# Embedded Real-Time Operating Systems

*Instructor:Dr. Ankur Beohar*
*School of Electrical and Electronics Engineering*
*VIT Bhopal University*

# Embedded Systems

- Suggested Textbooks:
  - Raj Kamal, “Embedded systems Architecture, Programming and Design”, Third Edition, Tata McGraw Hill, 2017.
  - Rob Toulson and Tim Wilmshurst, “Fast and Effective Embedded Systems Design – Applying the ARM mbed”, Elsevier, 2017.
  - Arnold S. Berger, “Embedded Systems Design: An Introduction to Processes, Tools, and Techniques”, CRC Press, 2002.
- Other sources
  - Lecture notes
  - Handouts
  - Blogs
  - MOOC courses

# Operating Systems (OS)

- An **Operating System (OS)** is a collection of programs that provides an interface between application programs and the computer system (hardware).
- Its primary function is to provide application programmers with an abstraction of the system resources, such as memory, input-output and processor, which enhances the convenience, efficiency and correctness of their use.
- An OS typically provides multitasking, synchronization, Interrupt and Event Handling, Input/Output, Inter-task Communication, Timers and Clocks and Memory Management.
- Core of the OS is the Kernel which is typically a small, highly optimised set of libraries.
- When we hear the word “Operating System” the first ones that come to our mind are those we experience/use in our day to day life, say, Windows XP, Linux, Ubuntu, Windows 7 for Computer systems, Android for mobiles and many more.

# Real-Time

- Real-Time indicates an expectant response or reaction to an event on the instant of its evolution.
- The expectant response depicts the logical correctness of the result produced.
- The instant of the events' evolution depicts deadline for producing the result.

# Real-Time Systems (RTS)

- Real-Time Systems (RTS) are those systems in which the correctness of the system depends not only on the correctness of the logical result of computation, but also on the result delivery time.
- An RTS is expected to respond in a timely, predictable way to unpredictable external stimuli.

# Real-Time Operating Systems (RTOS)

- A Real-Time OS (RTOS) is a subtype of operating system with special features that make it suitable for building realtime computing applications also referred to as *Real-Time Systems (RTS*).
- A Real-Time OS (RTOS) is used for those applications where data processing should be done in a small quantum of time.
- It is an operating system intended to serve real time applications that process data as it comes in, mostly without buffer delay.
- **RTOS** is therefore an operating system that supports real-time applications by providing logically correct result **within the deadline required**.

# Real-Time Operating Systems (RTOS)

- Real time processing requires quick transaction and characterized by supplying immediate response. *For example*, a measurement from a petroleum refinery indicating that temperature is getting too high and might demand for immediate attention to avoid an explosion.
- RTOS is key to many embedded systems and provides a platform to build applications.

# Real-Time Operating Systems (RTOS)

- All embedded systems are not designed with RTOS.
- Embedded systems with relatively simple/small hardware/code might not require an RTOS.
- Embedded systems with moderate-to-large software applications require some form of scheduling, and hence RTOS.

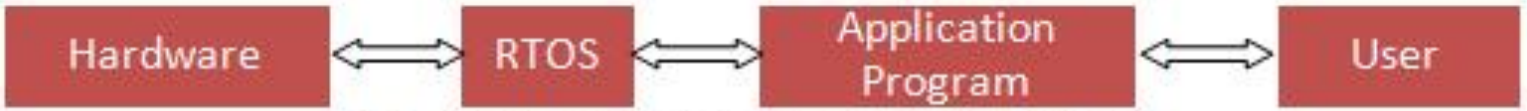



Fig. Real time embedded system with RTOS

# Applications of RTOS

- Air traffic control system
- Nuclear reactors
- Control scientific experiments
- medical imaging systems
- industrial system
- fuel injection system
- home appliances
- Airlines reservation system
- Used in any system that provides up to date and minute information on stock prices.
- Systems that provide immediate updating.
- Defense application systems like RADAR.
- Heart Pacemaker

# Some applications of RTOS

## Control systems

- RTOS are designed in such a way so that they can control actuators and sensors.
- They can execute control system commands. Controlled systems are those which can be monitored by using sensors and their tasks can be altered with the help of actuators.
- Now the task of the RTOS is to read the data from the sensors and move the actuators by performing some calculations so that flight's movement can be handled.

