

Application Level Debugging

An operating system designed for distributed real-time systems needs to provide more than basic task manipulation. It should also provide assistance for developing both multitasking and multiprocessor systems. Distributed real-time systems often involve complex interactions. You need to see the control flow passing between tasks, whether they are running on one processor or on multiple processors.

Often, distributed systems have highly dynamic architectures. They may need this to support high-availability and fault-tolerance features. Alternatively, the architecture may call for upgrades to be added without restarting the rest of the system. Such systems need dynamic tools, able to handle the changes in system configuration as target nodes enter and leave the distributed environment.

As a result, the real-time operating system and its tools need to have better development facilities than a simple ROM monitor and task debugger does. OSE has been designed and instrumented so that the kernel can communicate important, system-level events to a comprehensive, graphical toolset. The target debug server has been designed to support a wide range of tools, including system browsers, profilers and event-driven debuggers, that facilitate the design of complex distributed real-time environments.

The toolset is not just designed for the product development phase. Maintenance procedures may demand the use of portable equipment that is not compatible with the workstations originally used to develop the software. Providing access to the same level of debug information across different forms of hardware platform is difficult using conventional tool architectures. By writing the entire Illuminator suite in Java, Enea has made it possible to run all of the tools on any platform able to run the Java virtual machine. As an increasing number of portable platforms appear that offer Java compatibility, such as portable digital assistants and electronic instruments, Illuminator can be integrated with them to provide engineers with a complete maintenance and debug environment.

Illuminator can communicate with the target using TCP/IP over a network connection or through a standard source-code debugger, maintaining the same user interface in either case. If Illuminator is used for field debugging, the target debug server is left in the application and the front-end software connects to it using a serial link or Ethernet. When connected, debugging can be performed on the entire application without halting the system.

The Illuminator suite is split into three main components: the Process Browser, the Evact Handler and the Profiler.

THE EVACT HANDLER

The name Evact Handler derives from the tool's ability to handle events and actions. It is the main tool used to debug interprocess communications and other interactions between tasks. It works by combining events with actions so that when an event is encountered, the related action is performed.

Events are controlled by the operating system and make it possible to track any changes in the system that it mediates. For example, an event can be a message sent to a process, a context switch or the act of creating or destroying a process. The action set to catch an event in the evact handler can be to monitor, trace, or move to another state. Catch actions let the user set breakpoints on an event, so that the developer can view the state of the system when a particular event occurs.

Often a sequence of events will define a system condition that is causing a problem. Those events that happen outside of this sequence can generally be ignored. The way that the evact handler works lets the developer set a monitor or trace in action when the event sequence is detected so that the operation of the system can be watched in detail. When the system leaves its problem state, is stopped, the Evact Handler can be set to stop collecting data.

The user creates states by defining a set of evacts. Only the evacts in the current state are active. Evacts in other states are ignored and are only enabled if an event triggers a move to their associated state. This makes it possible to track complex sequences of events and to collect information about specific events.

Actions such as stopping a process or

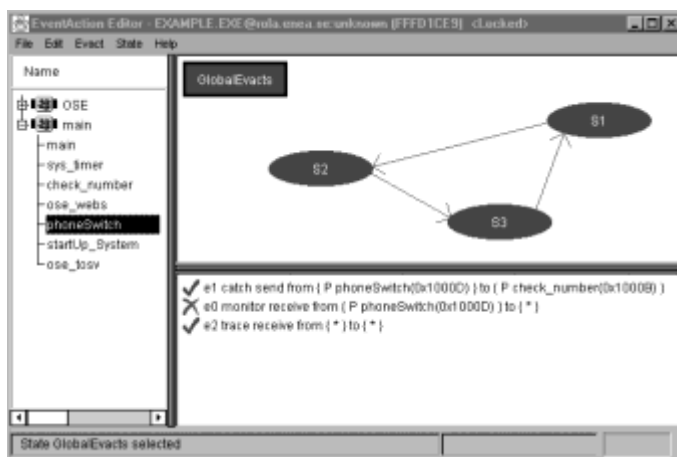


Figure 1. EventAction Editor

monitoring one can be applied to a limited set of processes, enabling parts of the application to continue to run unaffected, within a single processor or across several.

