instance in the future when the task will be activated

REGISTER SAVE area – the area where the processor register contents are stored upon task suspension

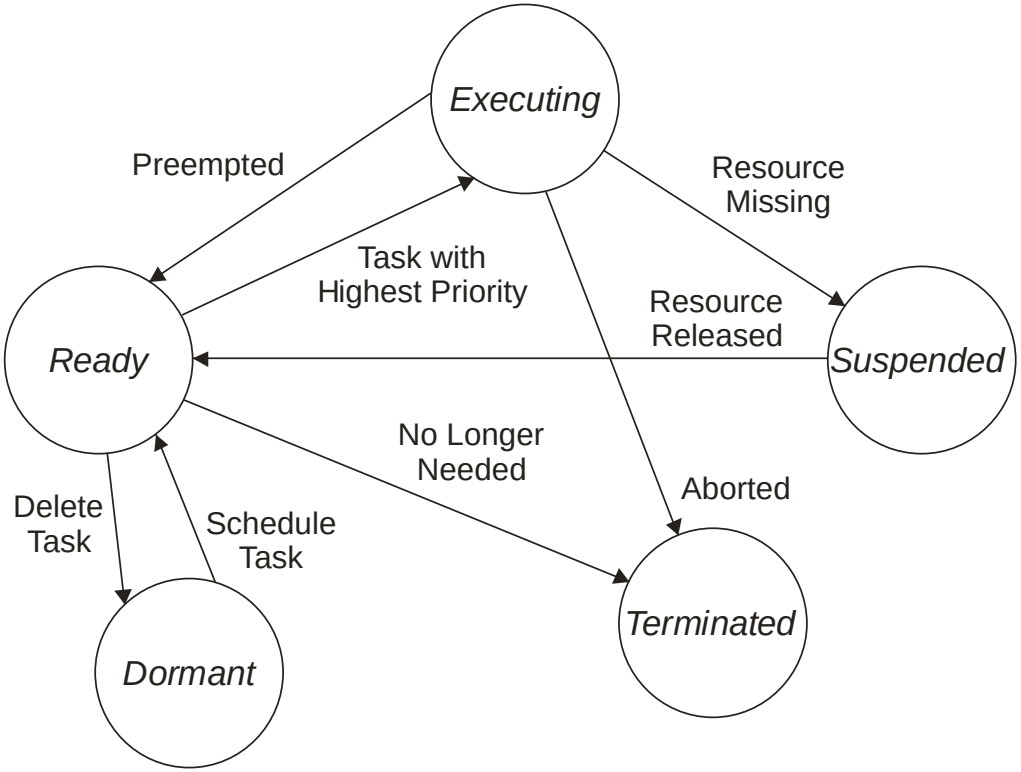
STATE – the state of the task typically

- *running* (currently has the necessary resources and is executing)
- *suspended* (currently blocked from execution awaiting action)
- *ready*, (not blocked from execution, but waiting for resources necessary for execution)

PRIORITY – the priority of the task which may remain fixed or change during the system operation based on static or dynamic priority assignment

The Concurrently Executing Task

- A process state diagram as a partially defined finite state machine



Mutex

- Threads operating in the same memory space have access to the same data structures & resources.
- A **mutex** (mutual exclusion object) is an software object used to control access to shared resources.
- A mutex is created for each resource requiring exclusive access.
- A thread requests access to the shared resource before using it (it may block)
- The thread releases the mutex before moving on
- A mutex is non-recursive

The System Timer

- RTOS - one of the most important tasks is to maintain accurate time (not world time for which there are special chips)
- Rather it is a measure of elapsed time between events
- Special timer sub-system that can be set to interrupt the operating system at regular intervals (timer interrupt based on clock)

The System Timer

- Typically this value can range from 1ms to 100ms (commonly 10ms)
- The clock tick interrupt can be viewed as the system's heartbeat
- Granularity – system need vs. overhead
- The "*real-time clock*" – supported in hardware by a peripheral - on or off chip

