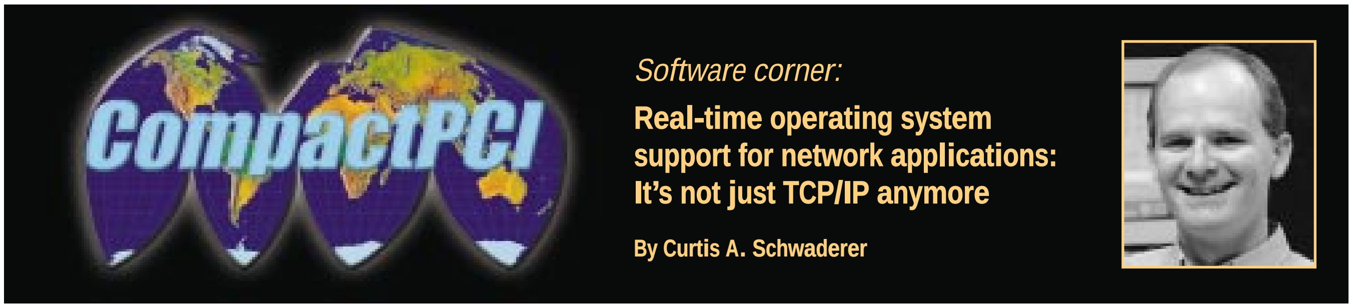


Software corner:

Real-time operating system support for network applications: It's not just TCP/IP anymore

By Curtis A. Schwaderer

This issue's column will look at various aspects that can affect the effort and development risk of a communications software product and how various commercial RTOSs address these issues.

Historical telecommunications systems revolve around centralized switching equipment with little or no intelligence outside the core network cloud. The Internet has polarized data communications around the IP protocol and a distributed network where all connected nodes have intelligence. Broadcast systems are yet another force shaping the future of network systems, bringing with it the capability of video and wireless transmission. All these things can be summed up with one word for today's software developer — complexity. You don't have to be creating something formally classified as "network equipment" to feel it; any product that communicates with external equipment is touched by this increasing complexity.

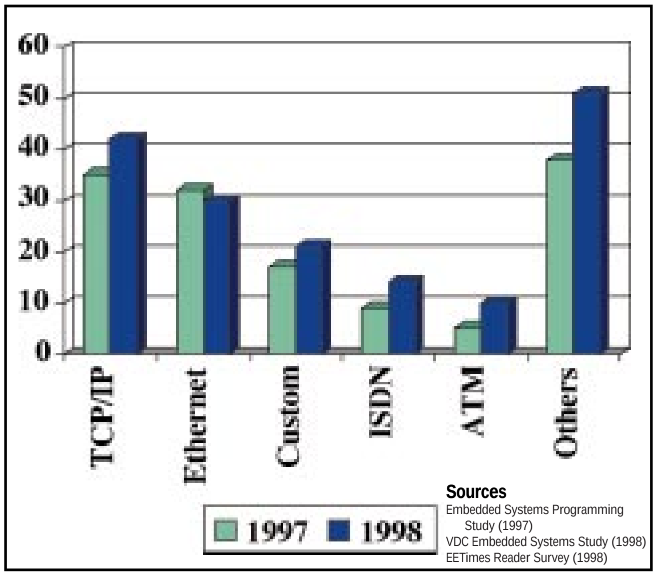
The cornerstone of the software developer's arsenal is the real-time operating system (RTOS). Contrary to popular belief, all RTOSs are not "created equal". That is, the design philosophy and capability set of every RTOS is different. What the RTOS does and doesn't include makes it a better or worse fit for a particular application. Whether it's an internally developed OS or a commercially available RTOS, it's important to have the right capability set in the RTOS to minimize the network complexity.

The areas of complexity that will be focused on include:

- Ease of development, integration, and adaptability; a communications framework that accommodates current and future network software requirements
- Required protocols and how they integrate and interoperate with each other and the RTOS
- Performance. Can the software solution perform in terms of maximum throughput and minimum CPU utilization parameters?
- Reliability/availability. Can a single bug/failure in the software bring the entire system down, or is the software environment fault tolerant?

Ease of Development

A communications framework that is integrated with the RTOS can go a long way to simplifying communications software development. A 1997 worldwide study by the Technology Institute of Finland cited one of the most important areas to address was networked embedded systems and their applications. The study concluded that one of the core software competencies required is a software architecture that integrates embedded communication and control. The first step to realizing this in a commercial RTOS is the presence of a defined I/O system. Many commercial operating systems have no formal I/O system. The only framework that exists is a task or thread context that can be used to create a framework that communicates with a protocol stack and network interface. Without a formal network I/O system, it's difficult to incorporate much more than a single-purpose network stack. If any custom protocols or processing is required, an RTOS with no I/O system makes this difficult. Operating systems like Microware's OS-9, Linux, and Lynx's LynxOS have formal I/O systems that facilitate the creation of a communications framework. Microware has gone one step further by actually creating a full featured communications software framework called SoftStax™ and offering it with the OS-9 operating system package. Operating systems like VxWorks, pSOS, and VRTX do not have formal I/O systems, making communications development more difficult.



