FreeRTOS

The real-time operating system (RTOS) is an operating system intended to serve real time application that process data as it comes in, mostly without buffer delay. FreeRTOS is an open source software used to switch between tasks, used to manage several tasks/ jobs running concurrently where meeting strict timing deadlines in important. It runs on the MCU and constitutes the CubeSat's flight software.

Terms used in RTOS

Here, are essential terms used in RTOS:

- **Task –** A set of related tasks that are jointly able to provide some system functionality.
- **Job –** A job is a small piece of work that can be assigned to a processor, and that may or may not require resources.
- **Release time of a job –** It's a time of a job at which job becomes ready for execution.
- **Execution time of a job:** It is time taken by job to finish its execution.
- **Deadline of a job:** It's time by which a job should finish its execution.
- **Processors:** They are also known as active resources. They are important for the execution of a job.
- **Maximum:** It is the allowable response time of a job is called its relative deadline.
- **Response time of a job:** It is a length of time from the release time of a job when the instant finishes.
- **Absolute deadline:** This is the relative deadline, which also includes its release time.

Components of the RTOS

Discussed below are the components of the RTOS:

The Scheduler: This component of RTOS tells that in which order, the tasks can be executed which is generally based on the priority.

Symmetric Multiprocessing (SMP): It is a number of multiple different tasks that can be handled by the RTOS so that parallel processing can be done.

Function Library: It is an important element of RTOS that acts as an interface that helps you to connect kernel and application code. This application allows you to send the requests to the Kernel using a function library so that the application can give the desired results.

Memory Management: this element is needed in the system to allocate memory to every program, which is the most important element of the RTOS.

Fast dispatch latency: It is an interval between the termination of the task that can be identified by the OS and the actual time taken by the thread, which is in the ready queue that has started processing.

User-defined data objects and classes: RTOS system makes use of programming languages like C or C++, which should be organized according to their operation.

Types of RTOS

Three types of RTOS systems are:

Hard Real Time:

In Hard RTOS, the deadline is handled very strictly which means that given task must start executing on specified scheduled time, and must be completed within the assigned time duration.

Example: Medical critical care system, Aircraft systems, etc.

Firm Real Time:

These type of RTOS also need to follow the deadlines. However, missing a deadline may not have big impact but could cause undesired affects, like a huge reduction in quality of a product.

Example: Various types of Multimedia applications.

Soft Real Time:

Soft Real time RTOS, accepts some delays by the Operating system. In this type of RTOS, there is a deadline assigned for a specific job, but a delay for a small amount of time is acceptable. So, deadlines are handled softly by this type of RTOS.

Example: Online Transaction system and Livestock price quotation System.

Applications within the OBC are organized in tasks to which a priority level and a deadline are assigned. These tasks are coordinated by a real time operating system (RTOS) which according to Whilmshurst (2007) accomplishes three major functions when implemented in a system:

- It decides which task should run and for how long
- It provides communication and synchronization between tasks
- It controls the use of resources shared between tasks, for example memory and hardware peripherals.

In the RTOS, direct to task notifications, queues, binary semaphores, counting semaphores, recursive semaphores and mutexes for communication and synchronisation between tasks, or between real time tasks and interrupts can be implemented. The thread tick method is used to switch tasks depending on priority and a round-robin scheduling scheme. Task Synchronization technique, Mutual Exclusion - Sleep

& Wakeup Events, is implemented in the RTOS which uses notification mechanism for synchronization. Power saving is achieved by the method of tickless mode which is limited by the necessity to periodically exit and then re-enter the low power state to process tick interrupts. Three types of task are defined for the purposed architecture which are Periodic Task, Periodic Update Task and Aperiodic Task.

Periodic Task takes action on a regular basis; Periodic Update Task collects data and places it in a global memory area for use by other task on regular basis. Aperiodic Task runs as a result of some external command response from ground station. The real time operating system (RTOS) is designed basically a task-scheduling program that responds to events such as interrupts in real time, i.e. a few milliseconds at most. Flight software architecture is developed using the types of task proposed above.