# Some applications of RTOS

## Image processing (IP)

- Real time image processing is done so that we can make some adjustments for the moving objects.
- In this, we need our computers, cameras or other gadgets should work in real time as utmost precision is required in the industrial automation tasks.
- For ex, something happens with the conveyor belt when the item is moving downwards or some other defect occurs, we can control these problems in the real time if our system works in real time.

# GPOS v/s RTOS

# GPOS v/s RTOS

| General-Purpose Operating System (GPOS) | Real-Time Operating System (RTOS) |
|---|---|
| Used for Desktop PCs and laptops. | Used for embedded applications. |
| OS's typically provide a non-deterministic response, where there are no guarantees as to when each task will complete, but they will try to stay responsive to the user. | It typically provides a highly deterministic reaction to external events i.e. consume only known and expected amounts of time. |
| No priority inversion mechanism is present in the system. | RTOS have mechanisms to prevent priority inversion. Once priority is set by the programmer, it can't be changed by the system itself. |
| A GPOS kernel is not preemptible. Without a preemptible kernel, a request from within the kernel, such as that from a driver or a system service, would override all other process and threads. | In RTOS, all kernel operations are preemptible, which is important when serving high-priority processes and threads first. |

| General-Purpose Operating System (GPOS) | Real-Time Operating System (RTOS) |
|---|---|
| Task scheduling is **not based on "priority" always**. It is programmed to handle scheduling in such a way that it manages to achieve high throughput. Here *throughput* means – the total number of tasks that complete their execution per unit time. | Task scheduling is **always priority based**. Here a high priority process gets executed over the low priority ones. All low priority process execution will get paused. A high priority process execution will override only if a request comes from an even high priority process. |
| Eg. the editing of a document on a PC | Eg. operation of a precision motor control |

# RTOS Classification

- RTOS specifies a known maximum time for each of the operations that it performs. Based upon the degree of tolerance in meeting deadlines, RTOS are classified into following categories:

1) **Soft real-time/Non-real time**
2) **Firm real-time**
3) **Hard real-time**

# Soft Real-Time

- Deadlines may be missed occasionally, but system doesn't fail and also, system quality is acceptable. A limited extent of failure in meeting deadlines results in degraded performance of the system and not catastrophic failure.
- Essentially the result can still hold some value even though it occurred after the required deadline.
- The pre-emption period is usually within a few milliseconds.
- For example, for TV live broadcast, delay can be acceptable. Another example is creation of a database that is used for storage purposes.

# Firm Real-Time

- The RTOS has certain time constraints, which are not strict and may cause undesired yet acceptable effects.
- Missing a deadline might result in an unacceptable quality reduction but may not lead to failure of the complete system.
- An example is a production plant where the quality of a product diminishes because the deadline was not met. This will affect the reputation of the brand, profits, etc.

