Using Linux PCs in DAQ applications

G.Unel[b], G. Ambrosini[b], HP.Beck[a], S.Cetin[d], T.Conka[d], G. Crone[e], A. Fernandes[b], D.Francis[b], M.Joos[b], G.Lehmann[a,b], J.Lopez[b], A.Mailov[d], L.Mapelli[b], G.Mornacchi[b], M.Niculescu[b,c], J.Petersen[b], L.Tremblet[b], S.Veneziano[b], T.Wildish[b], Y.Yasu[f].

[a]Laboratory for High Energy Physics, University of Bern, Switzerland.
[b]CERN, Geneva, Switzerland.
[c]Institute of Atomic Physics, Bucharest, Romania.
[d]Bogazici University, Istanbul, Turkey.
[e]University College London, London, England.
[f]KEK, Tsukuba, Japan.

Abstract

The ATLAS Data Acquisition / Event Filter Prototype "-1" (DAQ/EF-1) project provides the opportunity to explore the use of commodity hardware (PCs) and Open Source Software (Linux) in DAQ applications.

In DAQ/EF-1 there is an element called the LDAQ which is responsible for providing local run-control, error-handling and reporting for a number of Read-Out modules in Front End crates. This element is also responsible for providing event data for monitoring and for the interface with the global control and monitoring system (Back-End).

We present the results of an evaluation of the Linux operating system made in the context of DAQ/EF-1 where there are no strong real-time requirements.

We also report on our experience in implementing the LDAQ on a VMEbus based PC (the VMIVME-7587) and a desktop PC linked to VMEbus with a Bit3 interface both running Linux. We then present the problems encountered during the integration with VMEbus, the status of the LDAQ implementation and draw some conclusions on the use of Linux in DAQ applications.

I. INTRODUCTION TO DAQ/EF –1 PROJECT

The goal of the ATLAS DAQ/EF Prototype -1 (DAQ/EF -1) project is to produce a prototype system representing a "full vertical slice" of a DAQ suitable for evaluating candidate technologies and architectures for the final ATLAS DAQ system. It has been organised into four major subsystems: DataFlow, Back-End, Event Filter and Detector Interface. [1] The DataFlow is the hardware and software elements responsible for receiving, buffering, distributing event data, providing event data for monitoring; and storing event data from the detector. These functions are provided by the Front-End DAQ for the collection, buffering and forwarding of data fragments from the detector; the Event Builder for the merging of event fragments into full events; the Sub-Farm DAQ for the sending to and retrieving of events from the Event Filter and for sending events to mass storage. These subsystems are made from one or multiple copies of their basic crate units: Readout Crate (ROC) for Front-End DAQ, Dataflow manager crate (DFM) for Event Builder and Subfarm crate (SFC) for Sub-Farm DAQ. The element which is common to all the crates in all subsystems is the Local DAQ (LDAQ) which is responsible from local control and monitoring together with interfacing with the Back-End. The current implementation of this logical model is based on VMEbus single board computers (SBC). A typical example is the ROC where different elements such as LDAQ, ROB (ReadOutBuffer), TRG (trigger interface), EBIF (event builder interface) etc, are implemented as applications on PowerPC based SBCs running a conventional real time operating system, LynxOS. In the current prototype these SBCs are either RIO2 modules from CES [2] or MVME 2x00 modules from Motorola [3].