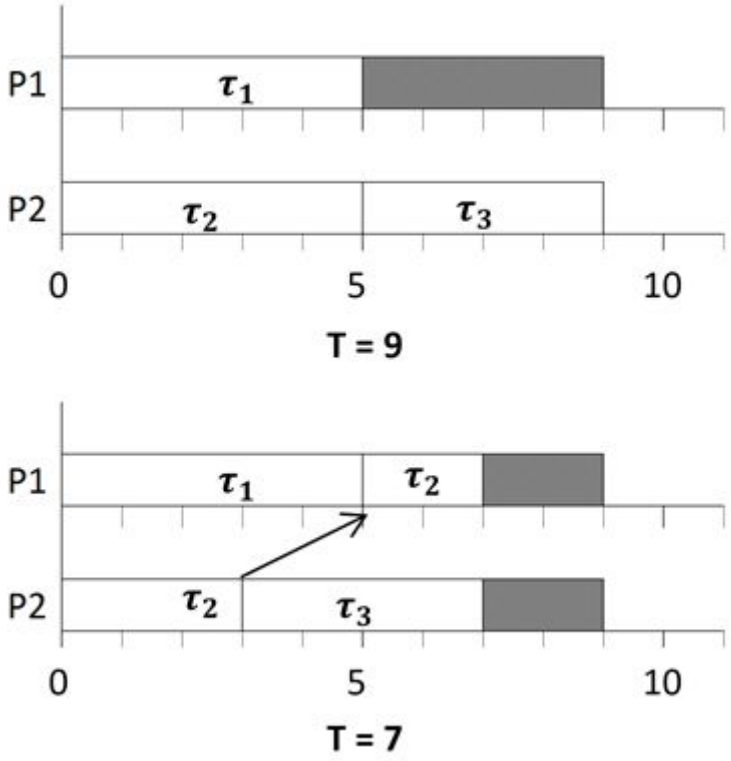
The System Timer

- Most of the modern ARM processor cores
  - have an additional component on them known as the "*SysTick*" timer. *Does STM32L4 have one??*
  - Implemented as simple increment or decrement counters that produce a hardware interrupt or system exception that must be handled by the CPU
  - RTOS treats this exception as high-priority and the "*scheduler*" part of the RTOS is invoked at this time
  - It makes decisions about system level issues such as which task should run in the next time slice

Preemptive Scheduling

- Task "*preemption*" and is one of the powerful tools that allows the RTOS to meet deadlines imposed by scheduling constraints
- Preemption results in the suspension of the currently executing task in order to permit another task to execute (after context switching of course)

Preemptive Scheduling



Preemptive Scheduling

- Static and dynamic task scheduling algorithms do not have guaranteed closed form solutions under most normal operating conditions and can sometimes lead to unpredictable behavior
- As systems get larger with many interdependent parts, it is increasingly difficult to do so, leading to the increasingly prevalent use of dynamic scheduling with techniques based on preemption

Precedence Constraints

- One task must complete execution before one or more of the other tasks can begin execution
- Add a layer of complexity to the scheduling process or algorithms
  - deterministic schedules - precedence constraints help to satisfy task synchronization requirements
  - non-deterministic schedules - precedence constraints implemented with inter-task communication