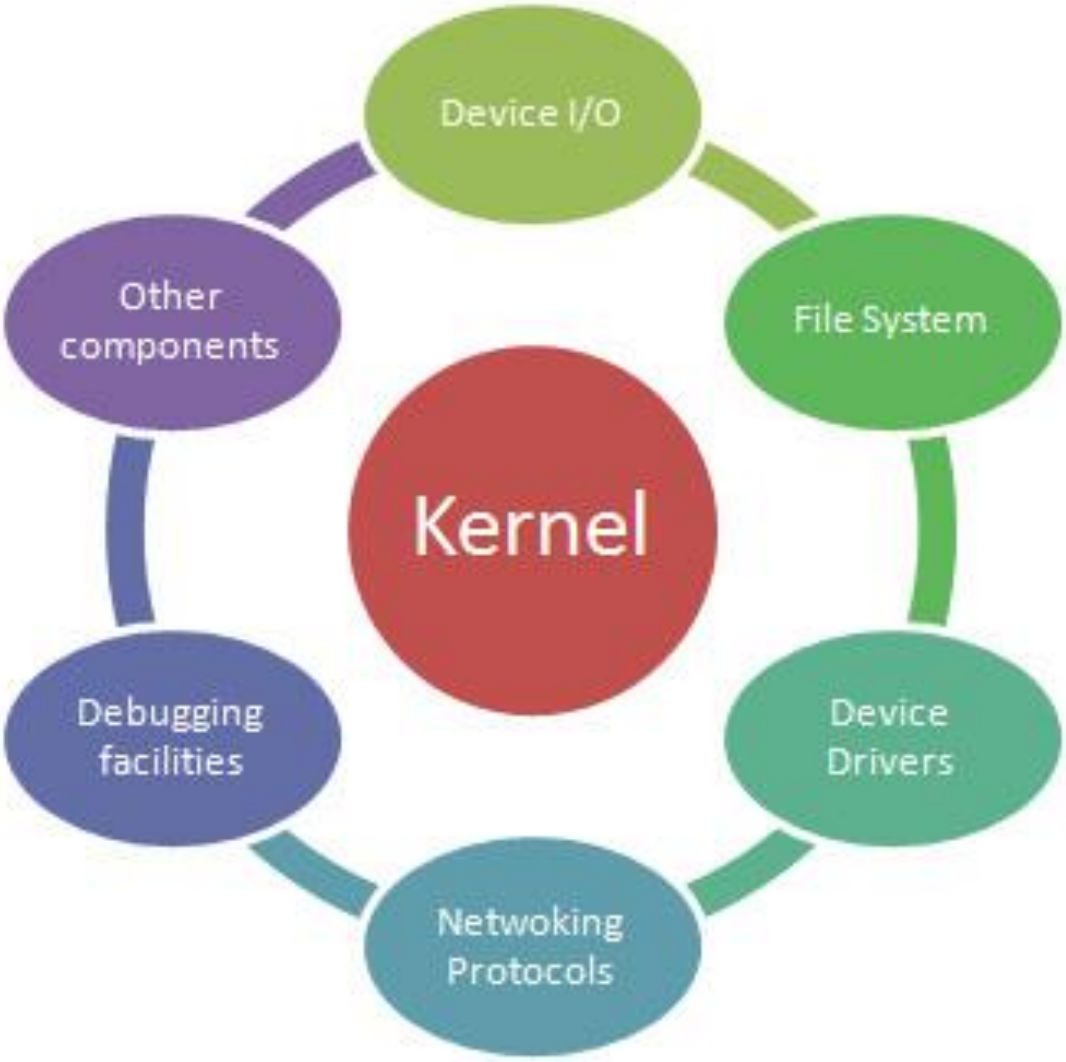
# Hard Real-Time

- The RTOS can meet timing deadlines deterministically. Degree of tolerance for missed deadlines is negligible. A missed deadline can result in catastrophic failure of the system. The system is taken to have failed if a computing deadline is not met.
- The pre-emption period for a hard RTOS is usually less than a few microseconds.
- A Hard RTOS is generally preferable for use cases involving mission critical applications such as those found in robotics and drones and also in Airbag control in cars.
- For example, an airbag has to deploy within the perfect time frame. Failure to deploy may cause death or extreme destruction. It means that the deadline was missed.
- For instance, if an output is expected within 10 seconds, the system should process the input and give out the output by the 10th second. Note that the output should not be released by the 9th or 11th second to prevent the system from failing.

# RTOS Architecture

- The architecture of an RTOS is dependent on the complexity of its deployment.
- Good RTOSs are scalable to meet different sets of requirements for different applications.
- For simple applications, an RTOS usually comprises only a kernel.
- For more complex embedded systems, an RTOS can be a combination of various modules, including the kernel, networking protocol stacks, and other components as illustrated in the Figure.

# RTOS Architecture



# RTOS Architecture

## Kernel

- Kernel is the smallest and central component of an operating system. It is the heart of every operating system.
- Its services include managing memory and devices and also to provide an interface for software applications to use the resources.
- Additional services such as managing protection of programs and multitasking may be included depending on architecture of operating system.

