

High Availability Provided in Software Using Supervised Logical Channels

Using Supervised Logical Channels leads to embedded and real-time application that are more robust regarding to failures and upgrades of hardware and software thus achieving higher performance, lower costs and providing better runtime maintenance. The term "High Availability" used more and more, is the characteristic of such a system.

INTRODUCTION

Modern embedded and real-time applications are becoming bigger and more complex, are being used in more critical applications and are generating more revenue from the services they provide than ever before. This is the direction the manufacturers of these applications are moving in order to address a market which is demanding higher performance and reduced costs. How can the manufactures best meet this increasing demand? How can they reduce the risk inherent in developing such complex systems? What technologies are important and how can they be applied? These questions will be answered in time, but we can look at what can be done today.

That the embedded and real-time markets are demanding higher performance and lower costs isn't any new observation. It's something just about every high-tech market is demanding. What is interesting is the combination of these two demands, and how they effect the systems manufactured for the market.

Providing Higher Performance

Higher performance could be achieved by building larger and more powerful machines. Consolidating a system, perhaps using one of the 64-bit RISC processors could be a solution? But, this path presents problems, not the least of which is it doesn't address embedded applications. However, cost is the most important factor. Larger machines are more expensive to build. If they fail, the whole system may fail and need replacement. They are difficult to scale to different situations. If a customer needs just partial functionality, the large machine is still required, which may represent five, ten or hundred times the capability he requires.

A better way to achieve higher performance is to distribute processing over many CPU's. The divide and conquer approach. By using multiple CPU's, the problems mentioned above can be effectively addressed. Smaller, power efficient CPU's allow building embedded systems. The unit cost of these smaller systems is less because of reduced CPU cost, memory requirements and chip count. If one unit in a distributed

system fails, a well designed system will continue to function, and failed units can be replaced at a lower cost.

By dividing an application, each component of an application can be larger, operate faster and have more functionality than if it were tightly coupled with the larger application and executing on one CPU. A distributed system can easily be expanded to add functionality or be expanded laterally by adding units in parallel.

Runtime Maintenance

Another argument against the large machine approach is the ever increasing need for runtime maintenance of a system. In many systems today, it is unacceptable to bring a system down for maintenance. In larger systems, the down-time required for maintenance, upgrading, reconfiguration, etc. can bring the system down for good! The benefits of runtime maintenance are numerous. Systems can continue to generate revenue even during maintenance. If they are part of a large or huge system, a petroleum processing plant for example, the entire plant can't be stopped just to fix a bug.

Some form of runtime maintenance has long been recognised as a requirement in some systems. It has been implemented in many ways; most less than adequate. Specific problems were addressed in custom ways which were not extensible to more general application. A general solution is needed to solve all these problems. The solution should not just make certain types of systems possible, but also add value to systems. For example, provide redundancy in software, add runtime debugging or allow systems to be distributed geographically.

The combination of the market demands for higher performance and lower cost leads naturally to the requirement of various forms of runtime maintenance. A term that is used to describe systems with this characteristic is "High Availability". The term is being used more and more as market demands increase in this area.

A better way to achieve higher performance is to distribute processing over many CPU's.

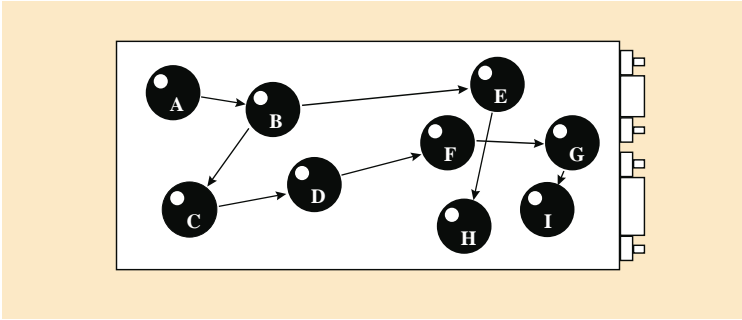


Figure 1. The Logical System

HIGH AVAILABILITY

High Availability is defined as a characteristic of a system which is robust with respect to certain runtime events. The types of events considered here are those associated with failures and upgrades of hardware and software. Complete Fault Tolerance requires redundancy in hardware to satisfy the definition of Fault Tolerance, which is "No single point failure" will bring down the computer. The goal of High Availability, on the other hand, is to provide flexibility, like runtime upgrades of hardware and software and runtime de-bugging. It is also a goal to provide redundancy in software for certain types of failures, like with networks and storage media. The usability and functionality of a system can be greatly increased without the limitations and costs associated with complete Fault Tolerant Systems.

