Case study on Operating systems

By: Sparsh Arya.

Registration number: 17BEC0656.

Slot: F2.

Subject: Operating system.

Topic chosen: Real time Operating system.

Introduction

A real-time operating system or RTOS is an operating system which intends to serve real-time applications that process data as it comes in. This is typically done to avoid buffer delays. Processing time is measured in tenths of seconds.

A real time system is a time bound system which has well defined fixed time constraints. Processing should be done within the defined constraints or the system will fail. They can be event driven or time sharing. Event driven systems switch between tasks on the basis of their priorities. Time sharing systems switch the task based on clock interrupts

Architecture of RTOS

RTOS is a kernel. Complex versions may include various modules like:

- Networking protocol.
- Stacks.
- Debugging facilities.
- Device I/Os

Kernel

RTOS kernel plays the role of an abstraction layer between the hardware and the applications. There are three broad categories of kernels:

- **Monolithic kernel**

Monolithic kernels are part of Unix-like operating systems like Linux. A monolithic kernel is one single program that contains the code necessary to perform all kernel related task. It runs all basic system services

- Process and memory management.
- Interrupt handling.
- I/O communication.
- File systems.

- **Microkernel**

It runs only basic process communication and I/O control. It provides only the minimal services such as:

- Managing memory protection
- Inter process communication
- The process management.

The other functions such as running the hardware processes are not handled directly by microkernels. It’s more stable than monolithic as the kernel is unaffected even if the servers failed. Microkernels are part of the operating systems like AIX, BeOS and QNX.

.

Hybrid Kernel

Hybrid kernels are extensions of microkernels with properties of monolithic kernels too. Hybrid kernels are similar to microkernels, except for the fact that they include additional code in kernel space so that such code can run more swiftly than it would were it in user space. These kernels are part of the operating systems such as Microsoft Windows NT, 2000 and XP.

.

Architecture - Task Management

Task Object

In RTOS, The application is broken into small, schedulable, and sequential program units known as “Task”. A basic unit of execution and is governed by three time-critical properties:

- Release time
- Deadline
- Execution time.

Each task may exist in states like Ready, Active, Dormant, Suspended and Pending.

During the execution of an application program, individual tasks are changing from one state to another. However, only one task is in the running model at any point of the execution.

In the process where CPU control is change from one task to another, context of the to-be-suspended task will be saved while context of the to-be-executed task will be retrieved. The following process is termed as context switching.

Scheduler

The scheduler keeps record of the state of each task and selects from among them that are ready to execute and allocates the CPU to one of them. Various scheduling algorithms are used in RTOS like Round robin, Polled System with interrupts, Hybrid System, Non pre-emptive scheduling or Cooperative Multitasking and Pre-emptive scheduling Priority multitasking.

Dispatcher

The dispatcher gives control of the CPU to the task selected by the scheduler by performing context switching and changes the flow of execution.

Synchronisation and communication

Task Synchronisation serves to pass information amongst tasks.

Task Synchronisation

Synchronization is essential for tasks to share mutually exclusive resources and allow multiple concurrent tasks to be executed

Task synchronization is achieved with the help of two types of mechanisms:

- Event Objects
- Semaphores.

There are three types of semaphore:

- Binary Semaphores
- Counting Semaphores
- Mutually Exclusion Semaphores

Memory Management

Two types of memory managements are provided by RTOS – Stack and Heap.
Stack management is used during context switching for TCBs. Heap memory is used for dynamic allocation of data space for tasks.

Timer Management

Tasks need to be performed after scheduled durations. To keep track of the delays, timers- relative and absolute- are provided in RTOS.

Device I/O Management

RTOS generally provides large number of APIs to support diverse hardware device drivers.

Services offered by RTOS

Task Control

Beyond task scheduling and communication, an RTOS will undoubtedly include functionality – API calls – to control tasks in a selection of ways. We can look at some of the possibilities.

Task Creation and Deletion – In a “dynamic” RTOS, there will be calls to enable the creation of tasks as and when they are required. Such calls include a wide range of parameters which define the task.

Task Suspend and Resume

Task Sleep

Relinquish

Task

Task Reset

Task Priority

System Information

An RTOS will provide a range of API calls to provide tasks with information about the system, including:

- Task information
- RTOS version information.

Memory Allocation

In many applications, it is useful for a program to be able to grab some memory when it is required, and release it again when it is no longer needed. Embedded software is not an exception to that.

Memory Partitions

A solution to providing dynamically available memory for a real-time system, in a reliable and predictable fashion, is to use a memory block-based approach. Such blocks are commonly called "partitions"; partitions may be allocated from a "partition pool".

Time

It is self-evident that functionality concerned with the use and control of time are likely to be available in a real-time Operating system. The offer varies from one RTOS to another. In all cases, a real-time clock is a given for any of these services to function.

Drivers

Most RTOS define some kind of device driver structure. The details vary from one RTOS to another, but a driver generally consists of two interacting components: some inline code and an ISR.

<u>Core functionalities of RTOS</u>

Reliability:

It operates for a reasonably long time without human interference. Reliability also means the configuration to enable the system to choose the right or most profitable action for current operation

Predictability:

A system must execute actions within a known time frame and produce results. Such results are determined by the procedures or operations taking place. The determination is by targets set during production or by procedural planning.

