Unit 5

Chpt 5 Real Time Operating Systems (RTOS)

5.1 Operating System Basics

5.2 Types of Operating Systems

5.3 Real Time Characteristics

5.4 Selection Process of an RTOS

Chpt 6 Design and Development

6.1 Embedded system development Environment – IDE

6.2 Types of files generated on cross compilation

6.3 Embedded System Product life-cycle

6.4 Trends in Embedded Industry

**5.**

**Real Time Operating Systems (RTOS)**

Many small scale embedded systems follow superloop based approach. In this approach, a single dedicated task is performed by the system. The instructions in the loop execute in a sequence. If there are any user or event inputs expected in the program, the instructions in the loop will wait for these inputs. If the system is not time critical, the approach works well. For example, electronic toys or video game. Here the events are not happening in real time and they do not have critical response need.

However, if the system is mission critical, and can not wait for events infinitely. For example aviation control system, the program can not wait infinitely. They need real time response.

Another case is of a system where number of small tasks are involved in contrast to a single dedicated task. In such case tasks need to have priorities. Such systems also need to be interactive. In case task priorities are required to change dynamically.

This can be explained with the example of an embedded system. Automotive embedded system like antilock break system (ABS) or nuclear monitoring devices.

Increasing need for real time interaction or responses from the system can be addressed by introduction of operating system to the firmware, or to design firmware based operating system.

**5.1 Operating System Basics**

Figure 5.1 shows operating system architecture. Operating system works as a bridge between the user level and hardware level components. User can dynamically give instructions to the hardware through the kernel level. Application programs actually create the user interface and hardware components such as RAM and ports actually implement the system. Kernel level and its responsibilities are discussed below -

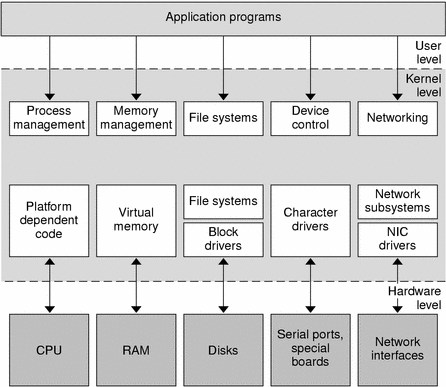
[](https://www.google.co.in/url?sa=i&rct=j&q=&esrc=s&source=images&cd=&cad=rja&uact=8&ved=0ahUKEwjm4L6Ex__XAhVCvY8KHUBnDZgQjRwIBw&url=https://docs.oracle.com/cd/E19683-01/806-5222/kernelovr-1/index.html&psig=AOvVaw1MIhyIEbfexIFWsl2tqJQa&ust=1512998941948512)

Fig 5.1 Operating system architecture

**5.1.1 The Kernel**

The kernel is the core of operating system and is responsible for managing the system resource. Various software services can communicate to the hardware components through the kernel. We can say that the kernel adds a layer of abstraction between user applications and hardware.

Kernel has services to manage the following for an operating system, in most cases, a general purpose operating system.

**5.1.1 (a)**

**Process Management**

A program, or part of program in execution is termed as a process. It can also be referred as instance of program execution. Managing these processes or tasks is nothing but process management. Process management includes following modules or tasks –

* Setting up memory space for the process
* Loading the process code into system’s code memory
* Allocation of system resources
* Scheduling and managing execution of processes.
* Managing Process Control Block
* Inter-process communication and synchronization.
* Process termination / deletion

**5.1.1 (b)**

**Memory Management**

Memory management operations are performed by the kernel for – volatile primary memory (RAM) and for secondary storage memory.

Processes are loaded into the RAM. Also, data associated with each process is stored in the RAM. Memory management unit of the kernel performs following tasks for the RAM –

* Keeping track of memory areas utilized by different processes at different times.
* Dynamic memory allocation

If any secondary storage devices are connected with the system, it is mostly used to back up the system. As main memory is volatile, backup is often necessary. These secondary storage devices are most likely hard disks.

Secondary storage management service of the kernel performs following tasks –

* Disk storage allocation
* Scheduling of disk back ups
* Disk space management.

**5.1.1 (c)**

**File System Management**

Many operating systems have hierarchical file system as its base. Files can be text files, image files or program files. Each file has different type of information stored and it is processed in different manner. Different operating systems have different file types and hence different file handling services. Generally, to handle files, operating system has the file management service. This service does the following jobs –

* Creation, deletion and modification of the files and directories
* Saving of files in secondary storage memory.
* Providing automatic allocation of file space, depending upon the available free space.
* Providing a flexible naming convention for the files.

