

Distributed Real-time Micro-kernel with Fault-tolerance: DREAM

Kun Mean Hou, Thierry Garcia, Emmanuel Mesnard and Philippe Kauffmann, Laboratoire LIMOS, ISIMA, UBP Clermont-Ferrand II, France.

Email: {kun-mean.hou, emmanuel.mesnard@isima.fr, thierry.garcia1@libertysurf.fr, kauffman@custsv.univ-bpclermont.fr

Abstract

DREAM is a super small distributed hard real-time micro-kernel based on LINDA concept. The key features of DREAM are: pseudo process and tuple migration, DSP hardware tuning and ease fault-tolerant applications implementation by using different approaches such as PB, DMR, TMR. In this paper, we present the basic concepts of DREAM that allow pseudo process and tuple migration. A case study of fault-tolerance (P/B approach) is implemented on TMS320C31 starter's kits.

Keys word: Super small distributed hard real-time micro-kernel, Fault-tolerance, Dynamic reconfiguration, LINDA, Pseudo process and tuple migration.

1. Introduction

Nowadays fault-tolerance real-time processing is necessary for many complex applications: aircraft navigation systems, process controls, intelligent vehicles, intelligent smart sensors for medical care etc. For the distributed fault-tolerant hard real-time applications, very fast hardware and software system are necessary because of the constraints of the processing time. Due to the limitation in the processing speed of conventional systems and the low cost of processing units, parallel distributed processing is one of the best solutions to implement complex hard real-time application. Meanwhile in parallel distributed processing, the development of the applications is much more difficult than the sequential monolithic applications, because we have to consider the hardware architecture, the inter-processes synchronisation and communication and the granularities of the processes to exploit the parallelism and the system utilisation. In fact, with LINDA concept, a parallel program can be developed without consideration of parallel hardware, consequently the implementation of an application is easier [1]. However the original LINDA concept is not adequate for distributed hard real-time parallel processing, because inter-process communication (IPC) time is not deterministic and increases dramatically when the number of processors is important (more than 100). To overcome these problems, we have implemented successively two distributed hard real-time micro-kernels based on LINDA concept: hierarchical LINDA (H-LINDA) [2] and Micro-LINDA (M-LINDA) [3].

H-LINDA is implemented on workstations and particularly on the parallel processing development system (PPDS) of Texas Instruments [4], while M-LINDA is implemented on ADSP-2106X SHARC of Analog Devices [5]. H-LINDA and M-LINDA both meet hard real-time parallel processing constraints but they do not have the functionalities such as process and communication support migration facilitating fault-tolerant applications.

In this paper we concentrate only on the kernel facilities for implementing the different fault-tolerance approaches. Thus we have not discussed the real-time fault-tolerant scheduling problem, which the reader can find in [8] [9].

Our objective is to develop a distributed hard real-time parallel processing micro kernel supporting fault-tolerant application and corresponding exactly what we need: a simple, efficient, very compact and light distributed hard real-time micro-kernel adapted to reliable distributed hard real-time intelligent smart sensors.

The rest of the paper is organised as follows. Section II presents the background and LINDA concept. In section III, we present the key features of DREAM and its internal structure, allowing pseudo process and tuple

migration. Section IV describes a case study of P/B fault-tolerance approach on the TMS320C31 starter's kit platform. Section V concludes the paper and presents the ongoing work.

2. BACKGROUND AND CONCEPT

In general, five classes of operating systems may be considered:

- Non embedded applications, monolithic kernel, soft real-time, resource consuming, interprocess communication and synchronisation (mailbox, message passing, semaphore and mutex ...): WINDOWS/NT, UNIX, RT-LINUX ...
- Distributed micro kernel: AMOEBA, CHORUS, ...[6]
- Hard real-time embedded applications: VxWORKS, VRTX, pSOS ...
- Specific DSP oriented kernel: SPOX, VIRTUOS ...

Super small real-time kernel: OSEK/VDX [7]. OSEK is dedicated for automotive applications.

The main key features of these kernels use classical concepts such as: time sharing with preemption (Process and thread manager), message passing, mailbox, port (Interprocess communication manager) and semaphore, mutex (Process synchronisation manager). Furthermore, the kernel is resource consuming, not adapted to distributed fault-tolerant hard real-time smart sensors applications. Thus, we were interested by the LINDA approach for parallel programming and interprocess communication [1]. The LINDA concept is a communication mechanism and a parallel programming model, instead of communicating with message passing or shared memory models; interprocess communication and synchronisation are based on the global shared data called tuple space containing a set of non order of tuples.