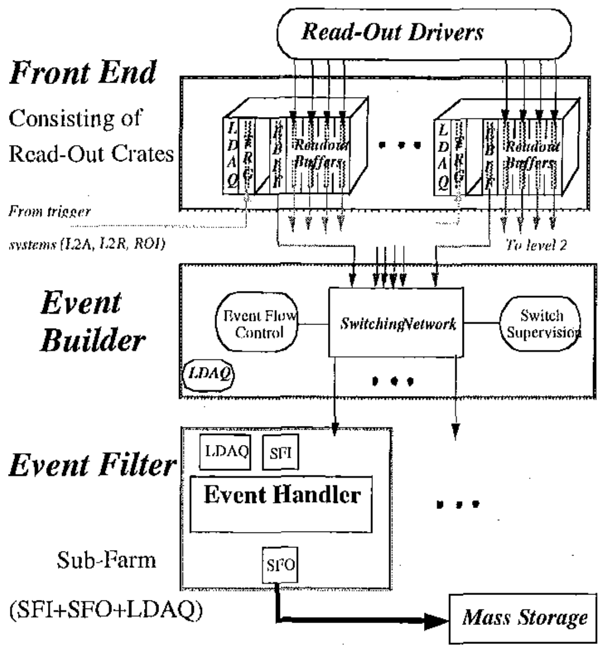

Figure 1: Global scheme of DAQ/EF –1 project.

II. LDAQ AND LINUX

The general use of today's commodity computer hardware, Intel compatible PCs, for HEP applications such as off-line farms and even on the on-line typically as processor farms in some experiments, raises the question of having PC based hardware in the DAQ crates as well. PC market opens doors to a much bigger number of hardware choices as well as manufacturers.

The choice of a suitable Operating System (OS) for PC compatible hardware is naturally dependent on the expected functionality. Since the software tools, like compilers and utilities (e.g. debuggers) which make a usable OS out of a simple kernel are, often, Open Source Software / GNU products, and they are considered as the "commodities" of the Internet, one may also consider the possibility of using a commodity Open Source OS, Linux[4].

Deeper understanding and control of the operating system, due to its openness, can be used in many different ways in a DAQ project. For the areas which require direct access to the hardware, where one tries to get the maximum performance, the knowledge of the internal details of the operating system is a possible source of improvements. In the local control of the hardware, a trustworthy and robust operating system would ease the development of the applications by letting the OS handle difficult tasks (e.g. scheduling, multi-threading). At the global control level, the key demands become portability, scalability and interoperability.

A. LDAQ and its prerequisites

The natural candidate to try out a PC based implementation is the LDAQ, since in the modular Data Flow system, it provides all the high level DAQ functions which are not related to the main flow of the event data. It is also the interface point to the Back-End DAQ system (BE) for the run control, configuration databases, Message Reporting System and the Information Service [5]. This interfacing task brings some hardware and software functionality requirements for the LDAQ application. In DAQ -1, The primary hardware requirement is the support for VMEbus, via which control and monitoring the modules in any Dataflow crate is done. For example, assuming a few Kbytes of data at a rate of 100 Hz, the requirement for VMEbus data transfer for the LDAQ (including the data sampling for monitoring task) is of the order of few Mbytes/sec. The basic software functionality requirement is the support for the typical workstation class software (multitasking, X11 development, scripting languages etc) since LDAQ is a control oriented application.

B. Linux and Its highlights

Linux is an independent implementation of the POSIX operating system specification, with SYS-V and BSD extensions. [4] Linux is freely distributable under the GNU Public License [6]. It runs on almost all the modern CPUs including Intel-PC compatibles, Alpha, SUN sparc, Power-PC, Motorola 68K and MIPS. It has most of the features one would expect in a modern fully-fledged Unix, including the possibility to add real-time extensions to the standard operating system.

Its attractive features include POSIX compliance, modular custom kernel, good performance, RT extensions, availability of latest development tools, continuous development to support new hardware, ease of administration and integration with other systems, full availability of the source code, free availability of the full system, large support over the Internet and the possibility of having commercial support.

The POSIX standards define, among other things, an interface to the OS, interprocess communications (1.b, Real-Time Extensions) and multitasking system calls (1.c, Thread Extensions). These standards are important to any project for portability and compatibility issues. Linux is POSIX conformant in the sense that it supports Posix Threads, Semaphores, Timers and Alarms, Shared Memory, Message Queues (implemented on top of systemV functions), Asynchronous I/O and Real-Time Signals. The last two are only implemented in the V2.1 of the gnu C library.