PHYSICAL AND LOGICAL CHANNELS

When two entities communicate, a Physical Channel of communication can be defined. This includes the entities themselves and any hardware or software used to pass the data. A Logical Channel

is a reference to the underlying Physical Channel without concern for the actual path or implementation. This could be compared to a telephone call. The Logical Channel in this case is between the caller and the receiver. The Physical Channel may be implemented with mobile phones or wall phones, various connections with the operator and many other devices and software. It is desirable for the users of the telephone to think in terms of the Logical Channel as opposed to the complexities of the actual implementation.

A Physical Channel is defined as a data communication link between two processes executing under a multitasking operating system in two different CPU's connected by a data bus. The Physical channel is made up of the two processes and the complete chain of hardware and software through which the data passes. The Logical Channel, however, consists only of the two processes. It is the Logical Channel that the application developer sees. Simple operations can then be carried out like looking for another process or checking its existence, sending data to another process or receiving data from it. One process

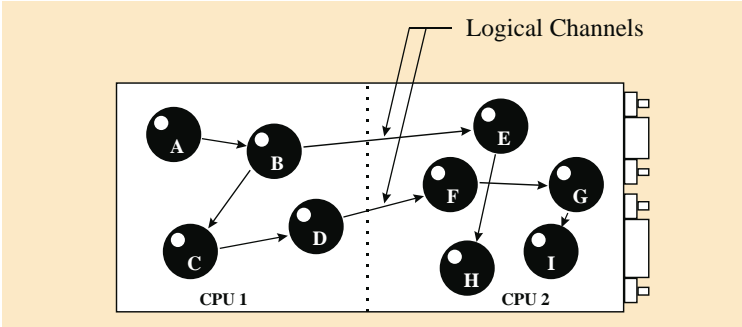


Figure 2. The Physical System

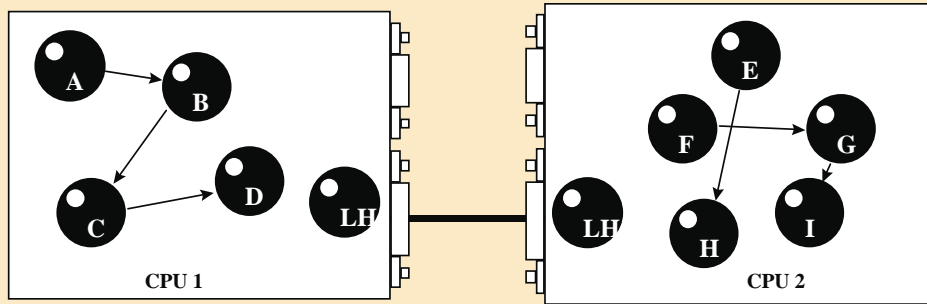


Figure 3. Link Handlers

sends data to another without concern for where the other process is located, or how the data is transferred. The complexities of the Physical Channel are hidden, thereby making programs easier to write and understand.

SUPERVISED LOGICAL CHANNELS

Once a Logical Channel is created, it can be supervised. Supervision means that if an event occurs which prevents one processes from communicating with the other, the process which requested supervision shall be notified. Supervision is not an easy task and must be implemented on many different levels if it is to be comprehensive. The supervision appears to a developer to occur on the logical level, but since failures occur on the physical level, it is the hardware and software over the entire Physical Channel which must be supervised.

UPGRADING SOFTWARE: AN EXAMPLE

Consider for example the case where a system is required to support runtime software upgrades. Assume the upgraded software can be downloaded to the code memory of the running target. This can usually be accomplished without too much trouble because the download service in the target is just another component of the application. It is idle until

the time comes to download software when it begins to operate and perform its download function. The difficulty comes when switching execution from the original, executing, software to the upgrade software while still carrying on with the duties and timing requirements of the application.

High Availability is defined as a characteristic of a system which is robust with respect to certain runtime events.

The Logical system

This example system could be described as in Figure 1, along with an operating system which provides typical services like process scheduling and inter-

process communication. Each circle represents one process or functional unit and the arrows between the processes represent inter-process communication. The system can be designed without concern for the number of CPU's, or which processes will execute in which CPU.

The Physical System

In Figure 2, the system is divided to allocate software functionality to hardware. Each part resulting from the division is mapped to one CPU.

