The Art of Separation

Electronic Design

Yi Zheng

Wed, 2015-01-07 09:26

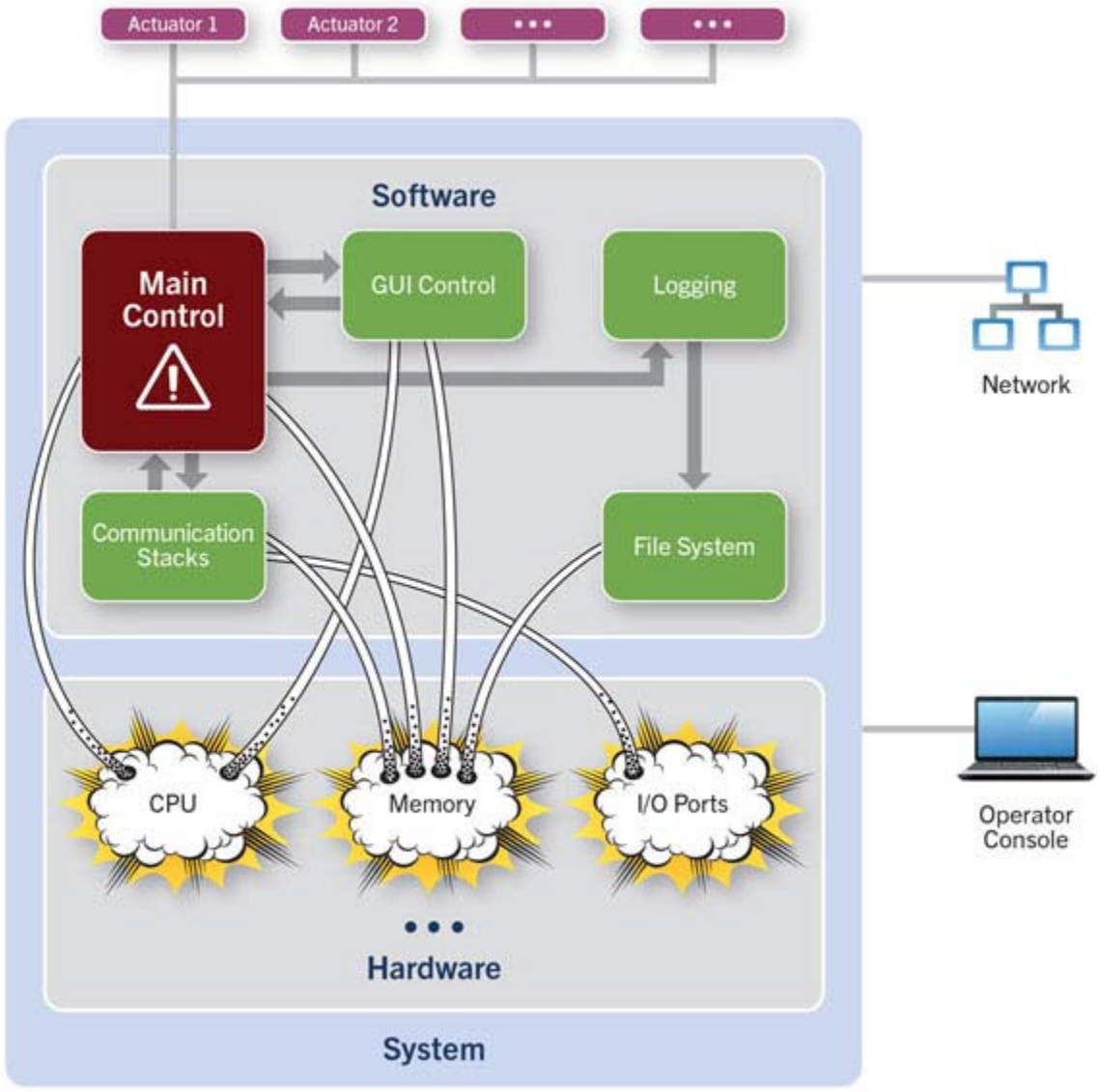
It may seem odd to see the word “art” in the title of a technology-oriented article. But when it comes to the topic of separation, “art” is the most apropos description of the status quo. If the science of separation existed, it would precisely prescribe the methodology for separating system functions, thereby saving system designers lots of headaches. Unfortunately, the science of separation doesn’t exist. Not yet, anyway. So for now, let’s take a closer look at the “art.”

The objects we’re trying to separate constitute parts of a system—for the purpose of this article, a system that consists of hardware and software. In the “good old days” when systems were simple, there was little need for separation. It would be pretty meaningless to separate one function from another in a single-threaded program running on a microprocessor designed to perform a simple task.

However, as powerful multicore hardware increases system complexity, meaning hundreds of thousands of lines of software code, the need for separation quickly becomes critical. This need intensifies when adding safety certification to the list of requirements to provide a publicly recognized level of guarantee of system reliability.

Seizing Control

The goal of separation can be boiled down to one word: control. Whether it is development costs, delivery schedules, future evolution, or product features and quality, various stakeholders—from senior management to programmers—want to take charge of the factors that matter most to them. In fact, separation is a strategy that influences controllability on almost every level. Shown is a fictitious system that controls an external actuator performing a safety-critical function *(see the figure)*. The system communicates with other systems over some form of bus and writes data into logs stored in a file system while sending information to be displayed on a graphical user interface (GUI).



The most simplistic design is to pile all functions together into a single software block and run it on a microprocessor. However, the time and effort saved to avoid a sophisticated system design will be negated many times over by the drawbacks of a system without separation. A malfunction of the data-logging function could negatively affect the control function and cause a critical failure on the actuator, which may, in turn, lead to human or property damage. Increasing system complexity heightens the probability of failure in critical components (due to external interferences). In fact, this is one of the key drivers behind the burgeoning safety standards for various markets in recent years.

