Using Linux in a HEP laboratory

Emanuele Leonardi [a,1], Giovanni Organtini [b,2]

[a] *Università degli Studi "La Sapienza" and INFN - Sez. di Roma P.le A. Moro, 2 - 00185 Roma (Italy)*

[b] *Università di Roma TRE and INFN - Sez. di Roma Via della Vasca Navale, 84 - 00146 Roma (Italy)*

A possible alternative to the use of the MS-DOS or Windows operating systems on a PC is represented by Linux. We present our experience in using Linux in a HEP environment.

Key words: Linux, CAMAC, device drivers, operating systems, free software.

1 Introduction

The Linux operating system is a free Unix-like OS, born for the Intel 80x86 CPU and now ported to several major architectures (Alpha, SPARC, M68000, Power-PC, etc.). It consists of a kernel, initially developed by Linus Torvalds, plus all the software utilities available from the Free Software Foundation GNU project.

We employed Linux in our HEP laboratory as a straightforward solution to the problem of using an off-the-shelf Pentium PC as the center of a full CAMAC based data acquisition and analysis facility. In this paper we will describe our experience with this system.

2 Linux features

During the development work we could appreciate several attractive features of the Linux OS, namely:

[1] leonardi@roma1.infn.it
[2] organtini@roma1.infn.it

- it presents a very high degree of software homogeneity with higher level Unix workstations (Digital, HP, Sun, etc.), widely used in most HEP experiments. This feature makes the porting of programs to several platforms very easy while allowing a ready adaptation of the average HEP user to the new environment;
- the total cost of a complete system is highly reduced with respect to other solutions since free versions of the system software, compilers (C, C++, Fortran) and of several applications are available under the GNU licence and do not need the acquisition of additional software licences. Moreover, PC commodity hardware is much cheaper than its Unix-workstation equivalent;
- most of the CERN standard libraries are available[3] thus allowing the use of very popular packages such as GEANT, Minuit, and PAW on low cost PC's;
- the development of new device drivers is straightforward due to the availability of the entire kernel source code: this means not only that device drivers for new hardware are readily available, but also that developing one's own drivers for specific applications is possible;
- the real multi-user/multi-tasking environment allows the contemporaneous run of several data acquisition, monitoring, and analysis programs by several users;
- the native implementation of all the TCP/IP based networking tools (telnet, ftp, X-Windows) allows for a smooth integration to existing networks and the fast creation of automatic data handling tools and remote access;
- POSIX compliance guarantees an immediate porting of new software packages to all Linux based platforms.
- a real-time[1] extension for those tasks that need short response time from the system was recently made available.

In the following we report on how we exploited these features to carry out all the typical activities in a high energy physics laboratory, namely: data acquisition and monitoring, data analysis and simulation.

3 DAQ using Linux

We connected our PC, a 200 MHz Pentium PC with 32 Mb of RAM and a 2 GB SCSI II disk, to a CAEN C111A CAMAC Crate Controller with a CAEN A151A interface board. This board uses the I/O mapping mechanism so that it appears to the OS as a block of consecutive I/O port addresses: sending commands or receiving data and status bits from the controller is equivalent to writing and reading sequences of 5 bytes to these addresses.

[3] A Motif licence is needed to compile the Motif-based version of PAW++.

3.1 Device driver operation

Interaction between application-level programs and hardware devices connected to the PC proceeds through a *device driver*: due to the low level of the interaction with the hardware, this is usually a software module which is part of the Unix system kernel. ¿From the user point of view, the hardware device is then seen as a simple file and all the I/O operations are performed using standard file operation, i.e. `open`, `read`, `write`, `ioctl`, and `close`.

Under recent versions of Linux, device drivers can consist in separated "kernel modules" which are dinamically loaded or unloaded, thus avoiding recompilation of the whole kernel [2]. This, together with the possibility of using high-level programming languages such as C to create these modules, greatly improves the driver development time.

The device driver we wrote[4] [4] simply translates from the read/write commands sent to a device file into the equivalent read/write commands to the port addresses corresponding to the interface card. As already mentioned, this phase was greatly simplified by the fact that the whole kernel code was available.

Although users can interact with the CAMAC Controller by reading and writing onto the device file, this is not the most natural way to do it. We decided to provide an intermediate-level library, implemented both in a C and a Fortran version, which allows a simpler interaction with the board by using a CAMAC-command related syntax like

```
call c111a(Crate,N,A,F,Q,X,D)
```

which issues a complete command to one of the up to 16 crates connected to the board and retrieves the resulting status and data words in the Q, X, and D registers.