**5.1.1 (d)**

**Device Management**

Devices used with embedded systems are either input devices or output devices. Whenever these devices communicate with the core of the system, they need to send request signals and will then send or receive data only after they receive acknowledgement.

Some user applications need to communicate with these devices. Hence, the input output request messages are needed to be routed to user applications. This job is performed by the kernel.

Kernel of well structured operating systems perform this task by using an API. The kernel maintains a list of all I/O devices of the system. This list can be available at the time of building the kernel, or it also can be updated dynamically.

All I/O related operation of the kernel are managed by a service called ‘device manager’. The name of service can differ from operating system to operating system.

Device driver service is low level system calls which communicate with all I/O devices.

Device manager is responsible for –

* Loading and unloading of device drivers
* Exchanging information and system specific control signals to and from device.

**5.1.2 Kernel Space and User space**

Application services are classified into –

* User applications
* Kernel applications

User applications are located in a specific area of primary memory. This area is referred to as user memory space or user space.

However, program code corresponding to kernel applications or services is stored in continuous, OS dependent primary memory area. They are protected from the user applications. The area where the kernel applications are stored is called as kernel place.

This separation of space is OS dependent. Some operating systems take up to partitioning of kernel and user space.

In some operating systems, two types of applications, are stored in the same space. In this type of arrangement, entire user application is not stored in the primary memory. Instead, a technique called swapping is used.

In this technique, code corresponding to individual process is loaded into the primary memory. Each process will run in its own virtual memory spce and will not access data from other process space

**5.1.3 Monolithic and microkernel**

Different approaches are used to build an operating system kernel. Two basic approaches are

**Monolithic kernel**

In this architecture, all kernel services are run in the kernel space. All kernel modules run in the same memory. All kernel modules run in the same memory under the same kernel thread. this enables utilization of lower level features. As the system depends upon the single kernel thread, an error may cause the entire system failure. This is the major drawback of the monolithic kernel.

LINUX, SOLARIS, MS DOS are examples of monolithic kernels

**Microkernel**

The microkernel performs the job in abstract manner. Only the essential Operating system services; like process management, memory management, interrupt handlers and timer systems; are provided by the kernel, all other services are implemented in servers. The servers run in userspace. This provides a modular operating system design.

Examples of microkernels are QNX, Minix 3 etc.

Benefits of microkernel based systems are –

Robustness - if any service provided by one of the servers does not function properly, other services still can be used by the system.

Configurability – a change in server system can be done without restarting the whole system.

**5.2 Types of Operating Systems**

Operating systems are classified into various types based on kernel, services provided by the kernel, system and use of the system where operating system is deployed, responsiveness of the operating system.

Two basic types of operating systems are –

* General purpose
* Real time

**5.2.1 General Purpose operating System (GPOS)**

Operating systems deployed in general computing systems are known as general purpose operating systems. The kernel of such operating system is general purpose and contains all types of services. In simple terms, it follows the monolithic approach.

Due to this, GPOS are said to be non deterministic in behaviour. They introduce random delays in application software. This causes unexpected delays in responses. Hence, GPOS are used in systems where deterministic behaviour is not required. Normally, desktop computer is a typical example of GPOS. Windows 7 is a popular GPOS.

**5.2.2 Real Time Operating System (RTOS)**

‘Real time’ operating system are generally the ones having deterministic behaviour. In simple words, the exact time needed for executing services can be known, irrespective of number of services running.

A real time operating system implements policies and rules for allocating of time critical system resources. RTOS will decide which application should run first and for how much time. In this way, by knowing the precise execution time and sequence of execution of services, RTOS can almost predict the execution time of all services.

Following are few operating system features from RTOS point of view –

**5.2.2 (a)**

**Task / Process management**

In operating system context, a task is considered as a job and is defined as the program in execution and related information maintained by operating system. The program, or part of the program in execution is also termed as process. In simple words, the terms job, task and process are more or less same in operating system context.

Multiple instances of the same program, or multiple processes can be executed simultaneously. Execution of program or program segment, need multiple resources from the system. For example, CPU time, memory, input and output devices and so on. Thus concurrent executions need meticulous planning of utilizing these resources sequentially. The basic concept of process itself leads to concurrent executions of tasks and utilization of various system resources.