Linux is supported by CERN-IT division (AFS, CERNlib, ASIS, SUE, HEPIX etc) and it is being used in the off-line as a working environment and/or as an analysis platform in ATLAS and other experiments at CERN (e.g. wa95, na48, na59).

Linux started to be used in the LDAQ context as a development platform for some applications such as the stand alone GUI and the GUI-LDAQ communications. The source code developed under Linux were used in the current target platform (LynxOS) easily due to their common Unix properties and the GNU tools used in both systems. The main reasons for using Linux as a development platform were the wide availability of Linux based PC stations, the existence of many development and debugging tools, latest libraries, utilities, compilers which speed up the initial creation phase, and the availability of fast, even SMP (Symmetrical MultiProcessing), systems profiting from price-performance ratio as COTS (Commodity Of The Shelf) solutions.

C. Real Time Extensions

Real Time extensions to Linux make it more deterministic. This is accomplished either by the insertion of a Real-Time kernel layer between Linux kernel and hardware interrupts (RTLinux) [7] or by changing the current scheduling policy inside the kernel to a tighter one (KURT Linux) [8].

In the DAQ -1 project, the LDAQ does not have any hard real time requirements. On the other hand, other modules have tight timing constraints imposed by high-rate data flow. Applications that deal with VMEbus are using hardware access libraries to bypass the kernel drivers for performance reasons; thus real time properties of LynxOS are not really used. This approach is acceptable since they are single tasked, therefore there is no resource sharing within a VMEbus module. Since the standard Linux has been found adequate for LDAQ application, it was decided to use it as such, keeping in mind the possibility to use RT extensions in the future.

III. Compatibility and Performance Tests

Before considering a Linux implementation of the LDAQ application a series of tests were done to understand the performance of Linux and to see its compatibility with the existing software. As the test environment, two Intel based computers were used: A "standard" desktop PC and an embedded VMEbus computer from VMIC, the VMIVME7587 [9]. The idea behind the study of the performance of the desktop PC is to understand the possibility of using it as a VMEbus CPU via a PCI-VME interface, for example Bit3 617 [10]. The following table is a feature summary of these computers:

Table 1
Testbed Systems.

	Embedded PC	Desktop PC
CPU	Pentium 100Mhz	PentiumPro200Mhz
Memory	32Mb+512K L2	64 Mb+256K L2
SCSI Interface	Adaptec Aic7xxx	Adaptec Aic7xxx
Ethernet Interface	Onboard SMC9194	PCI Intel Eepro
VMEbus Interface	Tundra Universe	Bit3 617
Extension slots	1 PMC	4 PCI

A. *Benchmarks*

A small set of benchmarks have been performed, to compare Linux 2.0 with the latest 2.1 series and thus to have an idea about the way Linux is evolving. The first of these tests basically estimates the raw performance of the CPU (Dhrystones), whereas the rest are some specific operating system call measurements: system V semaphores lock-unlock timing, the context switch time and the interrupt latency[11]. In table 2, the results from LynxOS and Linux on the same hardware (VMIC) together with some measurements with the Desktop PC are presented.

Table 2
Comparison Tests

Platform / OS	Dhrystone	Lock Unlock	Context Switch
VMIC / LynxOS2.4	133000	12.8 μs	N/a
VMIC / Linux2.0	160000	12.5 μs	16.1 μs
VMIC / Linux2.0*	170000	12.3 μs	14.8 μs
VMIC / Linux2.1	160000	10.6 μs	14.4 μs
Desktop / Linux2.0	400000	9.5 μs	6.0 μs
Desktop / Linux2.0*	446000	9.2 μs	5.6 μs
Desktop / Linux2.1	400000	6.8 μs	4.2 μs

*With Pentium optimizations.