Whether it is IEC 61508 for industrial automation, IEC 62304 for medical devices, EN 50128 for railway transportation, or ISO 26262 for passenger vehicles, the common goal is to ensure a defined level of confidence in the system's functional safety despite growing complexity. In addition to building a resilient system, safety standards add challenges for system manufacturers, too. Anyone managing a safety certification project understands the additional resource investment required, both in time and money, with no guarantee of certification.

Even though most safety standards contain multiple parts (for example, IEC 61508:2010 has seven parts and ISO 26262:2010 has 11 parts), their development impact primarily manifests in two areas: the development process and the actual design. It's in the design that the art of separation supports safe, reliable operation and

litates an easier path to certification. A closer look at the various ways to separate illustrates this point.

Hardware Separation

The most thorough method of separation comes via hardware. Using the previous example, one possible design puts the control unit functions on one microprocessor and the less-critical functions on another. Now the control unit is almost free from interference, except for what's brought by the communication links between the two processors.

When it comes to safety certification, only the control unit must be subjected to the process. This approach significantly reduces the amount of design work needed to address the standard's requirements. For example, the hazard and risk analysis (the starting point of most functional safety projects) is now limited to just the control unit. Likewise, the safety requirements, which are derived from the hazard and risk analysis, now focus on the control unit only (with the exception of the communication link between the control unit and the rest of the system). Without hardware separation, both the hazard and risk analysis and the safety requirements would cover the entire system.

Nonetheless, hardware separation has an obvious drawback: bill-of-materials (BOM) costs increase dramatically. Adding an extra processor for safety-critical functions is an expensive design decision that many systems cannot accommodate. Consequently, this thorough form of separation isn't widely adopted by designers and is most frequently found in systems with large price tags and high safety criticality, such as high-speed railway control units.

Related

Hypervisors And Separation Kernels

Software Process Standards Ensure Safe Transportation

Stabilize Software Upgrades in Critical Systems

A variation of the above approach uses different cores on a single processor to separate the critical and non-critical functions. Both approaches employ hardware for separation, to different extents; however, residual questions still must be answered.

In the two-processor scenario, there's concern about the communication link between the processors and how it could negatively impact the control unit. In the dual-core, one-processor scenario, additional questions arise. For instance, could non-critical functions affect shared resources in a way that causes a critical failure in the control unit? The pattern is clear—a more thorough separation leads to fewer residual questions regarding safety risks.

As mentioned, hardware separation is expensive and typically not a widely adopted separation technique. The competitive nature of the market makes cost reduction a constant issue for system manufacturers. The growing trend toward hardware consolidation renders hardware separation even less viable. Developers are often mandated to combine functions that were previously supported on multiple pieces of hardware onto a single, more powerful processor.

Software Separation

As a result, separation through software becomes an extremely important topic. For software separation to be effective, it must be applied on all critical resources, including CPU bandwidth, memory space, file descriptors, registers, access to hardware drivers, etc. The design decisions to achieve proper resource separation must be

de early in the software lifecycle, ideally during the system architecture phase. It's the system architect's ponsibility to factor in the separation requirements that lead to design decisions in later phases and serve as assumptions that developers can rely on during implementation.

Using memory space as an example, the system architect may decide, through simulation or other forms of analysis, that the control unit needs access to 40% of memory space. He or she can define a requirement to reserve 40% of system memory at system startup for the control unit. Designers may subsequently translate this requirement into an implementation decision that statically allocates 40% memory on startup into a pool accessible only by the control unit. Because of causality, the developer can safely rely on the assumption that after system startup, memory allocation for the non-critical units won't negatively affect the critical functions.

Static memory allocation isn't a common approach. Today, hypervisors are gaining momentum as a separation mechanism. Nonetheless, separation still requires some "art," because the hypervisor is just a tool. While it's important to have the right tool to finish a job, the outcome largely depends on the skill of the craftsperson.

Microkernel architecture, found in operating systems such as the QNX Neutrino OS, provides inherent separation of system resources, including memory space and CPU bandwidth (through the QNX adaptive time partitioning technology) — the two most important resources. Similar to the hypervisor, the separation effectiveness of the microkernel also depends on how it's put to use. With the hypervisor, some key design decisions require significant investigation and planning, such as the number of virtual machines to use, the distribution of functions into the various virtual machines, access privilege assignment (to hardware drivers) to a particular virtual machine, and management of shared resources to ensure safety integrity.

For example, placing all safety-critical functions in one virtual machine may not be an optimal design. Modern systems are sophisticated and functions rarely stand alone. If critical functions depend heavily on other (supporting) functions, placing both sets of functions on the same virtual machine can guarantee critical function performance and avoid unnecessary communication between virtual machines. The same type of discipline is also necessary when choosing a microkernel as the base platform. Instead of asking about virtual machines, the questions would revolve around the effective usage of processes, threads, privileges, and interprocess communications.

Each separation mechanism comes with pros and cons. The mechanism itself isn't nearly as important as the skill with which it's applied. The optimal design is based on a holistic view of the system. A skilled architect knows the right set of questions to ask for a design with mixed criticality levels to determine the most suitable mechanism. This process ultimately leads to a system design with efficient separation and an easier functional safety certification process. Unfortunately, the ability to ask the right questions doesn't come readily from a textbook. It'll be a while before the "art of separation' is refined to a science.

Yi Zheng is product manager for QNX Software Systems' safety and security products. Prior to QNX, Yi worked at Entrust Technologies, Autodesk, and Nortel Networks, designing a wide range of software applications. She holds a bachelor's degree in computer science from Carleton University and a master's in business administration from Queen's University, and is a certified management accountant.

Source URL: http://electronicdesign.com/embedded/art-separation