MDSPLUS REAL-TIME DATA ACCESS IN RTAI

A. Barbalace[1)], A. Luchetta[1)], G. Manduchi[1)], C. Taliercio[1)], T. Fredian[2)], J. Stillerman[2)]
[1)]Consorzio RFX, Associazione Euratom-ENEA per ricerche sulla fusione
Corso Stati Uniti, 4, 36127 Padova ITALY
[2)]Massachusetts Institute of Technology, 175 Albany Street,
Cambridge, MA 02139, United States

Abstract

The MDSplus package is widely used in nuclear Fusion research for data acquisition and management. Recent extensions of the system provide useful features for real-time applications, such as the possibility of locking selected data items in memory and real-time notification. The real-time extensions of MDSplus have been implemented as a set of C++ classes and can be easily ported to any target architecture by developing a few adapter classes. The real-time data access layer of MDSplus is currently available for Windows, Linux, VxWorks and RTAI. In particular, the RTAI platform is very promising in this context because it allows the co-existence of non-real-time and real-time tasks. It is hence possible to devise an architecture where real-time functionality is handled by a few selected tasks using the real-time data access layer of MDSplus, whereas background, non-real-time activity is carried out by "traditional" Linux tasks. This organization may be of interest for the next generation of fusion devices with long-duration discharges, during which the system has to provide feedback control in real time and to sustain continuous data acquisition and storage.

INTRODUCTION

The MDSplus system has been successfully adopted in many fusion experiments for data acquisition, storage and access[1]. Until 2006, however, MDSplus was not fit for the new generation of fusion experiments in which the plasma discharge may last minutes or hours. In this case data cannot be stored during the discharge in the local memory of transient recorders and then transferred into the pulse database after the discharge. Rather, data need to be continuously acquired and stored in the database so that it can be used during the discharge itself. The most recent release of MDSplus allows data samples to be appended to stored signals in the database[2]. This new feature is useful for handling event driven data acquisition, where bursts of data are acquired in correspondence to events occurring during the discharge. While the new features of MDSplus provide the hints to achieve continuous data acquisition during the plasma discharge, they cannot guarantee the real-time responsiveness required to use acquired data in the active control of the experiment. For this reason, work is in progress for introducing real-time functionality in MDSplus [3]. Feedback control is in fact becoming a standard procedure in fusion experiments, but in most current installations data acquisition is separated from feedback control, due to the hard real-time requirements of the latter.

SYSTEM ARCHITECTURE

The real-time layer of MDSplus is composed of two main components: support for data caching in memory and real-time notification of data update. They represent the basic ingredients required to build feedback control systems, where data are acquired from the sensors, a control algorithm is executed and the results are sent to the actuators.

The memory data cache component does not replace any existing component of MDSplus for data acquisition, and is built on top of the MDSplus data access layer. It allows selecting a subset of data items for which a copy is maintained by the system in memory in order to provide deterministic access time. It is worth noting that such mechanism cannot be compared to mapping disk files into memory, since it permits to define precisely the data items to be kept in memory. This feature is important because it provides an optimized usage of memory space since usually only a very reduced subset of data items is handled in control, in respect of the whole set of data involved in acquisition. The memory cache component relies on the underlying MDSplus functionality for all the other data–related operations. In particular, the combined usage of memory caches and the remote data management provided by MDSplus allows the construction of distributed control systems, where the database reside remotely, and every control node handles a local copy of those data that are involved in control.

A notification mechanism for data update is useful in feedback control systems where the system has to react to the occurrence of a new event, such as the availability of a new input sample from the sensors, which triggers the computation of some sort of control algorithm and the generation of a new set of outputs to the actuators. This functionality is achieved using a publish-subscribe pattern centered on data. An actor can express its interest in being notified when a given data item is updated, by passing the

address of a callback routine. Afterwards, the system will call such routine in a separate thread each time that data item is updated.