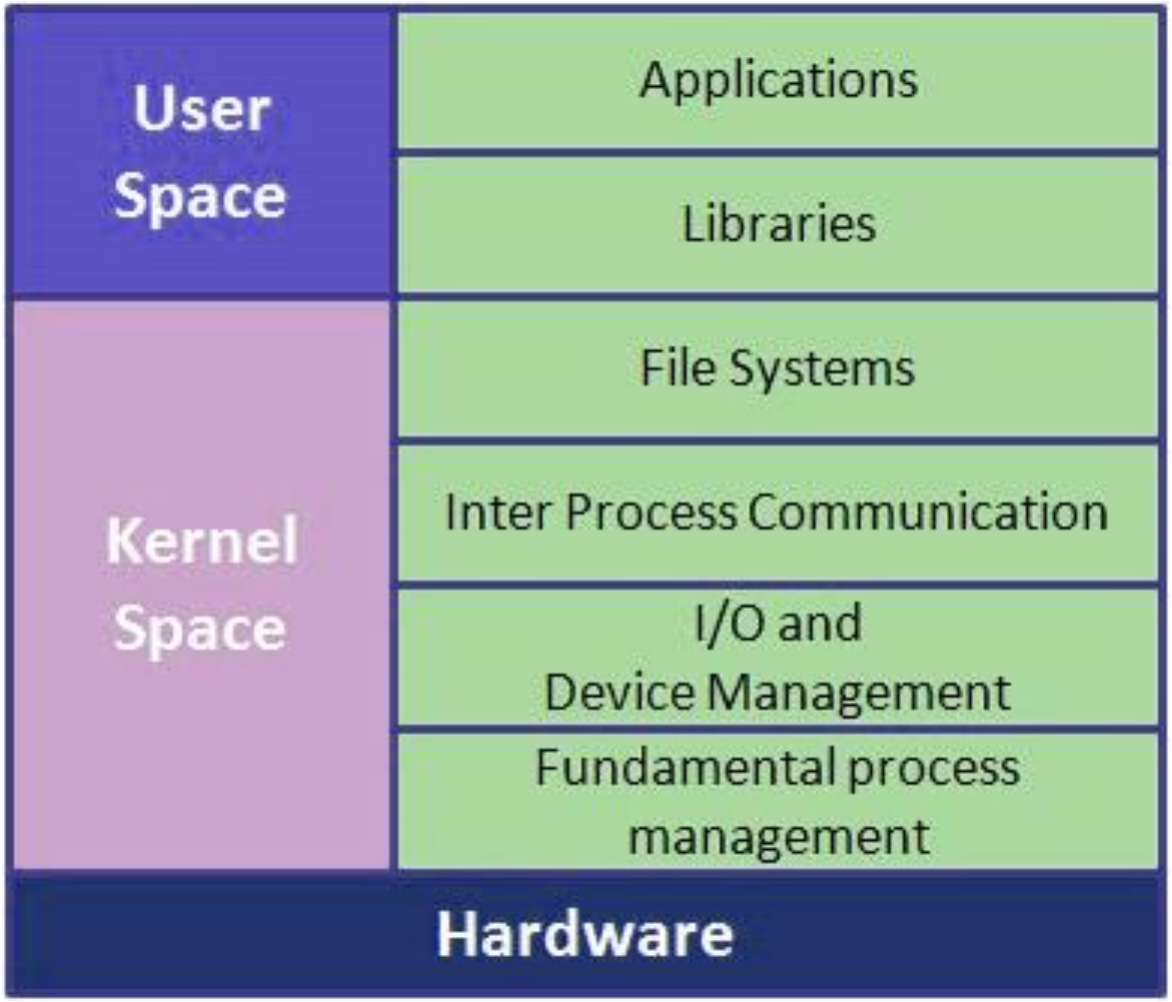
# RTOS Architecture

- There are three broad categories of kernel modes available, namely:

a) Monolithic kernel

  - A monolithic kernel is one single program that contains all of the code necessary to perform every kernel related task.
  - It runs all basic system services (i.e. process and memory management, interrupt handling and I/O communication, file system, etc).
  - As such, monolithic kernels provide rich and powerful abstractions of the underlying hardware.
  - Amount of context switches and messaging involved are greatly reduced which makes it run faster than microkernel.
  - Monolithic kernels are part of Windows and Unix-like operating systems like Linux, FreeBSD.

# RTOS Architecture




Monolithic kernel based operating system

# RTOS Architecture

b) Microkernel

- It runs only basic process communication (messaging) and I/O control.
- It normally provides only the minimal services such as managing memory protection, Inter process communication and the process management. The other system services (file system, networking, etc) reside in user space in the form of daemons/servers.
- The other functions such as running the hardware processes are not handled directly by microkernels. Thus, micro kernels provide a smaller set of simple hardware abstractions.
- It is more stable than monolithic as the kernel is unaffected even if the servers failed (i.e. File System).
- Microkernels are part of the operating systems like AIX, BeOS, Mach, Mac OS X, MINIX, and QNX, etc.

# RTOS Architecture

c) Exokernel:

- The concept is orthogonal to that of micro- vs. monolithic kernels by giving an application efficient control over hardware.
- It runs only services protecting the resources (i.e. tracking the ownership, guarding the usage, revoking access to resources, etc) by providing low-level interface for library operating systems (libOSes) and leaving the management to the application.