Logical Channels are shown, but an underlying Physical Channel is required to communicate data between the two CPU's. Part of the Physical Channel is the data communications link between the two CPU's. This could be any bus or network. A process could be added to each CPU to drive the link. These

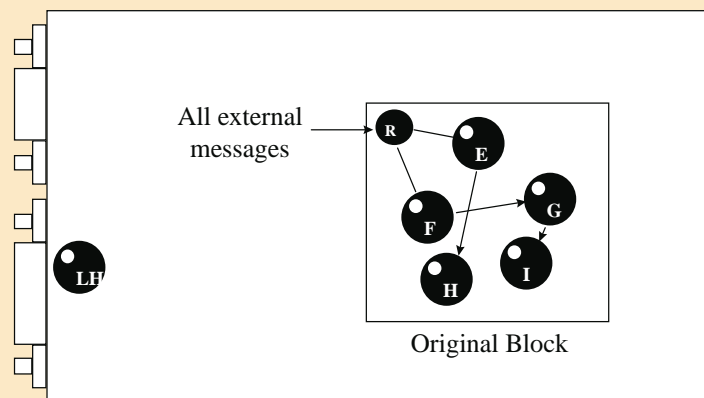


Figure 4. Only one process is accessible by the rest of the system

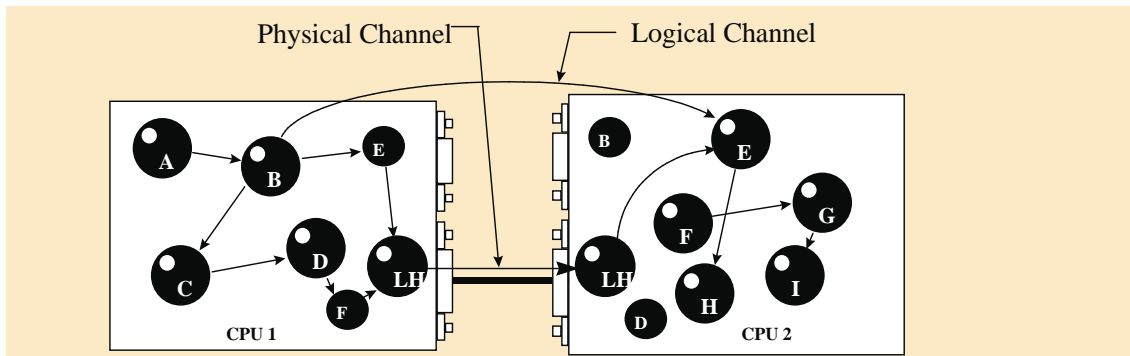


Figure 5. Both LH's create local images of the processes located in the remote CPU

processes are called Link Handlers (LH) as shown in Figure 3.

In this example, the processes E, F, G, H and I make up the original software which is being upgraded. They will be replaced by the upgrade software. The upgrade software does not necessarily need to perform the same functions or have the same design or structure as the original software. This implies that it does not need to have the same processes (E, F, G, H and I). However, in our example system, we have defined two Logical Channels. One between processes B and E, and another between processes D and F (Figure 2). It is necessary to maintain these two Logical Channels by providing the processes E and F in the upgrade software.

An alternative is to have only one process accessible by the rest of the system. This process handles all communication to the processes within the block. It operates as a traffic cop. As messages are received, they are routed to the appropriate process according to the type of message. In figure 4, the process R performs this function.

Following this methodology allows the internals of a block to change dramatically while still maintaining the same appearance to the rest of the system. This is very important in a system which will be upgraded at runtime.

We'll assume the example system is implemented this way, so that instead of establishing logical channels between B and E, and D and F. B and D establish the channels with process R. R must then exist in the upgrade software.

Requirements

Consider some of the requirements of the download operation. The original software must be stopped. The upgrade software must be started. Data and the state of the original software may need to be passed to the upgrade software. Resources owned by the original software must be cleaned up such as allocated dynamic memory and open files. Any processes which communicate with the original software must begin operating with the upgrade software.

Two fundamental problems exist in these requirements which must be overcome.

- The rest of the system, that which is not being upgraded, must be able to determine or be noti-

fied when it can no longer communicate with the original software.

- The rest of the system must be able to locate and establish communication with the upgrade software. Overcoming these two problems has many benefits and allows building High Availability systems.