In these tests the context switch time was measured to be half of the difference between the semaphore lock/unlock time between two processes and twice of the semaphore lock/unlock time in a single process. From these results it is possible to see that the addition of new features to Linux is not slowing it down, on the contrary V2.1 is performing better than V2.0; the effect of the C compiler on performance is obvious from the fluctuation in the dhrystones on the same SBC. In this context, for modern Intel systems, the Pentium optimized compiler is recommended for critical applications.

To measure the interrupt latency of the driver, a function generator connected to a VMDIS 8004 VMEbus display module have been used to generate interrupts in a continuous manner. The time between two adjacent interrupt acknowledge signals at the maximum frequency has been measured and found to be around 7.5 microseconds being the interrupt latency of the hardware and the kernel.

To fully understand a system, one needs a global test which utilizes most of the system resources relevant to the final application (all ideally) and which pushes the software/hardware to its limits. The software package for LDAQ communication was a good candidate for such a global test program since it basically uses the IO subsystem, the CPU, the Posix threads and signals, systemV semaphores, shared memory access and message queues. In table 3, the measured number of exchanged messages per second with different number of senders/receivers communicating via a single dispatcher, are given for the same VMIC board running two different operating systems. These measurements show that, on the same hardware Linux and LynxOS have a similar overall performance.

Table 3
Messages/sec under Linux and LynxOS

Senders/Receivers	LynxOS 2.4	Linux 2.1
S: 1 R: 1	2400	2800
S: 1 R: 2	1330	1515
S: 2 R: 2	725	806

B. *VMIC VMEbus Driver Functionality and Performance Issues*

On the VMIC card, the PCI/VMEbus interfacing is provided by the Universe Tundra chip [12] as on the MVME 2x00 cards. The existing driver for Linux 2.0 [13] supports single cycles, single and chained mode DMA transfers and handling of VMEbus interrupts. To understand it's performance, a series of data transfer tests have been done. A VMEbus memory module (MMI6390, [14]) capable of A32-D64 MBLT transfers has been used in the benchmarks. The data transfer timing has been measured on the VMEbus (using a VMETRO VMEbus analyser [15]) as well as in the kernel space and in the user space by the means of software methods. The summary of the DMA measurements performed is outlined in the following table:

Table 4
Time in microseconds to transfer 4000 bytes over VMEbus

DMA mode	VMEbus	Kernel Space	User space	Improved Version
A32-D32 Read	224μs	343μs	438μs	300μs
A32-D64 Read	113μs	222μs	309μs	187μs
A32-D32 Write	194μs	303μs	380μs	255μs
A32-D64 Write	104μs	220μs	296μs	172μs

The bottleneck for the driver's performance was the transfer of the data from/to kernel space to/from user space, which involves memory copy functions. To overcome this problem, some contiguous space was reserved in the memory and from the driver to the user space, instead of the data itself, only the address of it was transferred. The results of this exercise are presented in the rightmost column of table 4. The overhead for DMA access has been measured to be around 70 microseconds for all access modes.

C. Bit3 VMEbus Driver Functionality and Performance Issues

The SBS bit3 617 device [8] is made out of two cards, a PCI module for the desktop computer and a VMEbus module for the crate. It does not support MBLT transfers. The functionality of the existing Linux driver [11] has been successfully tested during our studies. The hardware itself has a higher data transfer rate on the VMEbus (28 Mbytes/s compared to 20 Mbytes/s for the Tundra Universe in D32 BLT) but also a higher DMA Latency (120 microseconds). Given the familiarity with the Tundra Universe chip through the MVME modules, it was decided to have the first LDAQ implementation on the VMIC VMEbus SBC.

IV. Implementing a Linux based LDAQ

The first implementation of a Linux based LDAQ was done with a functional system with 2 ReadOut, 2SubFarm and 1 Data Flow Manager crates running in *emulation* mode, the message passing being based on system V message queues instead of VMEbus. This exercise basically consisted of replacing the POSIX.4 calls used for LynxOS with the POSIX1.c calls for Linux and rearranging the resources to have all 5 virtual crates running on the same computer. A benefit of this exercise is to have improved the portability of the applications.

