

From pSOS to VxWorks

Problems and solutions

Fri, Feb 13, 1998

While porting the VME/IRM system code to using the VxWorks operating system kernel, rather than pSOS, several problems have arisen that require solutions. This note covers some of these problems and proposed solutions. The switch to the VxWorks kernel is dictated by the desire to run the system on a PowerPC-based CPU board. Since the pSOS we acquired 15 years ago is suited only for the 680x0 CPU family, we needed to use a different kernel. Administrative considerations dictate the use of VxWorks at Fermilab.

Task-based event handling

The pSOS kernel provided support for tasks to await events. Each task could have up to 13 events any combination of which it could await. Any other task, or an interrupt routine, could wake up a task by sending it an event for which it is waiting. VxWorks does not support such a facility. A scheme has been devised to emulate such support, however, that is described in a short note called Task Event Support for VxWorks.

Low memory usage

VxWorks apparently takes over use of low memory for its own purposes, which means such memory is unavailable for the user. (In contrast, pSOS assumed the use of no memory except that granted it in its configuration parameters.) There is some history here. The system code has long made use of fixed low memory addresses for system global variables and tables. These were convenient for many diagnostics and maintenance purposes. With VxWorks, we cannot use the same low memory for these same purposes. This will complicate the maintenance of the new nodes, as they will have to be treated differently by diagnostic programs. (Note we are not discussing system tables, which are addressed indirectly through the system table directory.)

Of course, access to memory from outside a system is done via memory data requests sent via the network. For such requests, it would be possible to consider mapping accesses to low memory addresses to another area. In this way we would pretend that the same low memory addresses are used in the same way. Doing this should not lose much, since VxWorks use of such memory is unknown, so that there would not be any reason to want to examine it. The scheme would be to allocate the same layout of low memory as is used in the present systems that includes both 68020/MVME-133 and 68040/MVME-162 support. If a data request for low memory is received, the data that would be returned would come from a different VxWorks-permitted area using a fixed offset.

Network transmissions

In the current system, support is given for messages that are queued to the network. When the queue is flushed, the messages to be sent to the same target node and UDP port number are combined into a single datagram as can fit within the maximum datagram size supported. The datagram is then queued to the network hardware, and the next set of messages from the network queued are similarly combined and queued to the network hardware. This continues until the queue is empty. This allows the task that flushed the network queue to proceed to completion without waiting for the datagram transmissions to completely be delivered to the network. One might call this a buffered output scheme. When the ethernet transmit interrupt routine is executed, a check is made for another entry in the

network hardware queue, and if there is, it is passed to the ethernet hardware. This proceeds until the hardware queue is empty. The hardware queue is emptied until interrupt control, even as the task-level code is permitted to construct more datagrams to be added to that hardware queue. Of course, if a datagram is queued to an empty hardware queue, it is passed to the hardware immediately.

VxWorks apparently does not support buffered output to the network. When a task makes a call to the *sendto* socket routine which transmits the UDP datagram, return is held up until the transmission is complete. This means it is not possible to continue building new datagrams to be transmitted. One could consider dropping to a lower priority for the call to *sendto*. This would permit other task activities to proceed without waiting for the datagram to be transmitted, but it would inhibit continuation of the current task, and thus forestall building more frames to be transmitted, as well as any other activities performed by that task. *We need to find a solution for this.*