When a process needs a message, it performs IN() operation by putting a tuple in the tuple space if the template of the tuple is matching (message is available) then the message is extracted and deleted from the tuple space. On the contrary, the process is suspended. The message is extracted by its contents thus the interprocess is not space couple (it is not necessary to know the port and process number). When a process sends a message, it puts a tuple in the tuple space by performing OUT() operation and it continues to run. The interprocess communication in LINDA is asynchronous, thus it is not time couple. The time and space decouple of LINDA concept ease parallel programming but as we state previously standard LINDA implementation does not meet distributed parallel hard-real time application. Furthermore, to ease fault-tolerant applications, process and communication support such as port migration are necessary.

3. Distributed REALtime Micro-kernel: DREAM

The main objective of our work is to implement a super-small distributed hard real-time micro-kernel moving towards fault-tolerant applications. The key features of DREAM are: pseudo process and tuple migration, and DSP hardware tuning.

DREAM allows two classes of processes: periodic and non periodic. Periodic process has the highest priority (10 is the highest priority level) and is executed periodically. It may be interrupted but not preemptible. Non periodic process has a priority level (0 to 9) and is interruptible and preemptible. Like most of real-time kernel, DREAM kernel has five main modules (figure 1):

- User interface
- Interruption manager
- IPC manager
- Process manager
- Scheduler.

In this paper, we focus on IPC structure allowing dynamic reconfiguration when trouble occurs.

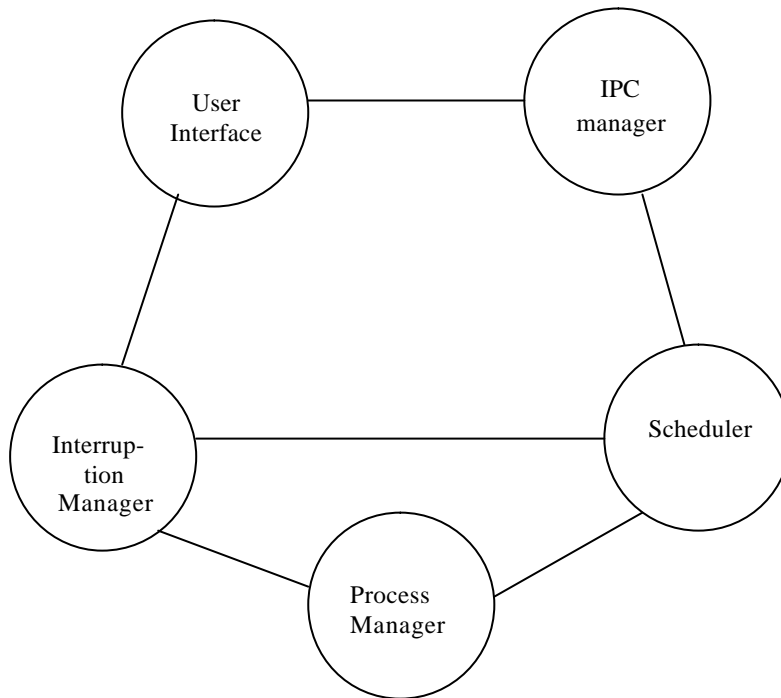


Figure 1: Main modules of DREAM

3.1. Process and tuple migration

Currently, intensive research is carried out to implement process migration over a set of heterogeneous computers MACH, AMOEBA, CHORUS ... [6], the issue is not evident, because of the multitude of processor instructions coding. To overcome this problem, we consider in our implementation the pseudo migration of process and of tuple in a network of homogeneous or heterogeneous system.

To minimise resource consuming and to increase efficiency of process execution time, like MLINDA, DREAM internal data are statically allocated and they may be configured to tune with an application: tuple space, process table etc. [3]. The drawback of this approach is a lack of flexibility, but this inconvenient is not an obstacle in the case of distributed specific embedded real-time applications such as intelligent smart sensors.

The migration of process means that the resource of the process (instruction, context, environment etc.) may be transferred over the network to another host to be executed. The migration of process implies the environment of the process to migrate, especially context and communication support: port. Currently, CHORUS enables only port migration and the process cannot migrate [10] [6]. In our implementation, pseudo migration is considered. To implement this concept, equivalent process is defined. A set of processes is equivalent if and only if they have the same functionality and I/O operations. In pseudo process migration, the resource of process is not sent from one node to another node. Pseudo process migration is implemented by activating the duplicated equivalent process in another node.

Two types of process migrations may be considered: process migration without context and process migration with context. A process migration without context means that the process may be started normally from the beginning with initial data (figure 2).