The Implementation of the RTOS

When the use of a specific RTOS is required, the selected microcontroller needs to be supported by it. An adequate integrated development environment (IDE) to write and compile applications or tasks is also needed in order to program the microcontroller or load the RTOS.

We can choose an IDE such as Microvision Keil for implementing the RTOS for our project. Firstly, we will need to list out the tasks required for the satellite operation keeping in mind that there are three types of task basis;

• Periodic task: This task must take action on a regular basis.

• Regularly update task: This task will collect data and place for regular use by the global storage area for other tasks.

• Aperiodic tasks: This task will be as a result of some external command response, such as running from the results of the ground.

The tasks are assigned a priority based on criticality of the task and execution time. Each task will have a message queue assigned to it and will block on the message queue, with a timeout equal to its period. If the task is aperiodic, it will have an infinite timeout.

Using this mechanism, there are two ways to activate a sleeping task: send a message to its message queue, which will be processed immediately (as long as another higher priority task is not already active) or wait for the timeout period to expire (assuming the task is periodic). The task manager is responsible for context switching between active tasks, based on their priority.

Utilization of an algorithm could significantly reduce the complexity of the code and save a lot of time. Algorithms that can be used for the development purpose are Prioritized Preemptive Scheduling and Co-operative Scheduling.

Prioritized Preemptive Scheduling

This Prioritized Preemptive Scheduling techniques has following features

• Each task is assigned a priority.
• Each task can exist in one of several states.
• Only one task can exist in the Running state at any one time.

• The scheduler will always select the highest priority Ready state task to enter the Running state. This type of scheme is called ‘Fixed Priority Preemptive Scheduling’. ‘Fixed Priority’ because each task is assigned a priority that is not altered by the kernel itself (only tasks can change priorities). ‘Preemptive’ because a task entering the Ready state or having its priority altered will always pre-empt the Running state task if the Running state task has a lower priority.

Co-operative Scheduling

A hybrid scheme is utilized where interrupt service routines are used to explicitly cause a context switch. This allows synchronization events to cause pre-emption, but not temporal events. The result is a preemptive system without time slicing

Implementation of tasks and priorities

Each task is assigned a priority from 0 to (**configMAX_PRIORITIES - 1**). **configMAX_PRIORITIES** is defined within **FreeRTOSConfig.h** and can be set on an application by application basis. The higher the value given to **configMAX_PRIORITIES** the more RAM the FreeRTOS kernel will consume. Low priority numbers denote low priority tasks, with the default idle priority defined by **tskIDLE_PRIORITY** as being zero. The scheduler will ensure that a task in the ready or running state will always be given processor time in preference to tasks of a lower priority that are also in the ready state. In other words, the task given processing time will always be the highest priority task that is able to run.

In conclusion, the overall system architecture and requirements of the subsystems is accomplished by the open-source real-time operating system (FreeRTOS) which supports various types of peripheral interfaces, telemetry storage and data processing. The only drawback of FreeRTOS is that tasks are designed to enter in infinite loops. Satellite function tasks should not be in infinite loop. The tasks should function in the specified conditions without going into infinite loops. To overcome this issue implementation of special techniques to avoid infinite loops is required.

The implementation of several applications (or multi-tasking applications) based on a RTOS constitutes the CubeSat’s flight software (FSW). Hidayat (2010) defines the FSW as “the central intelligent system of the modern satellite that can run on a single chip”. Enright (2002) further summarises the major flight software functions as follows:

• It schedules satellite activities autonomously when the satellite is away from the ground station radiation coverage, such as scheduling the payload imager to take a picture of a certain location at a particular time;

- It responds to ground user telecommands during the ground station overpass. The FSW should be able to interact with the ground user by accepting all the valid commands and execute commands based on their parameters;
- It performs data logging by taking measurements of the housekeeping parameters (telemetry) and stores them into the memory device until they can be sent to the ground station. Data logging also includes the logging of events that took place during flight;
- It performs data management, including data structure and protocol.