Establishing a Communication Channel

To demonstrate an approach of how to accomplish this, I will begin with the second of the two problems. The rest of the system can locate and begin communicating with the upgrade software by establishing Logical Channels. Let us take the establishment of a Logical Channel between process B in CPU 1 and Process R in CPU 2. Process B needs to communicate with process R. To do this it requires the process ID of process R. If process R existed in the same CPU, the ID would be immediately available and no other action would be necessary. In this case there is little difference between the Logical and Physical Channels. The normal inter-process communication within the CPU is used.

But, since process R is not located in the same CPU as B, the process ID of process R is not immediately available. In order to get the process ID of process R, an inquiry must be made within CPU 2. This is called a remote system call. The inquiry is passed from CPU 1 to CPU 2 between the Link Handlers. It is also beneficial if the inquiry can be left pending in the case where process R does not already exist. If and when process R is created, the inquiry is fulfilled and the process ID returned.

When the inquiry is fulfilled, it is guaranteed that process R exists and the LH in CPU 2 receives the process ID of process R. This is the first in a sequence of actions to build the Physical link between process B and R which occur when the process ID of process R becomes known. The LH in CPU 2 passes the result of the inquiry to the LH in CPU 1. Both LH's create local images of the processes located in the remote CPU as in Figure 5.

The LH in CPU 1 creates a process in CPU 1 which serves as a local image of the process R. The process ID of the local image is returned to process B. Likewise, the LH in CPU 2 creates a local image of process B. The local images serve only to redirect

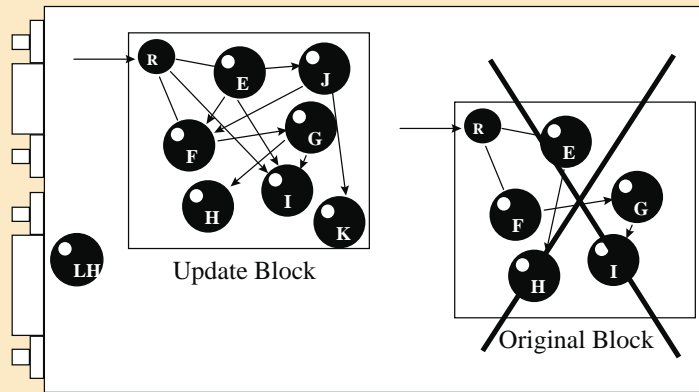


Figure 6. Upgrading the software

messages received by the LH. This completes the Physical Channel so that the Logical Channel can be used by the application.

How the Channel Works

A message sent from process B to process R travels to the local image of process R where it is redirected to the LH. The message is passed to the LH in CPU 2, which forwards the message to R. Since in this model, it is possible for process R to reply to the sender of the message, process B, the LH in CPU 2 indicates that the sender of the message is the local image of process B. If process R replies to the received message, the reply travels to the local image of B, to the LH, to the LH in CPU 1 and finally to process B.

The important points are that process B has located process R, and that in all aspects, communication between the two processes appears to be within the same CPU.

Establishing Supervision

Returning to the first problem - "The rest of the system, that which is not being upgraded, must be able to determine or be notified when it can no longer communicate with the original software." With the Logical Channel between process B and R, process

B must be notified whenever anything happens which breaks the Physical Channel between the two processes. In this example, process B should be notified when process R in the original software is removed. In the general case, B should be notified whenever communication is not possible. This could also occur if the bus between the two CPU's is disconnected or fails, or if CPU 2 were to fail or be removed.

Process B can be notified as follows. Guaranteed that the local image of process R will only exist when the entire Physical Channel is functional. Then, process B needs only be notified when the local image of process R is removed. The request for supervision of the local image of process R is a local request - within CPU 1 - and is serviced by the Operating System. If the local image of process R is removed, process B is notified and can take appropriate action.

The local image of R represents the entire Physical Channel. It will be removed under the following circumstances. If the LH in CPU 1 detects a fault in the communications with CPU 2 which inhibits communication, it must immediately remove all process images residing in CPU 1 of processes in CPU 2. The LH in CPU 2 functions identically, but must also request supervision of process R. If process R is removed,