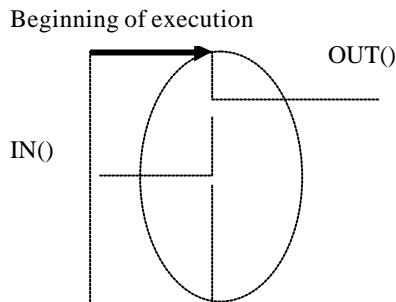


Figure 2 : Pseudo process migration without context

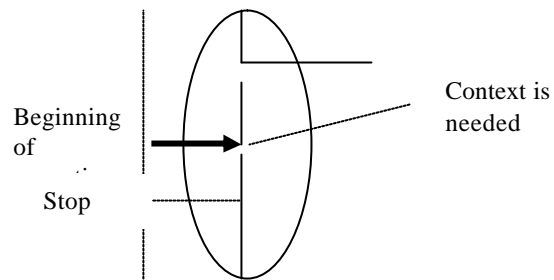


Figure 3 : Pseudo process migration with context

Process migration with context means that the process must be re-executed from the stop point (figure 3). This constraint may be overcome in the process coding. In practice, the execution of the process is not started exactly from the stop point, but it is started from the checkpoint where the process context is saved (rollback) [14, 15, 16]. For each IN() action at checkpoint, at least two nodes receive the same message (message is duplicated).

To allow pseudo process and tuple migration, the key capabilities are introduced. Key capabilities enable the kernel to change dynamically process input or output locality (local, shared and distributed ...) and other attributes without recompiling the application program. In the case of fault-tolerant application without context, the application is developed normally. The fault-tolerant (P/B) is specified in the configuration table.

The IPC in DREAM are based on: IN(Key, Message-Pointer) and OUT(Key, Message-Pointer).

Key: is a field of template and is a number which a message is attached, it may be considered as a logical key number.

KeyCapability: ([DirectOrIndirect], [MessageLocality], [TypeOfCommunication], [Timeout], [TypeOfBuffer], [NumberOfConsumers])

KeyCapability: [DirectOrIndirect] is a number corresponding to the physical key number if it is equal to the logical key number, the message is accessed directly. If not the message is accessed indirectly through the physical key number. During dynamic configuration, the physical key number may be renamed, thus the messages are redirected easily.

KeyCapability: MessageLocality defines the message locality (type of tuple space: Local, Shared or Distributed). It may be changed dynamically by the kernel during process migration to adapt to the new system configuration when a fault occurs. It will be illustrated by an example during pseudo process migration. In the case of distributed exchange, a group attribute may be defined to specify the number of processors that the messages must be sent.

KeyCapability: [TypeOfCommunication] defines if the interprocess communication is: unidirectional or bidirectional over the same key.

KeyCapability: [Timeout] for the secure communication the action of IN() or OUT() may be associated with a timeout. For example, if the time is elapsed after IN() or OUT() actions a decision is taken depending on the TimeoutDecision (reconfiguration, send again, do nothing etc.).

KeyCapability: [TimeoutDecision] defines the action of the kernel to take place when timeout occurs: do nothing, try again or read system configuration table to activate one or more duplicated processes and change the MessageLocality.

KeyCapability: [NumberOfConsumers] defines the number of processes consuming the messages.

KeyCapability: [TypeOfBuffer] defines the types of buffer: over-write or not etc.

3.2. Inter-process communication implementation: IN() and OUT()

Generally inter-process communications may be done through:

- a- Memory: local or shared memory
- b- I/O port or dedicated hardware: point to point direct connection (one hop) or point to point indirect connection (multihop), message must be routed through an interconnection network.

Inter-process communication through local or shared memory are easy to implement by using IN() and OUT() operations. The TypeOfBuffer and the MessageLocality define the type of message buffer and its locality: local or shared. The group of consumers processor is not required, because the tuple is attained by any process running in the cluster.