Performance:

Real-time operating systems are designed to the make work easier. Every system must solve the problem or reduce the workload. The developer should provide a system that is easily assimilated with existing software and hardware as well as aligned to the goals of the organization.

Manageability:

A system whose veracity or bulkiness is manageable is convenient to use. The software and hardware required to operate the RTOS must be of a reasonable size. Technicians should also be easy to find and orient. The idea is to reduce the cost of implementation.

Scalability:

A system may require an upgrade or downgrade. Such provisions must be made during design and installation of any RTOS.

<u>Applications of RTOS</u>

Applications with strict deadlines

Some tasks must be completed before the deadline. However, the exact instant when the task is performed is not crucial.

For example, a sound card buffer has to be refilled before it is emptied. The output port's voltage must reach a certain level before the value is read by the next peripheral device.

Applications with zero execution time

This is where a task has to be performed at a specific instant that is minimized.

For example, in a digital control theory, taking measurement, calculating a control action, and sending it to the peripheral device occurs instantly.

Quality of service

An RTOS can be used when a task must have a fixed service amount per time unit. However, it can be network bandwidth, disk access bandwidth or memory pages. This is very crucial for applications like network services and video and audio streaming.

Applications that involve competition for resources

In some applications, tasks compete for several resources like network, processors, and memory. This competition is stiffer in real-time operating systems than in general purpose operating systems. Therefore, a real-time operating system ensures that each task is allotted the necessary resources promptly.

Applications with worst-case scenarios

If there are numerous tasks that need a service, there will be a time when they will all need the service at once. Real-time operating systems are used in such applications to ensure that worst-case scenarios are given a sense of priority.

<u>Comparison of RTOS</u>

VX works

VxWorks is a real-time operating system developed as proprietary software by Wind River Systems, an Intel subsidiary of Alameda, California, US.

It was first released in 1987, VxWorks is designed for use in embedded systems requiring real-time, deterministic performance and, in many cases, safety and security certification, for industries, such as aerospace and defense, medical devices, industrial equipment, robotics, energy, transportation, network infrastructure, automotive, and consumer electronics.

- Scheduler: Pre-emptive
- Synchronization mechanism: No condition variable
- Custom hw support: BSP
- Multiprocessor support: VxMP/VxFusion

PSOS

pSOS (Portable Software On Silicon) is a real time operating system (RTOS), created in about 1982 by Alfred Chao, and developed/marketed for the first part of its life by his company Software Components Group (SCG). In the 1980s pSOS rapidly became the RTOS of choice for all embedded systems based on the Motorola 68000 family architecture, because it was written in 68000 assembler and was highly optimised from the start. It was also modularised, with early support for OS-aware debugging, plug-in device drivers, TCP/IP stacks, language libraries and disk subsystems. Later came source-level debugging, multi-processor support and further networking extensions.

- Scheduler: Pre-emptive
- Custom hw support: BSP
- Multiprocessor support: PSOS + m kernel
- Kernel size: 16KB

eCOS

The Embedded Configurable Operating System (eCos) is a free and open source real-time operating system intended for embedded systems and applications which need only one process with multiple threads. It is designed to be customizable to precise application requirements of run-time performance and hardware needs. It is implemented in C/C++ and has compatibility layers and application programming interfaces for POSIX and µITRON.

- Scheduler: Pre-emptive
- Posix support: Linux
- Custom hw support: HAL

Advantages of Real Time Operating System

There are some of the features of using RTOS that is described below

Maximum Consumption: – RTOS give maximum consumption of the system and gives us more output while using all the resources and keeping all devices active.

Task Shifting: – There is very little time assigned to shifting tasks in these systems.

Focus on Application: – These type of operating system focus on applications which are running. It usually gives less importance to other application residing in waiting stage of life cycle.

Real time operating system in embedded system: – Due to small size of programs RTOS can also be used in embedded systems like in transport and others.

Error Free: – RTOS is error free meaning it has no chances of error in performing tasks.

24-7 systems: – RTOS can be best used for any applications which run for 24 hours and 7 days because it do less task shifting and give maximum output.

Memory Allocation: – Memory allocation is best managed in these type of systems.

Disadvantages of Real Time Operating System

Limited Tasks: – There are only a number of limited tasks that run at the same time and the concentration of these system are on few application to avoid errors and other task have to wait. Sometime there is no time limit of how much the waiting tasks have to wait.

Use heavy system resources: – RTOS used lot of system resources which is not as good and is also expensive.

Low multi-tasking: – Multi tasking is done few of times and this is the main disadvantage of RTOS because these systems run few tasks and stay focused on them. So it is not best for systems which use lot of multi-threading because of poor thread priority.

Complex Algorithms: – RTOS uses complex algorithms to achieve a desired output and it is very difficult to write that algorithms for a designer.

Device driver and interrupt signals: – RTOS must need specific device drivers and interrupt signals to response fast to interrupts.

Thread Priority: – Thread priority is not as good as RTOS do less switching of tasks.