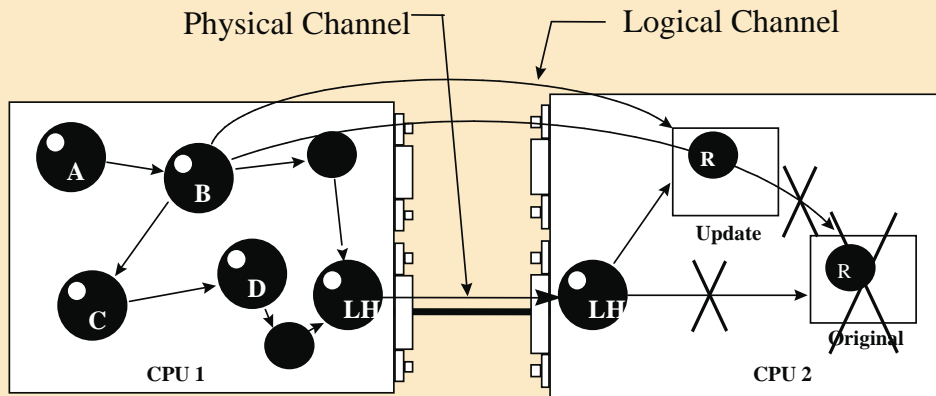


Figure 7. Process B proceeds to re-establishing the Logical Channel with the upgraded process R

Enea Data Ad

the LH in CPU 2 is notified. It in turn notifies the LH in CPU 1, which removes the image of process R.

So, complete supervision of the Logical Channel between process B and R has been created. Process B can communicate freely with process R as if it were resident in the same CPU, and it will be notified when it can no longer communicate with process R, i.e. when process R is removed to be replaced by the upgrade software.

Completing the upgrade.

Two options exist when upgrading software. The original software can be removed first and then the upgrade software downloaded and started. Or, the upgrade software could be downloaded and started before removing the original software. The second option is the most desirable from a performance standpoint because the switch over takes less time. However more memory is required to hold both executing versions simultaneously. The first option requires memory to hold only one version at a time, but neither version is executing during the download.

Functionally, both options are the same. It is only the order which is different. The case where the upgrade software is downloaded and started before the original software is removed is that considered in this example. See figure 6.

The original software continues to operate while the upgrade software is downloaded. After the download, the upgrade software is started. This software is written to be non-intrusive. That is, even though it is executing, it should not assume it is the only copy. It can wait for a start signal or just wait for data to be sent to it. It can however prepare for operation by establishing Logical channels and initialising itself.

In the mean time, the original software is removed. As stated earlier, when process R is removed, the LH in CPU 2 is notified because it has requested supervision of R locally. It in turn notifies the LH in CPU 1 which removes the image of process R. Process B is notified that the local image of R no longer exists. In figure 7, process B proceeds to re-establishing the Logical Channel with the upgraded process R.

As the new Physical Channel is established, new image processes are created for R and B, and process B receives the process ID of the new image of R.

In this example, only the Logical Channel between process B and R has been discussed. Of course many Logical Channels may exist. One must however use a good system design approach. It is most likely that even though Logical Channels are an absolute necessity in many situations their use should be limited to important interfaces between major blocks of processes which implement larger functional entities of an application. It probably is not a good idea to try to design a system so that every process could be upgraded individually, although this is possible.

CONCLUSION

As with most implementations, Supervised Logical Channels could be implemented with alternative approaches. Systems exist today which provide just the types of services presented here. The important issue is that the solution provides an easy to use a well defined interface for the application developer. A solution which needs to be rewritten for each new application, or that renders the application code useless with each new application is no solution at all.

Application code which is written to use Logical Channels should function regardless of the underlying Physical Channel. It should also be possible to reuse code in future applications, even if the hardware configuration has changed.

William J. McCombie is 37 years old and is the Manager of International Operations at Enea OSE Systems. He has been working with support and marketing of OSE Real-time Operating Systems for four years, and was before this employed at various software companies in USA, where he also graduated as M.Sc.

ADVERTISEMENT INDEX

COMPUTER PRODUCTS	25	INTEGRATED SYSTEMS	123
DIGITAL EQUIPMENT	17, 47	NATIONAL INSTRUMENTS	71
DSP 97	113	QNX SOFTWARE SYSTEMS	75
ECHTZEIT/MESSCOMP	63, 120	REAL-TIME ENCYCLOPAEDIA	65, 67
ENEA OSE SYSTEMS	21	RTS 98	105
FORCE COMPUTERS	124	RTUSI	2
GREEN HILLS SOFTWARE	51	SYSTRAN CORP.	69
GREENSPRING COMPUTERS	61	US SOFTWARE	13
GROUPIPC	109		

To reserve your advertisement, contact Alexander Teetaert at:

Tel: 32-2-520.55.77 or Fax: 32-2-520.83.09 or Email: info@realtime-info.be