Distributed Real-time and Fault-tolerance oriented Micro-kernel : DREAM

Authors : Christophe de Vaulx and Kun Mean Hou, Laboratoire LIMOS FRE 2239 CNRS, ISIMA, UBP Clermont-Ferrand II, France.

Email : devaulx@isima.fr, kun-mean.hou@isima.fr

Abstract

DREAM is a super small distributed hard real-time micro-kernel based on LINDA concept. The key features of DREAM are: efficient and simple process intercommunication, pseudo process and communication support migration and ease fault-tolerant applications implementation. In this paper, we present the basic concepts of DREAM such as process scheduling, intercommunication and 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, LINDA, Pseudo process and tuple migration.

1. Introduction

Nowadays, many applications like aircraft navigation systems, process controls, intelligent vehicles or intelligent smart sensors used embedded systems. The complexity of these applications become more complex, that's why the embedded systems must be increasingly fast, powerful and reliable (it should not break down if one of their process stop for example) while being small and resources sparing. To manage such applications we cannot used any more simple monitors, most of the time, we are even obliged to develop true multi-task, real-time and distributed operating systems. Moreover these systems must be open towards the external world, i.e. which it must support TCP/IP communication protocol for instance.

The main objective of our work is to implement a simple and efficient super small Distributed hard REal-time Micro-kernel named DREAM based on LINDA concept [1] (With LINDA concept, a distributed parallel program can be developed without consideration of parallel hardware, consequently the implementation of such an application is easier), allowing easy implementation of reliable and distributed real-time intelligent smart sensor applications. The key features of DREAM are: efficient and simple process intercommunication, pseudo process and communication support migration. These features ease distributed fault-tolerant applications implementation by using 'Primary Backup'.

In this paper, we present the basic concepts of DREAM such as process scheduling, intercommunication and migration. A case study of fault-tolerance (P/B approach) is implemented on TMS320C31 starter’s kits.

2. Objectives and motivation of DREAM development

Currently, many operating systems are on sale on the market. Although these systems are very different each other, we can classify them in five distinct categories:
- systems used for a general use and on servers such as LINUX, WINDOWS 9x/NT or UNIX,
- systems used for a general or modular use such as CHORUS, MACH or AMOEBA,
- hard real-time and embedded systems such as VRTX, or VxWORK … ,
- directed DSP systems such as SPOX, Virtuoso, Parallel C or RXTC,
- finally the real-time micro-kernel such as OSEK/VDX or DREAM.

Most of these systems :
- consumes a lot of system resources,
- is too much complex compared to the needs,
- lacks facility for the parallel programming.
Our objective with DREAM is to provide a kernel without the previous drawbacks. The main characteristics of DREAM are that DREAM:

- is hard real-time,
- is embedded,
- is based on the concept of LINDA (DREAM is a reduction of LINDA),
- is very compact and consumes very few resources (VLSI),
- is adapted to parallel and distributed applications,
- allows to implement friendly fault tolerant applications.

3. Distributed real-time micro-kernel: DREAM

As the majority of the real-time kernel DREAM, is composed of five modules (figure1):

- the process manager
- the scheduler
- the communications manager
- the exceptions manager
- the user interface.

![Figure 1: Main modules of DREAM](image)

3.1. The process manager

To implement this manager we based ourselves on work which was completed for MLINDA [2, 3]. Two types of processes can thus be used in DREAM, periodic and a-periodic processes:

- The periodic processes are critical and fast tasks. These processes are run periodically in a user given time interval. The user can also choose the delay when one of these tasks has to be executed the first time by setting what we call offset. These tasks are interruptible but not preemptible in order to ensure the time constraint in case of context switch during the execution. Finally the priority of these tasks is 10, it is the highest priority of the kernel.

- The a-periodic processes are generally less critical tasks than the periodic ones. The user initialised their priority 0 to 9 before the kernel compilation. These processes are interruptible and preemptible i.e. we can suspend their execution to run another task for example. They have to have their own stack, in order to save their context when they are suspended.

The main processes primitive of DREAM are summarised in the table below:

<table>
<thead>
<tr>
<th>Name</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>create_tsk()</td>
<td>allows to create a task</td>
</tr>
<tr>
<td>delete_tsk()</td>
<td>allows to delete a task</td>
</tr>
<tr>
<td>suspend_tsk()</td>
<td>allows to suspend a task</td>
</tr>
<tr>
<td>resume_tsk()</td>
<td>allows to resume a task</td>
</tr>
</tbody>
</table>

3.2. The process scheduler

The scheduler of DREAM is a pre-emptive scheduler based on task priorities. After each timer interruption, we run the tasks by decreasing set of priorities. In case of equality, the first created task is run first.
3.3. The communication manager

Generally inter-process communications may be done by using message passing through: local or shared memory and distributed network.

To set up DREAM communications manager, we improved the concept of tuple that was introduced into MLINDA. A tuple can be regarded as a block of shared data (it is saved in the tuple space). Inter-process communication is realised by using two primitives: in() and out(). A unique number, the key, identifies each tuple.

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. Furthermore, all the specific drivers or communication protocol (TCP/IP stack) are attached to the kernel through the tuple. The drivers or communication stack are encapsulated in one or more DREAM processes. Classical critical resources are managed by using semaphore or mutex. Thus the connection with a specific device or external world are done easily through in() or out() primitive.

To achieve the objectives, the capabilities of the key are defined in DREAM.

These key capabilities are : ([DirectOrIndirect], [MessageLocality], [Timeout], [typeOfCommunication], [TypeOfBuffer], [NumberOfConsumers])

- [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.
- [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.
- [TypeOfCommunication] defines if the interprocess communication is: unidirectional or bidirectional over the same key. In the case of distributed tuple, a protocol attribute is used to specify the type of the communication protocol: DREAM internal protocol or TCP/IP …
- [Timeout] for the secure communication the action of in() or out() may be associated with a timeout (duration).
- [TimeoutDecision] defines the kernel action to take place when timeout occurs.
- [NumberOfConsumers] defines the number of processes consuming the message.
- [TypeOfBuffer] defines buffer type: over-write or not etc.
3.4. Process and tuple migration

Generally the process migration means that the process resource (instruction, context, environment etc.) may be transferred over the network to another host to be executed. The process migration 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 [4, 5]. In DREAM, only 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. Only two types of process migrations may be considered: process migration without and with context:

- A process migration without context means that the process may be started normally from the beginning with initial data (figure 5).

- Process migration with context means that the process must be resumed from the stop point (figure 6). 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) [6, 7, 8]. For each in() action at checkpoint, at least two nodes receive the same message (message duplication).
4. Case study: TMS320C31 plat-form

For the case study, two TMS320C31 DSP starter’s kits [9] with a memory and serial port add-on board are used to illustrated P/B fault-tolerance approach [10]. Two set of processes are executed on two different nodes (figure 7-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 7-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 7-a).

5. Conclusion and ongoing work

In this paper, we presented the DREAM kernel supporting embedded TCP/IP stack of Nexgen Software Inc® and a fault-tolerant application. DREAM is well adapted to distributed embedded fault-tolerant smart sensors. The next step of our work are focused on two directions: the DREAM development environment such as distributed performance evaluation tool and the reliability and the consistency of the distributed fault-tolerant application by using I/O automaton model [11].

References


© copyright of NexGEn-Software, Inc.