CASE STUDY: EVOLUTION OF RTOS

SYNOPSIS/EXECUTIVE SUMMARY

A real-time operating system (RTOS) is an operating system (OS) intended to serve real time application requests. It must be able to process data as it comes in, typically without buffering delays. Processing time requirements are measured in tenths of seconds or shorter. Real-time operating systems have evolved over the years from being simple executives using cyclic scheduling to the current feature-rich operating environments. The standardization of POSIX 1003.1, ISO/IEC 9945-1 (real-time extensions to POSIX) has contributed significantly to this evolution, however, the specification leaves plenty of room for individual implementations to both interpret and specialize their RTOSs. Accordingly, there has been a proliferation of both commercial and free RTOSs, notably, the ITRON OS, the OSEK-VDX OS specification, commercial RTOSs like VxWorks, VRTX, LynxOS, OSE and QNX, and free RTOSs like RT-Linux (RTAI), and Windows CE. The goal of the work reported in this paper is to draw the real-time systems practitioner and researcher’s attention to these choices and bring out the similarities and differences among them.

DISCUSSION

The concept of real time systems was introduced almost seventy years back. James Martin who was a famous British information technology consultant and author, proposed one of the first definitions for real time systems – A real time computer system may be defined as one which controls an environment by receiving data, processing them and taking action or returning results sufficiently quickly to affect the functioning of the environment at that time. In the initial years, there was no operating system developed that was inherently real time. Whenever there was a requirement for a real time task, the whole system, both the hardware and software was designed in such a way that it is customized to that particular real time task alone. The roots of real time systems are deeply embedded in these two areas. Operations research is mainly involved in decision making process. By the use of advanced analytical models, it helps in strategic decision making. In the context of real time systems, it helps us to decide which task should be run next by an operating system. Queuing theory, as the name implies, deals with the learning of queues. The amount of time an element should wait in the queue, the average length of the queue etc is studied in this area. Queues play a major role in real time systems as a number of real time tasks will be ready to run in a particular instance, all of which will be waiting inside a queue. Operating system should use data structures in such a way that it minimizes the waiting time of the tasks, and no high priority task should be kept waiting while a lower one is being executed.

RTOS vs. GPOS

RTOS and GPOS are distinct from each other because of the following characteristics.
1. Time criticality vs. throughput: Main goal of RTOS is to achieve deterministic behavior. Results have to be produced within strict deadlines. Usually RTOS is used only for customized applications. Hence focus is more on the timeliness rather than the amount of work done, whereas in GPOS main aim is to achieve high throughput. GPOS is dedicated to a large number of tasks. Hence maximum amount of work has to be completed within a given amount of time.
2. Scheduling Algorithms: GPOS has the liberty to use any scheduling algorithm as long as the throughput is met. Whereas in RTOS, algorithm is always priority based. A higher priority task is never made to wait. Hybrid algorithms can be used but priority based algorithm must be one among them.

3. Latency: In GPOS there is unbounded dispatch latency. The more number of threads to schedule, more latency will get added. In RTOS, processes and threads in it has got bounded latencies due to the application of queuing model.

4. Hardware: RTOS is light weight and small in size compared to a GPOS. A GPOS is made for high end, general purpose systems. An RTOS is usually designed for a low end, stand alone device. It is economical to port an RTOS to an embedded system of limited expectations and functionalities.

**Important Terminology and Concepts**

Determinism: An application (or critical piece of an application) that runs on a hard real-time operating system is referred to as deterministic if its timing can be guaranteed within a certain margin of error. Soft vs Hard Real-Time: An OS that can absolutely guarantee a maximum time for the operations it performs is referred to as hard real-time. In contrast, an OS that can usually perform operations in a certain time is referred to as soft real-time.

Jitter: The amount of error in the timing of a task over subsequent iterations of a program or loop is referred to as jitter. Real-time operating systems are optimized to provide a low amount of jitter when programmed correctly; a task will take very close to the same amount of time to execute each time it is run.

The main components in the functional diagram are the hardware and the kernel of the RTOS running on top of it and servicing tasks and interrupts that comprise the real-time application.

The function of OS are (i) task management (scheduling, dispatching, creation and termination of tasks etc.), (ii) synchronization (for resource sharing) (iii) interrupt handling (manipulate and monitor the interrupt descriptor tableIDT) to service hardware interrupts (iv) memory management (virtual memory and dynamic memory allocation) (v) programmable clocks and timers, and (vi) inter-task communication (sockets, pipes, FIFO, shared memory etc.).

**CHALLENGES**

RTOS developers face following challenges -

- Importance of time to market,
- Comparing design outcomes,
- Avoiding overqualified RTOS,
- Problem of delays in embedded applications.

These are some of the prominent RTOS that currently exist in the market –

<table>
<thead>
<tr>
<th>Name</th>
<th>License</th>
<th>Platforms</th>
</tr>
</thead>
<tbody>
<tr>
<td>LynxOS</td>
<td>Proprietary</td>
<td>Motorola 68010, x86/IA-32, ARM, Freescale PowerPC, PowerPC 970, LEON</td>
</tr>
<tr>
<td>RTLinux</td>
<td>GNU GPL</td>
<td>Same as Linux</td>
</tr>
<tr>
<td>TI-RTOS</td>
<td>BSD license</td>
<td>Primarily Texas Instruments: MSP430, MSP432, C2000, C5000, C6000, TI's ARM families (Cortex M3/4F, Cortex R4, Cortex A8, Cortex A15), SimpleLink Wireless MCUs (CC2xxx, CC3XXX)</td>
</tr>
<tr>
<td>ThreadX</td>
<td>Proprietary</td>
<td>ARC, ARM/Thumb, AVR32, BlackFin, 680x0- ColdFire, H8-300H, Luminary Micro Stellaris, M-CORE, MicroBlaze</td>
</tr>
</tbody>
</table>
CONCLUSION

RTOS has evolved to a great extent in the recent years. Every RTOS has a distinct feature of its own. But the growth in this field is not fully utilized by the developers yet as most of the RTOS is licensed and highly expensive. Peripheral support and stack availability also varies widely from one RTOS to another. Out of the few available free/open source RTOS, support for proprietary protocols is minimal. Developers do not have the luxury of time to adapt RTOS to project requirements. Hence they are unwilling to change from their current working condition. It would be greatly beneficial if all the project requirements are listed in the beginning itself, so that the developer can ensure that the RTOS chosen has support for all the necessary drivers and protocols.

The domain of real-time operating system has a very active area of research in recent years. The field has seen many RTOS's being built with many different ideas, principles and paradigms. However, additional practical experience with such RTOS's is desirable.

REFERENCES


Mrs Jyoti Kori
Domain in Charge
Embedded System Design