By Curt Schwaderer

CompactPCI & AdvancedTCA Systems

Execution partitioning for embedded systems increases security, reliability

Embedded systems software continues to grow in complexity. With the increase in performance and capacity of embedded hardware platforms, embedded software programming has grown to where it is not uncommon for an embedded system to reach or exceed 100,000 lines of code. In fact, one QNX Software Systems white paper refers to research suggesting that the code base for a typical embedded project doubles every 10 months. Increasing complexity, coupled with the fact that today's embedded systems are network-connected, leads to performance, reliability, and security issues that need solutions. This column is the first of a two-part exploration of adaptive partitioning from QNX, a leading Real-Time Operating System (RTOS) provider. Both parts of the column are available in full at www.compactpci-systems.com/software_corner. The discussion centers on how adaptive partitioning addresses security and reliability concerns.

Security and reliability

The concept of security for a network-connected device is a well-documented and well-known challenge for networked embedded systems. These issues strike at the very heart of the embedded system, the RTOS. Historically, two models of OSs have been used in embedded systems. One model is a simple threads-based approach where each thread has the ability to access any resource (memory or I/O) in the system, accidentally or maliciously.

The other model is a process model where each process in the system runs in its own protected memory space. The OS manages memory and I/O, and processes can only access memory, I/O, and other resources within the embedded system environment through formal methods using RTOS API calls. For low complexity, simple embedded systems, a threads-based OS has the advantage that everything can be directly accessed without the overhead of the OS management. But for the vast majority of today's embedded systems applications, a process model OS and the protection it provides against accidental or malicious corruption is a necessity. Most process model OSs are also multithreaded, so the developer can take advantage of the simplicity of thread interactions while still having the protection of a process model environment.

Another dimension is the development complexity involving process and thread execution not only from the memory and resource sharing point of view, but from the execution interaction point of view. When multiple teams are developing software subsystems to be deployed on an embedded system, finding and fixing stray memory pointers or improper use of I/O resources is only the beginning of the test process. Perhaps the most daunting task in the system test process involves the complex execution interactions between subsystems when run together on the same embedded platform. Subsystems that work fine running alone might spuriously error or malfunction when run with the entire system. These issues have led to the concept of partitioning CPU cycles in an analogous manner to process model memory and I/O partitioning. Figure 1 is an excerpt from a QNX white paper on partitioning that shows the options

Partitioning Approach	Product Cost	SW Development Cost	Time to Market
Hardware Partitioning	• Redundant hardware cost passed on to customers; results in less competitive pricing	• Less software complexity; requires less development effort	• Favorable time to market, but higher hardware costs
OS-controlled Partitioning	• Minimal processing overhead	• Minimal software complexity to provide CPU guarantees	• Favorable time to market assuming existing code base can be easily used
Application-level CPU Control	• Processing overhead consumed by application may require incremental hardware	• Complex design required to manage CPU allocation	• Complicated, multiparty design and implementation reduces productivity and slows time to market

Figure 1

a developer has when deciding how to implement the execution partitioning of an embedded system.

Execution partitioning

Execution partitioning means that each subsystem is developed and compartmentalized within a specific execution *partition*. The execution partition is guaranteed to be allocated the defined percentage of the CPU cycles for the platform. The idea is twofold:

1. Address security challenges – Execution partitioning minimizes Denial of Service (DoS) attacks by limiting the amount of CPU cycles applications can consume. Thus, malicious applications cannot totally consume the hardware processing resources.
2. Address system integration challenges – Each subsystem can be tested with a certain percentage of the CPU cycles allocated.

This twofold approach minimizes the effects that change the execution characteristics of the subsystem within the context of the entire system software.

Approaches to partitioning

Hardware partitioning (where subsystems execute on different processors within the embedded system) can reduce the software complexity, but adds significant cost to the product.

Application level partitioning increases engineering costs and complexity because software must be written within the application to implement the partitioning algorithms. This makes the application less portable, more complex, and does little to prevent denial of service attacks since the applications are governing themselves.

The OS level partitioning is a favorable mix of software managed execution partitioning with no impact on the application and software subsystem complexity. When this form of partitioning is done properly, the applications are not even aware of the execution partitioning happening in the system.

Therefore, the concept of execution partitioning boils down to identifying what percentage of the CPU cycles each task or group of tasks are to be allocated and, within the OS, implementing the algorithm to enforce the execution partitioning. For example, I may have some networking tasks that are very important to the system, while the graphics and serial I/O subsystems are of less importance. I can assign all threads and processes relating to the networking a partition to get 66 percent of the execution cycles while assigning 20 percent to the graphics tasks and 14 percent to the serial I/O tasks.

Both parts of this column appear in full at www.compactpci-systems.com/software_corner, where you will find more on how the OS schedules tasks for partitioning, on allocating execution partitions, and on the QNX Neutrino OS and the company's Momentics development suite.

For more information, contact Curt at cschwaderer@opensystems-publishing.com.

This column appears in full at:
www.compactpci-systems.com/software_corner

More information on QNX adaptive partitioning: www.qnx.com