Choosing a RTOS

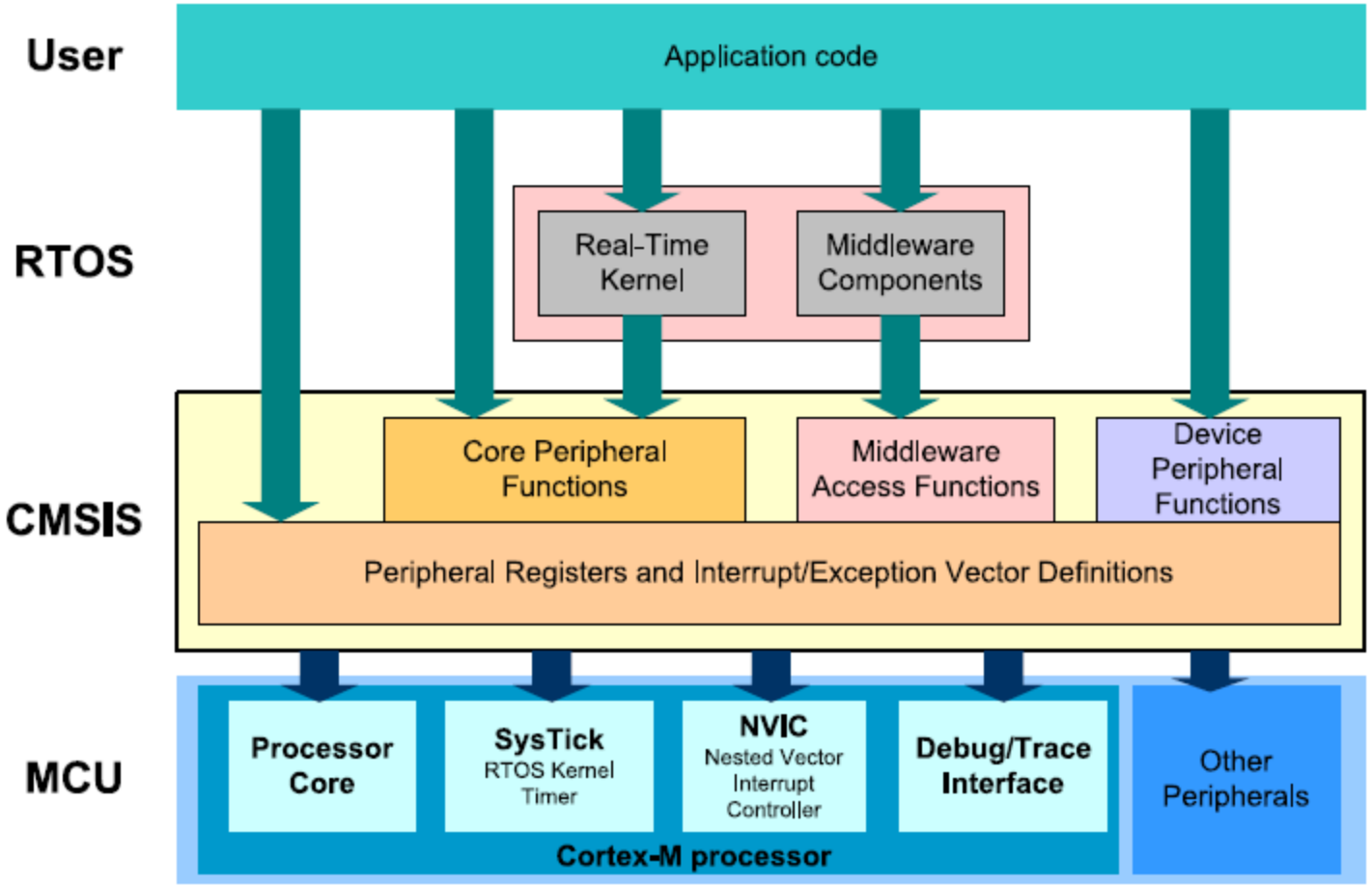
- The choice of the RTOS is one of the most important decisions made by the system designers. It is very important to review RTOS features before arriving at a choice since it is very difficult to change it once it has been made. This is especially true for products and companies that must support their software through several generations of the product. Software can be difficult to port and test on a different RTOS in the future.

Commercial RTOS

Variety of flavors and pricing options

- FreeRTOS.org
- POSIX (IEEE Standard)
- AMX (KADAK)
- C Executive (JMI Software)
- RTX (CMX Systems)
- eCos (Red Hat)
- INTEGRITY (Green Hills Software)
- LynxOS (LynuxWorks)
- µC/OS-II (Micrium)
- Neutrino (QNX Software Systems)
- Nucleus (Mentor Graphics)
- RTOS-32 (OnTime Software)
- OS-9 (Microware)
- OSE (OSE Systems)
- pSOSystem (Wind River)
- QNX (QNX Software Systems)
- Quadros (RTXC)
- RTEMS (OAR)
- ThreadX (Express Logic)
- Linux/RT (TimeSys)
- VRTX (Mentor Graphics)
- VxWorks (Wind River)

CMSIS Model (RTX)



CMSIS = Cortex Microcontroller Software Interface Standard