Spatial Isolation and Integrated Modular Avionics (IMA)

Urueña, S.; Pulido, J.A.; Zamorano, J.; de la Puente, J.A.
Technical University of Madrid (UPM)

Spatial Isolation is a key requirement for Integrated Modular Avionics (IMA). It must be guaranteed that an application of a given partition will not corrupt the memory region of another partition, otherwise the whole system would have to be certified to the highest crititicality level. In addition, the failures detected by a fault-containment region \emph{inside} a partition are also an aid to find and fix hidden software bugs, thus reducing the development costs. Spatial isolation can be obtained using several methods, including *static information-flow analysis*, *run-time checks*, and *hardware memory protection*. The first two methods can be used as fault-containment regions *inside* partitions, but nowadays cannot be used to provide spatial isolation *among* partitions. This investigation is focused in employing the hardware fence registers of current space processors like ERC32 or the LEON family for IMA.

In the space domain nowadays all the embedded software is executed in supervisor mode, i.e. not only the kernel can execute privileged instructions but any application. This is the usual practice in real-time embedded systems when the computing resources are very scarce to avoid wasting CPU cycles in processor-mode switches. For example, both RTEMS and ORK have this architecture. Furthermore, all the code execute inside a single memory space, and all the applications are linked statically to a single binary (including the real-time kernel) regardless of their criticality. That is, all the executable code is linked into a single *.text* section, the global variables are located in the *.data* section, and the stack for each thread is created in the *.bss* section during initialization.

This model has several advantages, like increased CPU performance and memory footprint reduction. There is no code duplication because all the applications share the same code, including static libraries. The operating system can be simpler, e.g. there is no application loader. However, hardware memory protection cannot be used to provide complete memory isolation because all the global variables are located in the same section (*.data* or *.bss*) regardless of their criticality level. Only the stacks can have some memory protection because they are clearly separated in memory. Global data of different criticalities are grouped into the same memory region.

To implement spatial isolation using fence registers it is necessary to put the global data and stacks (and heap, if available) of each partition into separate memory areas. This way, when the operating system schedules a specific thread the kernel can provide write permission only to the data area of the partition of the thread. This work analyses the changes needed in the current real-time operating systems and compilation toolsets used in the space domain, able to provide spatial isolation with the adequate safety but without incurring in the footprint and CPU overheads of traditional operating systems. Namely, the advantages and disadvantages of different *compilation models* (Separate Compilation; Programming Rules) and *linking methods* (Custom Linking Script; Meta-Linker) are evaluated, concluding with the recommended compilation approach, and the set of changes needed in future ESA processors to provide the functionality needed in next-generation systems of the aero-space domain.