# RTOS Architecture

- An RTOS generally avoids implementing the kernel as a large monolithic program.
- The kernel is developed instead as a micro-kernel with added configurable functionalities.
- This implementation gives resulting benefit in increase system configurability, as each embedded application requires a specific set of system services with respect to its characteristics.

# RTOS Features

A basic RTOS will be equipped with the following features:

- **Multithreading and preemptability** – The scheduler should be able to preempt any task in the system and allocate the resource to the thread that needs it most even at peak load.
- **Task Priority** – Preemption defines the capability to identify the task that needs a resource the most and allocates it the control to obtain the resource. In RTOS, such capability is achieved by assigning individual task with the appropriate priority level. Thus, it is important for RTOS to be equipped with this feature.
- **Reliable and Sufficient Inter Task Communication Mechanism** – For multiple tasks to communicate in a timely manner and to ensure data integrity among each other, reliable and sufficient inter-task communication and synchronization mechanisms are required. .

# RTOS Features

- **Priority Inheritance** – Priority inheritance is a mechanism to ensure that lower priority tasks cannot obstruct the execution of higher priority tasks. RTOS should have large number of priority levels & should prevent priority inversion using priority inheritance.
- **Predefined Short Latencies** – An RTOS needs to have accurately defined short timing of its system calls.
  1. Task switching latency: The time needed to save the context of a currently executing task and switching to another task is desirable to be short.
  2. Interrupt latency: The time elapsed between execution of the last instruction of the interrupted task and the first instruction in the interrupt handler.
  3. Interrupt dispatch latency: The time from the last instruction in the interrupt handler to the next task scheduled to run.

# Selection criteria for an RTOS

- **Run-time Performance**: Performance is the most important factor required to be considered while selecting for a RTOS**.** RTOS run time performance is governed by context switching time, interrupt latency and other kernel performance metrics. Run time performance consideration is ideal when an application's performance assessment on an RTOS is prototyping performance critical aspects on standard hardware.

- **Scalability:** Most RTOSs are scalable in which only the required code is added to the final memory footprint. Therefore, finding granular scalability in an RTOS is important because it minimizes memory usage.

- **Portability:** An application may outgrow its hardware as product requirements increase. An RTOS with appropriate capabilities can be ported between specific target systems and processor architectures.

# Selection criteria for an RTOS

- Middleware
  - Almost all real-time operating systems feature middleware components or third-party components that are integrated with the RTOS. You should evaluate the middleware to ensure that it has a seamless integration method. If your RTOS lacks middleware support, you may have to deal with time-consuming integration processes.
  - A real-time operating system should be of premium quality and easy to navigate. Developing embedded projects is hard and time-consuming; developers should not have to struggle with real-time system-related issues that can be distracting. An RTOS should be a trusted component that any developer can count on.
  - Good examples of real-time operating systems are the RTX (32-bit) and RTX64 (64-bit) solutions that allow you and your team to focus on adding value to your applications. This software is designed to serve as a hard real-time system that delivers output within a specified time frame to improve embedded systems' quality.

# Selection criteria for an RTOS

- **Security:** For any device that is going to be connected to the internet, selecting an RTOS that has gone through security certifications and was built for security is a must. Security certifications for an RTOS are relatively new but will be an important criterion for any team that wants to make sure every component in their system is designed with security in mind. One immediate example is verifying that the RTOS has Arm's Platform Security Architecture (PSA) certification at least to level one.
- **Cost:** With new RTOS vendors emerging daily, cost should be a main consideration when selecting an RTOS. Some RTOS packages are offered as complete operating systems that include a real-time kernel, windowing systems, input and output manager, language interface libraries, a file system, networking, cross platform compilers and debuggers. Note that RTOS costs range from $70 to over $20,000. Some RTOS vendors may charge per target system based loyalties that range from $5 to $200 per unit.

# Selection criteria for an RTOS

- **Unique features**: Every real-time operating system has unique features that determine how it operates to execute commands. As such, you must evaluate the features required to run your system effectively and choose an RTOS with the relevant features. A good RTOS should be scalable and feature efficient memory protection systems.

# Misconception of RTOS

- **RTOS must be fast:** The responsiveness of an RTOS depends on its deterministic behavior and not on its processing speed. The ability of RTOS to respond to events within a timeline does not imply it is fast.
- **RTOS introduce considerable amount of overhead on CPU:** An RTOS typically only require between 1% to 4% of a CPU time.
- **All RTOS are the same:** RTOS are generally designed for 3 types of real-time systems (i.e. hard, firm & soft). In addition, they are further classified according to the types of hardware devices (e.g. 8-bit, 16-bit, 32-bit MPU) supported.