To implement the LDAQ application communicating with other modules over VMEbus under Linux, the low level access libraries written for LynxOS had to be ported to Linux or replaced by functionally equivalent counterparts if this proved to be impossible. These libraries deal with contiguous physical memory, VMEbus access, DMA transfers etc. Since the original target for the DAQ/EF -1 project was to have the final applications on LynxOS, all the Linux implementations are kept API compatible with the LynxOS versions.

The standard Linux kernel does not allow reserving contiguous memory larger than 128Kbytes. Thus, the "Big Physical Area" (BPA) extension [16] has been applied allowing the user to specify an amount of memory which will be marked as I/O space at boot time. Therefore this part of the DRAM, still accessible to the kernel through its physical address, serves as a "shared memory" between different processes which send/receive the pointers to data segments. To let the different applications allocate, use and deallocate segments from this contiguous memory, a simple memory manager kernel module (driver) has been written for both Intel i386 and PPC platforms. The API compatibility requirement with the LynxOS has been fulfilled with a user level library communicating with the driver via simple IOCTL calls.

VMEbus access library was implemented using a static VMEbus mapping done from user space as a part of the bootstrap procedure. To access the PCI configuration space on the VMIC card from user code, special C functions (inl/outl) which require superuser permissions had to be used. On the MVME2x00, the work was much simpler since contrary to Intel architecture, as on PowerPC based cards the PCI configuration registers are memory mapped.

To use the current message passing scheme, each participant VMEbus module reserves a contiguous memory space with a physical address known in advance, accessible by other modules through the VMEbus back plane. This memory, used to store pointers to actual data buffers, was implemented using SRAM on CES RIO or RTPC cards and NVRAM on Motorola MVME cards since they are the only memory locations whose addresses are known in advance. The NVRAM option was also possible with Linux, but the access to NVRAM being bytewise and non-standardisation on the usage of NVRAM between different manufacturers necessitated to find another solution. So, a kernel module has been written to reserve, at boot time, 4 Kbytes of DRAM in a fixed address within the BPA. The DRAM portions reserved for the contiguous buffers in LynxOS can be anywhere in the total system memory, thus one has to map all memory to the VMEbus requiring a compromise between the total number of modules in the crate or their total physical memory. In Linux since all the buffers will be in the initially reserved BPA area, this inconvenience is well handled. Another point which is worth mentioning is the possibility to "clean" the BPA buffers since the bookkeeping is done at the kernel level. This feature adds more robustness to Linux based applications.

VMEbus interrupts are needed in the message passing between different modules. In DAQ/EF -1 project, a general purpose interrupt handler had been written using the VMEbus IRQ and vector decoding facilities provided by LynxOS. The interrupt handling in the Linux Tundra Universe driver has been reused to provide these facilities in the Linux port of this driver. The result was a kernel module which is API compatible with the existing user library. The functionality and performance of this package has been tested on the VMIC board using the RCB8047 module from CES [2]. The interrupt latency at the user program has been measured to be 19 microseconds which is comparable to 25 microseconds that has been previously obtained [17] from PPC based SBCs.

The current message passing scheme requires that any VMEbus CPU can read from and write to both local and remote memories. In an environment where all the other CPUs are big endian, to have a little endian CPU with such a requirement imposes the need of defining a global endianness mode. This mode has been chosen as big endian and on little endian CPUs all the data related to the VMEbus, even the ones to be written on the local memory, have to be byte swapped since it was required to be able to insert and remove the VMIC module without changing the LynxOS API and without disturbing the rest of the system. The resulting message passing library port might not be giving the optimum performance because of the continuous byte swapping but its functionality and performance has been tested using PPC based VMEbus modules. The bandwidth between the VMIC and another 100Mhz PPC based SBC (RTPC8067 from CES) with a custom VMEbus interface has been measured to be 3.9 Mbytes/sec with an overhead of 4.7 microseconds for single cycle access. These values are comparable to the ones previously obtained [18] from two PPC based such SBCs: 3.3 Mbytes/sec for the bandwidth and 4.8 microseconds for the overhead. From these results it is clear that the effect of byte swapping is negligible.