The distributed inter-process communication through I/O port is cost effective in term of time and resource consuming. For its implementation, we get experience from the TCP/IP and ISO communication protocol. The message contains source and destination address and group value allowing the router to find the optimal path. the global distributed communication is organised as following:

- Network layer: routing algorithm based on specific or general topology. The message format is:
[Source Address] [Destination Address] [Group] [Control] [data][crc]

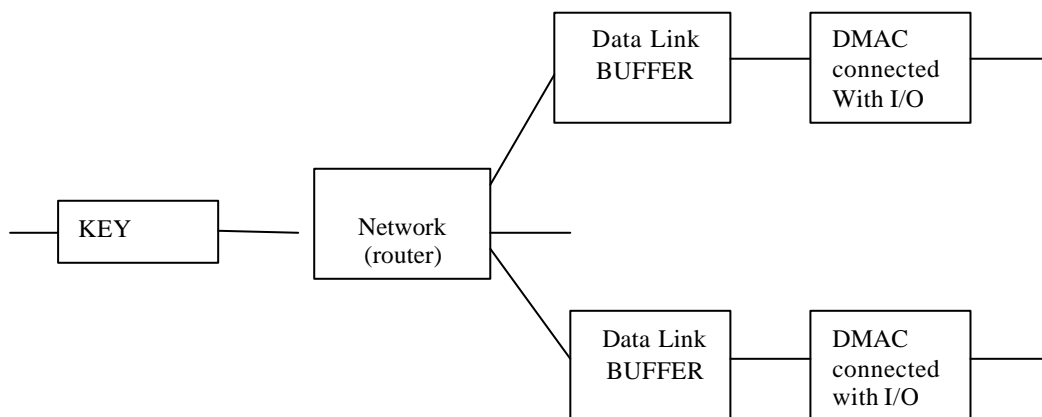
Group : defines the number of destination processors (multicast), it also allows to broadcast the message in the network.

To send a message (OUT() operation), the network layer determines the optimal path to send the message based on the destination address included in the group value and system hardware topology.

When a message is received, the network layer determines if the message attains the destination or not, if not the message will be forwarded. Once again the network layer uses group and destination address. Even the destination address is not corresponding to the node address if the group value has the node number the network layer will extract the message and delete this node number from the group value. The address destination is evaluated and the message is forwarded. This mechanism is introduced to avoid that the same message always circulates in the network.

- Data link layer: the basic functions of data link are:
 - Packing the packet into one message during reception,
 - Unpacking a message into several packets for emission,
 - Error management and recovery.
- Physical layer: the physical layer is tightly coupled with hardware support such as DMAC and serial port controller.

Each layer is assured by one or more kernel processes. The internal structure of the distributed inter-process communication through I/O port is illustrated by the figure 4.



4. Case study: TMS320C31 platform

Figure 4 : Structure of I/O point to point indirect connection

For the case : ial port add-on board are used to
illustrated P/B fault-tolerance approach [8].

Two set of processes are executed on two different nodes (figure 5 -a). In normal mode, P1 exchanges with P4 when P1 is down during the IN() or OUT() operations a timeout is occurred thus after reading the key capability: TimeoutDecision and system configuration, the kernel of node 2 decides to activate P'1 (P1 equivalent process) and

changes the MessageLocality of P4 and of P'1 from distributed to local. Consequently P4 will exchange with P'1 instead of P1 (P1 is considered as shutdown) (figure 5-b).

When P1 is recovered, it sends a message to the kernel. A check procedure is applied to assure that P1 is out of failure, thus the kernel will change the system configuration and the MessageLocality, deactivate P'1 and activate P1. Consequently, the system will recover all its functionality and performance (figure 5-a).

The manner we define the key capabilities eases the implementation of the pseudo process migration in a homogeneous or heterogeneous distributed multiprocessor system.

In the case of P/B approach without context, the implementation of an application is done as usual. By specifying the system configuration table, the application will be fault-tolerant.