Process management deals with creation of the process, setting up memory space for the process, loading the process code into the memory space, allocating system resources and setting up process control block for termination / deletion of the process.

**5.2.2 (b)**

**Task / Process Scheduling**

In order to do multitasking, there has to be a method to share the CPU between multiple tasks. Also, a decision has to be made as which task will execute at a given point in time. Determination of a task to be executed at given point in time is known as task scheduling or process scheduling. Scheduling policies are implemented in an algorithm, by the kernel and is run as a service. The kernel which actually implements the algorithm is called the scheduler. Various states of process and their state transition is shown below –

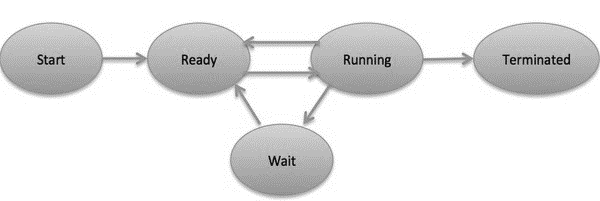


Fig 5.2 Process States

The process scheduling decision takes place when the process state changes from one to another. For example, from ready to running state.

Scheduling for the process state can be preemptive, (that is all the process resources are pre-empted.) or non preemptive. It can also be cooperative.

Following are the factors responsible to chose a scheduling criterion.

* CPU Utilization
* Throughput
* Turnaround time
* Waiting time
* Response time

**5.2.2 (c)**

**Error / exception handling**

Error or exception handling mechanism of RTOS deals with errors occurred or exceptions raised during the execution of the tasks. There can be several exceptions or errors that may arise during the execution. A few examples could be – deadlock, timeouts, bus error, division by zero error etc.

Some errors or exception are at kernel level, while others are at task level. For instance, deadlock occurs at the kernel level, where as timeout occurs at the task level. The information about the error is given by the os kernel in form of API (Application Program Interface). These interfaces are system calls which can provide additional information about execution of the task.

Certain tasks may involve waiting for external events from external devices. If the device is not responding the task may end up waiting infinitely pulling the entire system in hung state. Hence, a properly defined timeout and a mechanism which monitors this timeout is needed. This job is performed by watchdog timer. A watchdog is loaded with maximum possible wait time for the event. If the event does not take place for this time, then timeout occurs and the information is given to the task and operation is aborted. If the event occurs before timeout, then the watchdog is reset.

**5.2.2 (d)**

**Memory Management**

Memory management in GPOS and RTOS is done differently in some aspects. Memory allocation time changes as per the memory block size and state. Initialized memory block will take more allocation time as compared to non initialized memory block. The prediction of execution time is an important factor in RTOS. For this purpose, RTOS does blockwise memory allocation unlike the dynamic allocation of memory done by the GPOS.

Memory blocks are stored in queues and RTOS kernels allow memory access of these memory blocks to the tasks as per the need arises. No memory protection is considered as RTOS kernels assume that the entire memory design is correct. Certain RTOS offer protection and they have a defined ‘fail-safe’ mode for illegal memory access.

Certain RTOS utilize virtual memory. In block based memory approach, a block of fixed memory is always allocated to tasks as per their need. The memory block acts as a unit and question of fragmentation does not arise. Also, garbage collection overhead is avoided as the entire block is one memory unit.

Thus the block based memory implements non deterministic behaviour at the cost of choice of memory block size and options in memory usage.

**5.2.2 (e)**

**Interrupt Handling**

Interrupts make changes to the task execution flow of embedded system. Interrupts can either be synchronous, which occur in sync with the currently executing task; or they can be asynchronous, which occur due to external peripheral devices which are not in synchronization of the microcontroller.

Synchronous interrupts occur during the execution of a task. They are majorly generated by software programs. For example, division by zero. The interrupt subroutine, in such cases interrupt handler is normally part of the same interrupt initiating task.

Asynchronous interrupts occur due to speed mismatch, or communication error between the input output devices and the microcontroller. Serial data reception / transmitter is an example of asynchronous interrupt. The interrupt handler in such cases is written in a separate context. Each of the asynchronous interrupt can be set with different priority. They can also be individually enabled or disabled.