V. CONCLUSION

The Linux evaluation done has shown that it meets the requirements of DAQ/EF-1 project applications with no strong real-time needs. On the same hardware, Linux and LynxOS performances of DAQ/EF-1 project applications are quite similar.

The message passing over VMEbus, essential for LDAQ application, has been implemented and tested for single cycles access between the PC and PPC based SBCs. The byproduct of this exercise is to have Linux on the Motorola MVME 2x00 module as an alternative to LynxOS. The fully functional LDAQ application on the embedded VMEbus PC will be available at the end of summer 1999.

The port to the desktop PC with the Bit3 617 VMEbus interface is expected to be completed by end of 1999.

PC based hardware together with Linux has proven to be a cheap, commodity system to be used in the context of DAQ /EF –1. The effort needed to implement Linux versions of the existing LynxOS applications was minimal because of the common POSIX features of the two systems.

In view of these results, other parts of the ATLAS DAQ/EF-1 system (Sub-Farm and Event Builder) will be made available on Linux in addition to LynxOS.

VI. REFERENCES

[1] G. Ambrossini et al., "The ATLAS DAQ and Event Filter Prototype "-1" Project", *Computing in High Energy Phyics*, Berlin, Germany, 1997.

[2] Creative Electronics Systems, S. A., 70 route du Pont Butin, 1213 Petit-Lancy 1, Switzerland http://www.ces.ch

[3] Motorola Inc., http://www.mcg.mot.com

[4] G. Unel, Evaluation of the Linux Operating System for the DAQ/EF Prorotype –1 Project, Technical Note 110, http://atddoc.cern.ch/Atlas/post-script/Note110.ps

http://www.linux.org and references therein.

[5] I. Alexandrov et al., "BackEnd Subsystem of the ATLAS DAQ Prototype", *Computing in High Energy Phyics*, Chicago, USA, 1998.

[6] Free Software foundation., http://www.gnu.org.

[7] RT-Linux Project, http://www.rtlinux.org/~rtlinux.

[8] Kansas University RT Linux project,

http://hegel.ittc.ukans.edu/projects/kurt/index.html

[9] This device is manifactured by VME Microsystems International corp., 12090, South Memorial Parkway, AL 35803-3308. http://www.vmic.com

[10] This device is manufactured by SBS Technologies inc., 1284, Corporate Center Drive, MN 55121-1245. http://www.bit3.com.

[11] G. Ambrossini et al., " Operating System studies for DAQ applications", *Computing in High Energy Phyics*, Berlin, Germany, 1997.

[12] This device is manufactured by Tundra Semiconductor Corp., http://www.tundra.com.

[13] Linux Lab. Project, http://www.llp.fu-berlin.de

[14] This device is manufactured by Micro Memory inc., http://www.umem.com

[15] This device is manufactured by VMETRO inc., 1180 Diary Ashford, #535, TX 77077

[16] Big Physical Area extensions, http://wwww.uni-paderborn.de/fachbereich/AG/heiss/linux/bigphysarea.html

[17] M. Joos and J. Petersen, General Purpose Interrupt Handler for CES RIO8060 and RTPC 8067 modules running LynxOS, ATLAS DAQ/EF Prorotype –1 Technical Note 19, http://atddoc.cern.ch/Atlas/post-script/Note019.ps

[18] G. Crone et al., Inter and Intra-IOM communication Summary Document, ATLAS DAQ/EF Prorotype –1 Technical Note 120, http://atddoc.cern.ch/Atlas/post-script/Note120.ps