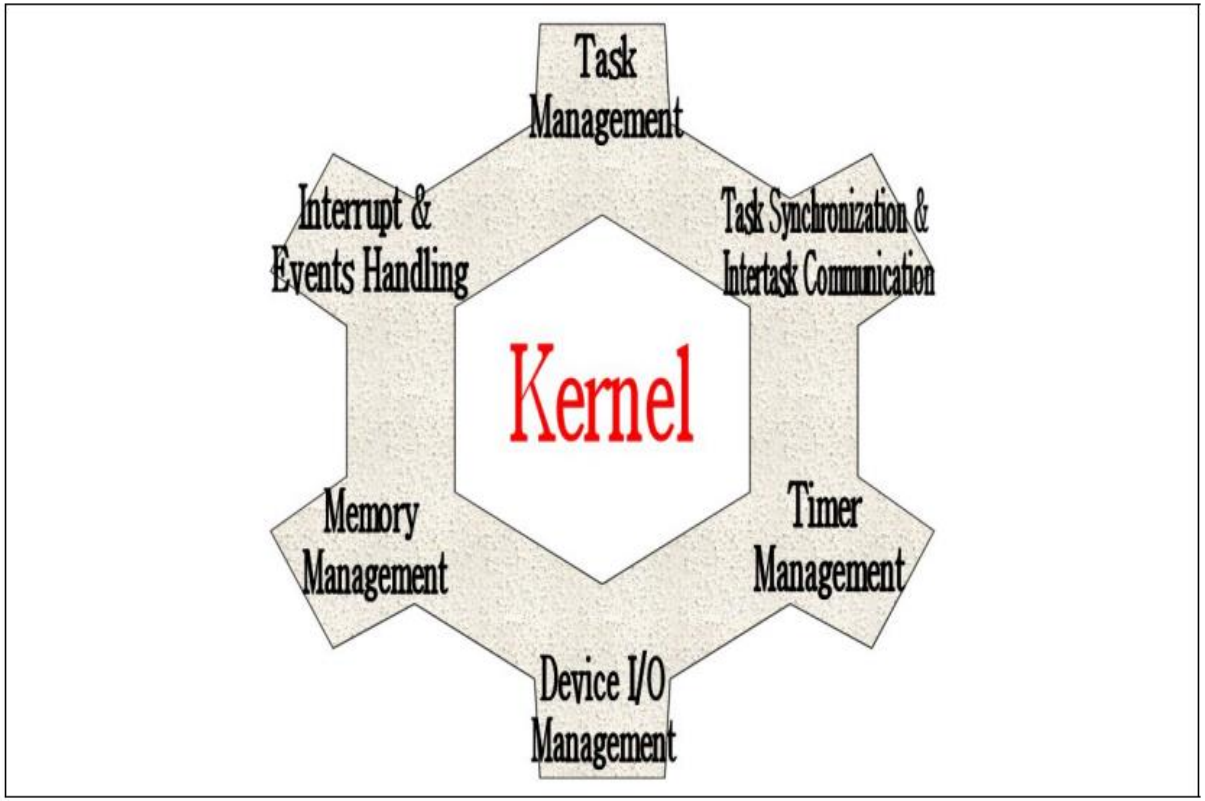
# Examples of RTOS

- LynxOS
- OSE
- QNX (Most traditional RTOS in the market)
- **VxWorks** (Most widely adopted RTOS in the embedded industry, Used in famous NASA rover robots Spirit and Opportunity, Certified by several agencies and international standards for real time systems, reliability and security-critical applications)
- Symbian (Designed for Smartphones, Supported by ARM, x86 architecture)
- Windows CE
- RT Linux

# RTOS
# Kernel Services

# RTOS Kernel Services

- The kernel of an RTOS provides an abstraction layer between the application software and hardware. This abstraction layer comprises of six main types of common services provided by the kernel to the application software.



# Task Management

- A task is an independent thread of execution that can compete with other tasks for concurrent task execution time.
- Task management allows programmers to design their software as a number of separate “chunks” of codes with each handling a distinct goal and deadline.
- This service encompasses mechanism such as scheduler and dispatcher that creates and maintain task objects.

# Thank You