**5.2.2 (f)**

**Time Management**

Precise response time is one of the most important characteristic of embedded system. The time reference to the RTOS kernel is provided by Real Time Clock (RTC). RTC is nothing but a hardware chip. This hardware timer interrupts the microcontroller after every fixed interval of time. As the timer interrupt ticks after fixed time, it is known as the *timer tick.* This is considered as timing reference for the RTOS kernel. The interval of the timer tick may vary according to the hardware. The time parameters required for any tasks will always be multiples of this timer tick interval.

System time is also updated based on this timer tick.

Working of timer tick is explained in following example –

If system time register of a microcontroller is of 32 bit size, and timer tick interval is 1 ms, then the system time register will be reset in –

232 \* 10-3 / 24 \* 60 \* 60 = 49.71 days ≈ 50 days.

Timer tick interval can be implemented for performing following actions –

* Save current context
* Increment the system time register
* Update timers implemented in kernels
* Activate periodic tasks
* Invoke scheduler
* Delete all terminated task

**5.2.2 (g)**

**Hard and Soft Real Time**

Real time operating system that needs to follow the timeline very strictly, is hard real time operating system. For example, antilock break system in a car, needs to respond at real time without any delay in order to perform the task assigned to it. A late response in such a system can be fatal, and these systems are also termed as mission critical systems.

The job of maintaining perfect timeline in multiple possible tasks, is performed with the help of a scheduler.

The response time of the system is predefined. When any task is triggered by an external event, like interrupt, the currently executing task must be pre-empted.

In order to meet the time frame listed in the problem definition, time spent between the deployment of one task and implementation of the previous task should be minimal, or zero.

Hard real time system do not use virtual memory. This eliminates any delay which can arise due to swapping of data in virtual system.

Most hard time systems are automated and thus they avoid excess delay introduced by means of human inputs or control to the program. They do not follow human in the loop approach.

On the other hand, real time operating system do not meet the time deadlines of kernel. They are known as Soft real time system. This missed deadline, or some permissible delay for the soft real time system must be within a specified range. This range is known as – Quality of Service.

These soft real time systems do use the human in the loop approach (HITL).

ATM can be a typical example of soft real time system.

**5.3 Real Time Characteristics**

In real time systems, specially the hard real time systems, timely response of the system is absolutely essential. Late response can be as good as wrong response of the system. As design of real time embedded system is mission critical, designers need to know about its performance and behaviour.

**Real time and reactive operation :**

The time taken by a real time operating system to respond to event is crucial. Worst case performance is the maximum time taken by the processor / controller to start the interrupt service routine. This is known as maximun ISR latency. This should be minimum for a good embedded system.

**Deterministic nature :**

If the worst case response time is known, the system can be better designed. The deterministic nature of a real time operating system leads to a design with perfectly calculated worst case response time and interrupt latency. Interrupt latency is the total length of time from an interrupt signal's arrival at the processor to the start of the associated interrupt service routine.

To execute an ISR processor needs to perform following steps –

* Complete execution of current instruction.
* Recognize the interrupt type using the hardware.
* Start the ISR if interrupt is enabled.

If the interrupt is disabled, however, it will be the maximum interrupt latency.

**Time required to perform context switch**

It is the time taken by the microcontroller to switch between two execution sequences. The primary program and the secondary ISR. This time may also be as much as the entire execution time of ISR.

This time is processor specific and will vary from one processor to another, thus from one system to another. Minimum context switch time will make better real time operating system.

**I/ O device flexibility**

Compatibility of a real time operating system with more I/O devices and device drivers make the system versatile. However, it may or may not be the criterion for many embedded designs, as they perform specific tasks and with specific devices.

**5.4 Selection Process of an RTOS**

Several real time embedded systems are available. Choosing the correct RTOS is a crucial task. The RTOS selected will play major role in performance of embedded system.

Commercial operating systems offer a set of functionalities, performance, and price. Those at the lower end of the spectrum offer only a basic scheduler and a few other system calls. These operating systems are usually inexpensive, come with source code that you can modify, and do not require payment of royalties. Examples of such lower version RTOS are - Accelerated Technology's Nucleus and Kadak's AMX, also the embedded versions of DOS.

Operating systems at the other end of the spectrum typically include a lot of useful functionality beyond just the scheduler. They might also make stronger (or better) guarantees about real-time performance. These operating systems can be quite expensive. Examples of such RTOS are - Wind River Systems' VxWorks, Integrated Systems' pSOS, and Microtec's VRTX.

Several factors need to be analysed before selection of RTOS. There will be functional and non functional needs to be considered for selecting correct RTOS.