Sources
Embedded Systems Programming Study (1997)
VDC Embedded Systems Study (1998)
EETimes Reader Survey (1998)

Figure 1

Protocols

Figure 1 shows the percentage of use for various protocols in networked embedded applications. One observation shows that TCP/IP protocol usage has risen by almost 10% while Ethernet usage has declined slightly. This means that TCP/IP is seeing increased usage on other network topologies. This is not surprising considering the number of Internet appliances that use other WAN and wireless technologies to access the Internet. Another interesting observation from the chart is that a growing number of applications require custom protocols. All viable RTOS packages come with TCP/IP on Ethernet, but how easy is it to use TCP/IP over other network topologies? This is something that must be considered. Most operating systems implement device drivers that can be invoked by the IP layer so the creation of TCP/IP based systems over other network topologies isn't too bad, but what about custom protocol creation and interoperability? This is an area of major concern for many software developers. The Microware OS-9 RTOS seems to win out here, mainly because protocol interoperability is linked to the availability of one common communications framework for the RTOS. Microware provides a "null layer" protocol driver in the product that allows the developer to implement protocol processing in this module. All other protocol layers for OS-9 are written with the same framework, so the custom protocol is immediately interoperable. There are vendors such as Mentat who has created a STREAMS framework for RTOSs such as VxWorks. This additional product coupled with VxWorks can also alleviate custom protocol interoperability problems. The difference between the Mentat approach is that the STREAMS implementation uses the task framework layered on top of the RTOS while the Microware approach for OS-9 uses the driver-based SoftStax framework integrated with the RTOS.

Performance

The advent of faster network topologies such as fast Ethernet and ATM has caused the industry to rethink networking software development. The historical approach has built-in inefficiencies such as multiple context switching between the application and tasks implementing the protocol stack(s) and associated critical section protection. These inefficiencies are magnified on higher speed network topologies. This factor, in conjunction with the requirement to increase throughput and carry real-time data, causes software engineers to really look at software solutions and how they are addressing performance.

For the past six months, I tracked fast Ethernet performance on Microware Systems OS-9, Integrated Systems pSOS, and Wind River's VxWorks operating system. The OS-9 TCP/IP stack employs a driver based TCP/IP stack where pSOS and VxWorks use a task-based TCP/IP stack. Two reference customers had demanding throughput requirements on the Motorola 860T processor and these three operating systems were benchmarked over time to see which solution could meet the throughput requirements. See Figure 2 for the PowerPC 860T, 47 MHz performance comparison. OS-9 is currently ahead in the performance battle attributed to the driver based architecture the BSD v4.4 TCP/IP stack lives within and the tuning Microware has done to the TCP/IP environment itself. The pSOS operating system started out as the highest performing, but has remained constant over the past six months. VxWorks is not known for performance, so it's understandable that VxWorks lags behind in the performance area.

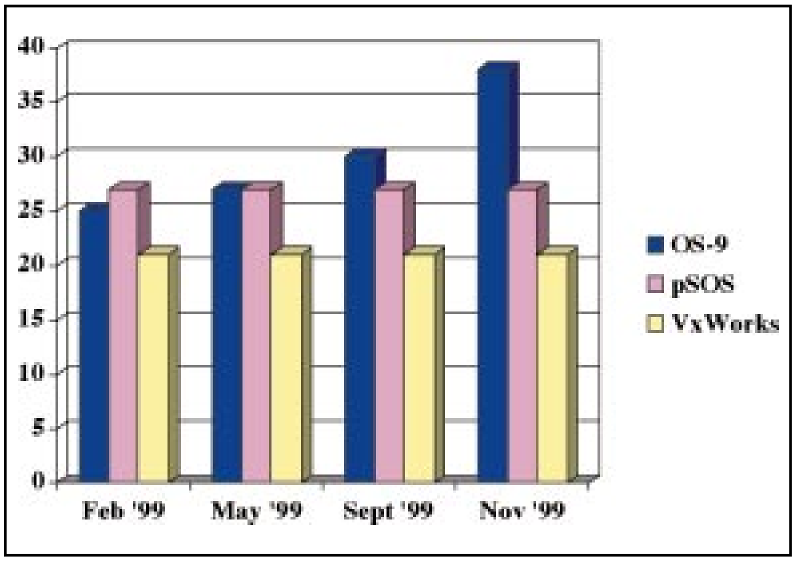


Figure 2

Reliability/Availability

As embedded applications become more complex, the number of inherent bugs in the system increases. Accepting the fact that there will be software bugs buried in the final product causes the designer to focus on how the software baseline handles a manifestation of any of these bugs that occur during operation. This is where it's important to understand how vulnerable the operating system is to accidental or malicious system corruption. Support for a memory management unit (MMU) is critical for high reliability systems. Some operating systems offer MMU add-on packages, but this requires the application to be written correctly to use the MMU itself, which defeats the purpose of inherent memory protection. There are a variety of operating systems that focus on high reliability environments. These operating systems employ a "process model" approach as opposed to a "threads based" approach to make use of the MMU, and provide memory and resource protection between processes. Operating systems such as QNX, Enea OSE, Microware OS-9, and Lynx LynxOS employ system security and memory protection. Even WindowsCE is a process model operating system, while not intended for high reliability applications right now, it's interesting to note that this new operating system was designed with process model support. Enea OSE and Microware OS-9 also have the ability to add, remove, and replace software components while the system is online and in use. This provides an additional measure of availability, even if the system is being maintained or upgraded.

Conclusion

Just about every software package today has integrated TCP/IP/Ethernet support. As we've discussed, this tells little about the software package's ability to accommodate or interoperate with other network topologies and WAN/custom protocol layers. This consideration in addition to performance and reliability and availability requirements can save significant development time and headaches for network software developers. As you begin work on new projects, I urge you to consider these issues. With the right match of capabilities to your requirements, a successful product can be launched and you may even get home a little earlier as well!

***Curtis Schwaderer** has studied and been involved in the development of networked embedded systems for over 15 years. In this column, Curt discusses four issues critical to successful software development of a networked product and the level of support various RTOSs have for these issues.*
Curt can be contacted at curtsch@juno.com.