Both memory caching and notification are supported also in a distributed environment. Distributed memory data caches require exchanging information when a given data item is cached in different machines, in order to maintain data consistency. MDSplus defines two different approaches for handling cache coherence: *push* mode and *pull* mode. In pull mode, when a data item is updated in one cache, all the other data caches holding the same data item are notified that this cache has become the current owner of that data item. In this case, when the data item is read in another cache, it is first requested from the current owner. In push mode, the current owner, i.e. the cache where data have been written, sends an updated version of the data item to all the caches sharing it, which hold therefore an up-to-date version of the data item. The push mechanism is used also to achieve remote notification, i.e. activating a callback routine in response to updating that data item in a different machine. Depending on the way data are accessed, either push or pull mechanisms may minimize the number of exchanged messages. For example, if a data item is updated frequently on one machine and read only rarely on another machine, the pull option is preferred. If however the data item has to be read in one cache every time it is written in another cache, a typical situation in feedback control, the push mechanism is better.

Figure 1 shows a typical configuration for the real-time layer of MDSplus in distributed control. Two networks are involved: an off-line network is used by the underlying MDSplus system to read data from the pulse files, hosted in a separate data server. Data access via the off-line network is carried out during non real-time operation, normally before the discharge, in order to copy data into the memory cache, and is achieved by means of the TCP/IP-based protocol used by MDSplus for remote data access. During real-time operation, most of the data access is performed on the local cache. To maintain the consistency of data items shared by two or more computers, the communication is achieved via messages exchanged along the online network, usually an isolated Ethernet segment, or reflective memory. The off-line network may also be concurrently used by low priority, non real-time tasks for data logging.

ACHIEVING MULTIPLATFORM SUPPORT

The MDSplus package is currently available in a variety of platforms including Linux, Windows and VxWorks.

The entire system, with the exception of the real-time layer, has been written in C, and therefore multiplatform support is provided in the traditional way of using the #ifdef conditional preprocessor directive. The real-time layer is instead written in C++, and in this case multiplatform support has been achieved by encapsulating system–dependent code into a subset of C++ classes. Porting the system to a new platform requires therefore re-implementing only those classes. In particular, the system-specific classes refer to shared memory allocation, thread activation, locking and event notification.

Real-time communication, required to maintain consistency in distributed data caches, relies on the implementation of a set of generic classes for communication. Currently, a UDP-based implementation over Ethernet is available, but other communication media can be integrated, such as ATM and reflective memories.

The real-time layer of MDSplus has been ported to Linux, Windows and VxWorks. Only the latter system provides real-time responsiveness, and in this case real-time notification can be carried out by letting a high priority task execute the callback routines in response to data update. Such determinism in response time cannot be achieved in Windows or Linux. However, quasi real-time performance can be obtained using the Linux kernel 2.6 because it is possible to associate a fixed priority with a subset of processes and the kernel has been made pre-emptive by accurately defining the uninterruptible segments in the kernel and protecting them with spin locks, rather than disabling interrupts. For this reason, the performance of the real-time MDSplus layer can be satisfactory for those systems for which an occasional delay in the response time is tolerable.

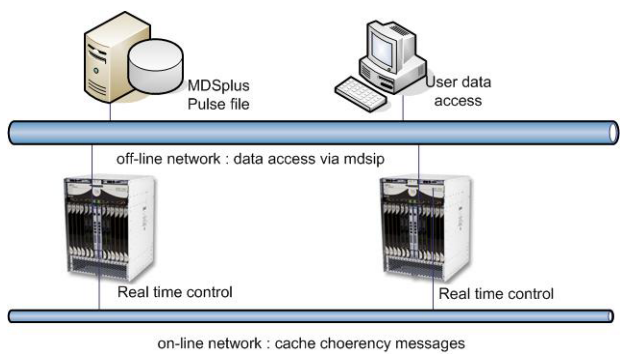

Figure 1: A typical configuration for distributed control.