**Functional Requirements**

* Processor support

*It is essential that the RTOS to be selected supports the processor or controller. Processor architecture should be accessible by the RTOS selected.*

* Memory requirement

*OS requires RAM to load the services provided and ROM to store OS files. ROM is non volatile like FLASH memory. Adequate size of RAM and ROM required for the RTOS should be available in the hardware of the system. RTOS will not function efficiently if these requirements are not satisfied.*

* Real time capabilities

*Operating systems used for all embedded systems should have real time capabilities. This is important as embedded systems respond to real time events.*

* Kernel and Interrupt latency

*Interrupt latency, or the time required for the interrupt service routine to start execution, plays a key role in calculating crucial response time of an embedded system. Kernel can enable, disable and set priorities of interrupt and are essential component of RTOS selection.*

* Inter-process communication and task synchronization

*Inter process communication and synchronization implementation is dependent of OS kernel. Different kernels, in other words, different RTOS may have different policies and options. Certain RTOS do not allow dynamic priority inversion in resource sharing.*

* Modularisation support

*Modular design in strength of any working system. In such design developer can choose essential modules and can reprogram them for improved functionality.*

* Support for networking and communication

*It depends upon the RTOS selected, that which communication devices can be used for interfacing and networking. The interfaces required by the embedded system should be accessible by the RTOS to be selected.*

* Development languages support

*Certain operating systems include runtime libraries for running applications written in certain languages. OS may include these components as built in components. If not then they will have to be made available through a third party program. This has to be confirmed before selection of operating system.*

**Non - Functional Requirements**

* Custom developed or off the shelf

*Operating system could be developed using open source software or from scratch for specific embedded system. That means, operating systems developed for a specific embedded system can only be used for that system. Another option is to use any ready made available operating system. This ready made product is said to be off the shelf product. In this case, off the shelf operating system.*

* Cost

*Before choosing an operating system, total cost of buying and maintaining the operating system has to be taken into account.*

* Development and debugging tool availability

*Embedded program development tools are the ones responsible for programming the embedded system. Operating system used must support these development tools. Hence, while selecting the operating system, it is essential to check support of operating system to development and debugging tools.*

* Ease of use

*Ease to use and user friendly interface and commands is another important factor to be considered in choice of RTOS.*

* After sales

*For commercial RTOS, support after sale and implementation is critical. The entire embedded system is designed, developed as per the custom design and deployed. After investing cost and resource into this system, users expect the system to last long. Hence the factor after sale support is important factor in decision making of RTOS.*

**Summary**

Operating system used in mission critical, real time embedded systems are real time operating system. (RTOS)

Operating system works like a bridge between user level components and hardware level components.

Kernel is the core of the operating system and is responsible for managing the system resource.

Main RTOS features are –

* Process Management
* Memory Management
* File System Management
* Device Management

Application services can be classified as –

* User space
* Kernel space

Different approaches are used to build OS kernel. Two of the approaches are –

* Monolithic
* Microkernel

Operating System are basically of following two types –

* General purpose
* Real time

General purpose operating system contains all types of services.

GPOS is non deterministic in nature

RTOS is deterministic in nature.

Exact time needed for execution of the service can be predicted.

Operating System Features of RTOS are –

* Task / Process Management
* Task / Process Scheduling
* Error / exception Handling
* Memory management
* Interrupt handling
* Time management
* Hard / soft real time system.

Real time characteristics of RTOS are –

* Real time and reactive operation
* Deterministic Nature
* Time for context switching
* I/O device flexibility

Selection of correct real time operating system suited for embedded system is a crucial task and it depends upon following functional and non functional requirements of the system –

Functional requirements –

* Process Support
* Memory Requirement
* Real time Capabilities
* Kernel and Interrupt Latency
* Interprocess communication and task synchronization
* Modularization Support
* Support for networking and communication
* Development Languages Support

Non functional requirements –

* Custom developed or off the shelf
* Cost
* Development and debugging tool availability
* Ease of use
* After sales

**Questions**

* Explain the need of real time operating systems.
* With neat diagram explain the basic architecture of an operating system.
* What are different types of operating systems? give two examples of each.
* List the different services provided by kernel of operating system.
* Differentiate between general purpose and real time operating system
* What are different features of RTOS?
* What is process scheduling? What are factors responsible in choosing scheduling criterion?
* Explain the term interrupt latency.
* List and explain real time characteristics of RTOS
* What is the importance of choosing the correct RTOS?
* List the factors contributing to selection of RTOS.