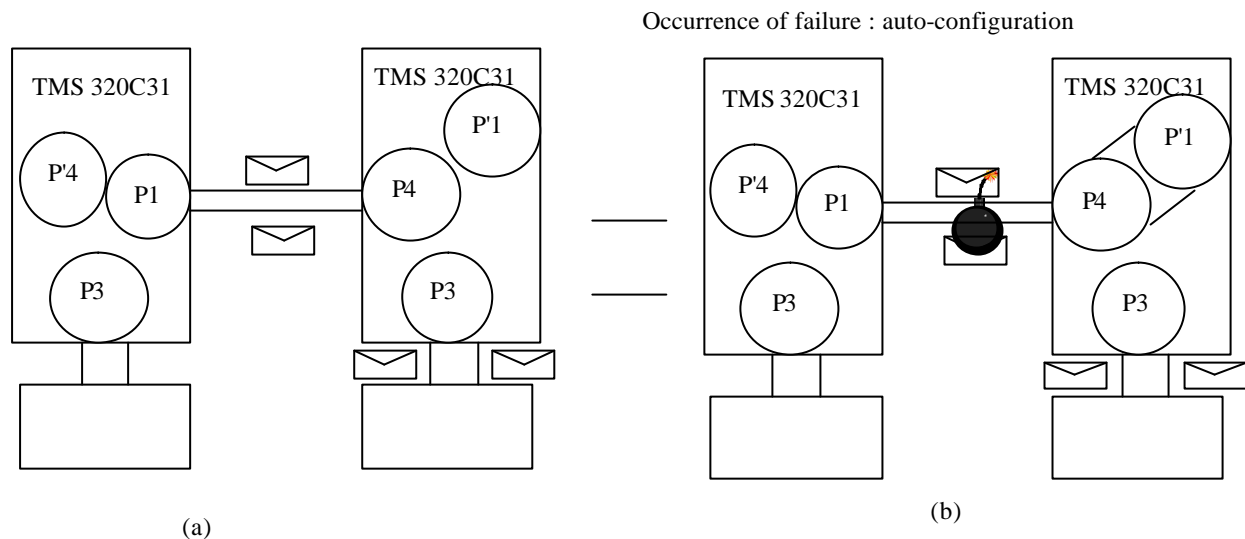


Figure 5 : Case study of P/B approach

The system configuration table is the collection of the signatures of all the system processes especially (out(A) \cup in(A)). It may be verified by I/O automaton formalism to check the system consistency and primary static deadlock [13, 12].

5. Conclusion and ongoing work

As we previously state, our work is focused on the fault-tolerant hard real-time kernel. We show that P/B application without context can be implemented easily. Thus DREAM is well adapted to distributed intelligent fault-tolerance smart sensors.

To assure the DREAM reliability and the consistency of the distributed fault-tolerant application, I/O automaton will be used. Furthermore, an embedded performance evaluation tool will be embedded with DREAM [12].

References

- [1] S. Ahuja, N. Carriero and D. Gelernter (1986). "Linda and Friends", Vol. 19, No. 8, Aug., pp. 26-34.
- [2] E. Yao, B. Jardin, Y.H. Park, A. Belloum and K.M. Hou (1993). "Real-time Multiprocess Kernel: Hierarchical LINDA", Proc. of ICSPAT'93, pp. 26-34.
- [3] Y. H. Park (1997). "Etude en vue de la réalisation d'un noyau temps réel multiprocesseur et l'environnement de développement intégré", Thèse de doctorat de troisième cycle de l'Université de Technologie de Compiègne.

- [4] Texas Instruments (1992) "TMS320C40 Parallel Processing Development System technical reference", Texas Instruments, Digital Signal Processing Products.
- [5] Analog Devices (1995), 'ADSP-2106X SHARC user's Manual, First Edition, Analog Devices Inc.
- [6] A. S. Tanenbaum, (1995). "Distributed Operating Systems", Prentice Hall, Englewood Cliffs, N.J. 07632.
- [7] <http://iiit.etec.uni-karlsruhe.de/pub/OSEK>.
- [8] S. Ghosh, R. Melhem and D. Mossé, (1997). "Fault-Tolerance Through Scheduling of Aperiodic Tasks in Hard Real-time Multiprocessor System", IEEE transactions on parallel and distributed systems, Vol 8, No 3, March.
- [9] Y. Oh and S. Son, (1991). "Multiprocessor support for real-time fault Tolerant Scheduling", Proc, IEEE 1991 Workshop Architectural Aspects of Real-time Systems', pp 76-80, San Antonio, Tex., Dec.
- [10] Chorus Systèmes (1991). "Chorus kernel v3 r4.0: Programmer's reference Manual" September.
- [11] Texas Instruments, (1997). "TMS320C3X Digital Signal Processor Starter's Kit", TMDS3200031, 12 June.
- [12] Y.H. Park and K.M. Hou, (1999). "Embedded performance analysis tool for a distributed hard real-time micro-kernel: H-LINDA", OPODIS'99, Hanoi, 21-23 October.
- [13] N. A. Lynch and M. R. Tuttle, (1988). "An Introduction to Input/Output Automata", MIT Technical Memo MIT/LCS/TM-373.
- [14] A. Ziv and J. Bruck, (1998). "Analysis of Checkpointing Schemes with Task Duplication", IEEE transactions on computers, vol. 47, No 2, February.
- [15] A. Zhen and K.G. Shin, (1998). "Faulty-tolerance Real-time Communication in Distributed Computing Systems", IEEE Transaction on parallel and distributed systems, Vol 9, No 5, May.
- [16] M. Choy and A. K. Singh, (1996). "Localizing Failures in Distributed Synchronisation", IEEE Transactions on Parallel and Distributed Systems, Vol 7, No 7, July.