INTEGRATION IN RTAI

In order to port MDSPlus to a free, open source real-time operating system, we have considered RTAI, a real-time extension of the Linux kernel[4]. There is a substantial difference in software organization between VxWorks and RTAI and, more in general, the real-time extensions of Linux. In fact, in VxWorks all code runs natively in privileged mode, thus letting user program have full control of the hardware resources, providing at the same time an environment which is quite similar to that available in user mode on Unix. In this case the porting of the MDSplus, and in particular of the real-time data

access layer, has been almost straightforward. This required only slight changes in the system-specific classes, such as a straight declaration of a memory buffer instead of using system calls for handling shared memory, since memory is natively shared by all tasks in VxWorks. On Linux, instead, we had to decide which components of the real-time data access layer had to be moved to kernel space. Moving to kernel space, however, would have required a deep re-arrangement of the code to provide the required interface to the GLIBC support, not directly available in kernel mode. This would have required re-writing large parts of the code, something which is against the general approach taken in MDSplus, i.e. keeping the required system-specific differences to a minimum in order to increase the maintenance of the whole code. A solution to this problem is provided by the fact that user processes can be made real-time in RTAI. In this case the only changes required to take advantage of the real-time capability of RTAI, with respect to the Linux version, is the usage of a RTAI-specific real-time semaphore in the data update notification.

Using this approach, a typical configuration for feedback control, where a control computation is performed on a set of inputs to produce the controller output to be sent to the actuators, logging at the same time the output references to a disk-based pulse file, would be the following:

1) The input device generates an interrupt when a new data sample is available;

2) The processor calls the associated Interrupt Service Routine (ISR) via the general dispatching mechanism of RTAI;

3) The ISR code, running in kernel mode, reads the data sample from the registers of the input device, copying it in a memory segment shared by a user task, which gets then awaked by a RTAI real-time semaphore;

4) When the ISR exits, the RTAI scheduler forces a direct context switch to the user process waiting for the data, bypassing the Linux scheduler;

5) The user process makes the required control computation and writes the results via the MDSplus real-time data access layer;

6) When data have been written, a non real-time task is activated by MDSplus for data logging, and another real-time user process is awakened by the system which will send data to the output driver (e.g. via a write in mmap area).

Observe that the whole process activation chain is carried out in real-time by RTAI bypassing the Linux scheduling mechanisms, while retaining all the MDSplus code in user mode. At the same time, a Linux process will carry out data logging on permanent storage.

The possibility of letting user processes become real-time processes is offered also by Xenomai [5], another real-time extension of Linux. We carried out a performance comparison between RTAI and Xenomai [6], and we observed that Xenomai has a slightly poorer performance, but it is better engineered than RTAI. For both RTAI and Xenomai, the porting of the non real-time part of MDSplus is straightforward because both systems are built on top of Linux, and therefore no changes are required in the Linux version of MDSplus. Although we are currently porting MDSplus to RTAI, we plan to do a similar job for Xenomai in the future, since porting the real-time layer of MDSplus to either RTAI or Xenomai requires only a slight change in the system classes in order to use the system specific semaphores and processes.

When considering distributed systems involving real-time communication to achieve data cache coherence, a promising approach is the usage of RTNet [7], an open source hard real-time network protocol stack for RTAI and Xenomai which makes use of standard Ethernet hardware. The integration of RTNet into the framework requires an implementation of the system specific classes for communication, and is currently under development.

In conclusion, the availability of real-time user processes represents a good tradeoff between performance and software maintenance, and the real-time extension of MDSplus over RTAI or Xenomai represents a valid candidate for the development of real-time control systems in the next generation of long lasting fusion experiments.

ACKNOWLEDGEMENT

This work was supported by the European Communities under the contract of Association between EURATOM/ENEA

REFERENCES

[1] T . Fredian , J . Stillerman ”MDSplus Current developments and future directions, Fusion Engineering and Design”, vol 60, 2002, pp 229-233

[2] T. Fredian, J. Stillerman, G. Manduchi, “MDSplus extensions for long pulse experiments”, to appear in Fusion Engineering and Design.

[3] G.Manduchi, A. Luchetta, C. Taliercio, T. Fredian, J. Stillerman “Real Time Data Access Layer for MDSplus”, to appear in Fusion Engineering and Design.

[4] RTAI Home page, [Online]. http://www.rtai.org

[5] Xenomai Home Page, [Online]. http://www.Xenomai.org.

[6] A. Barbalace, A. Luchetta, G. Manduchi, M. Moro, A. Soppelsa and C. Taliercio “Performance Comparison of VxWorks, Linux, RTAI and Xenomai in a Hard Real-time Application”, to appear in IEEE Transactions on Nuclear Sciences.

[7] RTNet Home Page [Online] http://www.rts.uni-hannover.de/rtnet