THE PROCESS BROWSER

The System Browser shows the structure of an entire distributed environment, providing access to every OSE-based target in the system. Because systems built using OSE are often dynamic, the System Browser updates its display to reflect new configurations as targets are added to the network and existing ones removed.

The browser uses an explorer-style interface, similar to that used on PCs. Double clicking on a target opens it, providing access to detailed information on the target. Processes running on the target can be queried individually. The browser will show information about OSE objects such as memory blocks and message queues.

It is possible to start, stop or terminate processes from the browser and messages can be created and sent on the fly to processes in the system to test its response to unexpected events and aid system testing. To make this process easier, message editing is performed symbolically, providing names of messages. Members of the data structure for the message are presented along with data values.

As applications run, they call on memory for data storage, interprocess communication and for the stacks of dynamically created processes. Memory buffers and stack locations are constantly being allocated and freed. Traditional debugging techniques do not show information on these processes. However, memory allocation errors and stack overflows are serious enough to cause the failure of a process in such a way that a user with typical debugger will find difficult to track down.

THE PROFILER

Because OSE is an instrumented kernel, it can provide the developer with vital information on the state of the system in terms of its memory usage. This information is presented to the user in an easy-to-understand, graphical fashion through the Profiler.

The Profiler shows how each target uses its memory. It can expose memory allocation errors, such as a process' failure to release memory properly and provide the information needed to optimise the target's memory configuration. Initially, for each selected target, the Profiler provides an overview graph of memory usage that shows the relative size of each pool, as OSE can support multiple memory pools on each target. These pools can be shared or dedicated to one process.

Additional overview graphs show how many buffers of each size and within each pool are allocated. The most common buffer sizes and areas where there may have been allocation errors are marked.

For each pool, the Profiler can provide more detailed information. In this view, the top ten processes using the pool are graphed and ranked according to the

number of message buffers owned or the actual number of bytes. An alternative view shows all the processes using the pool. Using these views, it is easy to find processes that have likely allocation errors as well as those that need more optimisation work in terms of their memory usage.

For example, the buffer size could be adjusted to bring the number of bytes used per buffer closer to the configured size. A powerful feature of the profiler, and only possible using OSE because of the way that the RTOS is designed, is the ability to search for messages matching certain criteria, and then sort the results.

This can help track down problems with deallocation failures by showing which processes continue to own message buffers long after they should have expired. Once the developer has this information, it should be possible, using a conventional debugger, to see which part of the code is not functioning correctly.

The second set of views supported by the Profiler provides a real-time picture of how an application is using the processor. It shows the distribution of processor usage between the processes that make up an application. As processes are swapped in and out of execution, the CPU Profiler keeps statistics that are processed and graphed in real time.

Using the Profiler, weaknesses in the design of the application can be identified and improved. It also allows more efficient use of the processor by showing where peak activity occurs so that non-real-time tasks can be scheduled to run where there is likely to be less activity. In this way, the Profiler can be used to demonstrate that the margins on processor usage are sufficient.

Processor usage can be viewed by process, or by priority and interrupt level. By process, the software can display the highest consumers of processor time, or those selected by the user. The view by priority makes it easier to alter priority settings for the most efficient processor usage. The charts can be displayed as a dynamic scrolling graph or as peak and average usage. Long-term statistics can be collected, showing the peak and average usage maintained since system start-up. Information can also be viewed in numeric form to help show extremes or very small or rapidly changing values that do not display well graphically.

Effective debugging for distributed real-time systems demands more than an effective source-level debugger did. The developer needs to be able to obtain a wide range of views of the system to be able to determine where efficiencies can be made and to track down bugs that would be almost impossible to find by other means. As an RTOS, OSE provides the necessary support for those tools. ■

Robert Largren is the Product Manager for telecommunications industry at Enea OSE Systems. Robert has been working with OSE for more than 10 years and in the embedded real-time world for more than 15 years. He has a Master of Science from the Royal Institute of Technology in Stockholm and his expertise is distributed high availability systems.