

Asynchronous direct message passing rapidly gains popularity

by David Kalinsky, Enea Embedded Technology

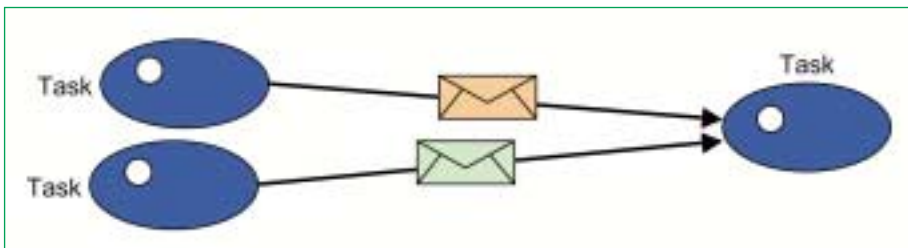


Figure 1. Asynchronous direct message passing model

Users of real-time operating systems are increasingly basing their software designs on message passing, since it very naturally provides a clean separation between software components such as tasks and processes. Message passing has long been popular in multi-processor embedded systems, and in designs where a prime architectural objective is the support of High Availability. Its popularity is also growing rapidly in single-processor systems. Other traditional RTOS mechanisms for intertask communication such as mailboxes, semaphores, mutexes, pipes and event flags are error-prone and become unwieldy when extended into distributed and multi-core embedded applications.

A specific style of message passing called “asynchronous direct message passing” is increasing in popularity most rapidly. In this style, tasks send messages directly to one another, as

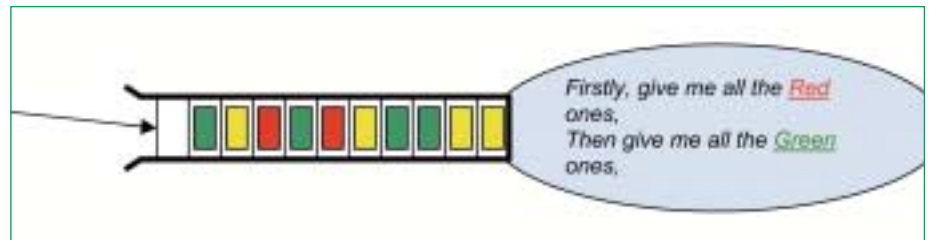


Figure 2. Arriving message can queue in asynchronous direct message passing, if there is congestion at the recent task.

shown in figure 1, without intermediary RTOS mechanisms such as “mailboxes” along the way. Software architectures done in this style are elegant in their simplicity, resulting in higher quality and greater maintainability. Software components are loosely coupled and asynchronous: tasks that send messages do not wait for the tasks that receive them. Interfaces between the communicating components are narrow and well-defined.

In distributed multi-CPU and parallel processing applications, it has long been well-known that there is great benefit in passing data between tasks on different CPUs without requiring software on the different CPUs to synchronize with one another. A synchronization requirement for communication could well lead to long delays while a task on one CPU waited to “rendezvous” with a task on another CPU as part of the data transfer. Asynchronous direct message passing RTOSs nowadays pro-

vide very high performance and are easy to program, debug and maintain; hence they are valuable for single CPU systems as well. As systems grow in complexity, they often evolve from single-CPU to a multi-CPU designs. When they do, they are easily partitioned if they were initially designed using the asynchronous direct message-passing model.

Most traditional RTOSs do not support the sending of messages directly from one task to another, but instead require that an intervening message queue or “mailbox” must first be created as an independent RTOS “object”. On the other hand, in an asynchronous direct message

passing RTOS, application software need not set up a queue or “mailbox”. When messages arrive at the receiver task more quickly than they can be handled, the RTOS automatically establishes a queue for the recipient task and queues arriving messages until they can be processed. This queue is not an independent RTOS “object”, but rather just a part of the receiver task. (See figure 2.) When there is no congestion of messages, there is no queue and messages are delivered straight to the recipient task.

Asynchronous direct message passing RTOSs are simpler to use than traditional RTOSs. Hence application design is simpler, and system debugging is simpler. Applications can be developed more quickly and reliably, and they can evolve into multi-CPU systems without major upheavals in application code. They represent a mega-trend in RTOS technology that can take us into a new era of increasingly complex embedded systems. **b**

**ESC - Modul with Linux.
Get independent!**

www.dilnetpc.com

ESC

ESC

ESV Embedded Systems kge@ist1.de Tel.: +49-511-40000-45 Fax: +49-511-40000-40

Visit us at **electronica 2004**
Hall A6 • Booth 642