In the A151A board the LAM (look-at-me) request from any CAMAC station can be mapped (if enabled) to a CPU interrupt line[5]. Linux interrupt handlers are part of the kernel and are defined at device driver level. Since in our case the task of the interrupt handler is to execute a sequence of CAMAC commands defined by the end user, one would like to have this sequence dynamically defined without interacting directly with the kernel. With our library it is

[4] This module is heavily based on code developed by Ole Streicher for the Aachen DBCC CAMAC controller board [3].

[5] Due to the different interrupt scheme of the old PCs, this was not possible with the old version of the card (A151), for which an older version of the driver, only working in poll mode, is available

possible to load the interrupt handler with the set of commands to be executed using the same sintax described above. If the function code F exceeds 100 the command is not executed, but the function F-100 is stored into the handler memory. When the interrupt service is enabled, each time a CAMAC module sends a LAM request, the set of commands loaded into the interrupt handler are executed and data and CAMAC status are written to a FIFO queue. It is then responsability of the user level application to correctly retrieve these data from the queue.

3.2 User Interface

Tcl/Tk [5] is a freely available package for the easy development of graphical user interfaces. The functionality of each element of the interface (buttons, list boxes, input boxes, etc.) can be defined using a C-like language. All the low level X-Windows programming, such as mouse control, graphics management, etc., is hidden to the programmer who can then concentrate on the high level definition of the graphic elements.

We built our user interface using a standard layout: the user can select the run type and specify the number of events to be collected by the DAQ, as well as other control parameters, simply typing into input boxes or selecting from predefined lists. At the "Start of Run" signal, triggered by the pressing of a button on the interface, the DAQ program is launched and all the parameters entered by the user are logged onto a file. The run number is automatically assigned and the proper trigger conditions to be used for the chosen run type are automatically selected by writing a bit pattern to a CAMAC Output Register, which in turn controls the NIM based trigger logic.

All data collected during the day are automatically copied overnight to an analysis facility, consisting in a cluster of HP9000 workstations, using an FTP based transfer.

4 Monitoring

DAQ Monitoring is accomplished by using the shared-memory routines available from the HBOOK package of the CERN libraries [6], freely available for several platforms to any HEP institute: histograms are booked in the shared area and filled during DAQ. Access to the histograms is then possible without interfering with the running DAQ. In our case, we decided to use PAW [7] as the monitoring program in order to implement a simple on-the-fly fitting procedure.

A simple macro, automatically executed by PAW at start-up, takes care of regular updates of the histograms shown on the screen. As the start-of-run phase, the monitoring program is controlled by a Tcl/Tk-based user interface which interacts with the PAW macro via a control file containing the user's requests. In this way the user can select which histogram is to be shown, the time interval between updates and the lower and upper limits of the histograms.

During execution, the monitoring program can be temporarily stopped without affecting the DAQ. Stopping the monitoring task allows the user to use any PAW command and, in particular, to execute any operations on the histograms and to run some fitting procedures.

The real multi-user/multi-tasking environment allows more than one monitoring task to run at the same time. TCP/IP-based remote X-Windowing allows the user to monitor an on-going run even from a location far from the laboratory where the measurement is taking place: this feature is especially attractive if one wants to perform very long unattended runs. Moreover, multi-tasking allows any other activity to take place during the run, the only drawback being the loss of computer resources for the data acquisition process: this may translate into a reduction of the maximum DAQ rate supported by the system.

5 Data analysis and Simulation

Data analysis on Linux can be done using the same tools available on higher level platforms. Interactive programs like PAW and software packages like ZEBRA, HBOOK and Minuit exists for Linux. This makes the transition from workstations to PC very easy, since the availability of all the major data analysis software packages makes the porting of existing programs to the new architecture straightforward: most of the time, in fact, only the compilation scripts must be slightly modified to adapt to the GNU compiler options. The most frequent problem arises from the non standard ordering of declaration and data statements sometimes used in existing FORTRAN programs. This problem can be solved by using the `f77reorder`[8] extension provided by the HERMES[9] collaboration.

As for data analysis, the porting of existing simulation software or the creation of new applications is very easy on Linux since the widely used GEANT library [10] is available togheter with the other CERNLIB tools.

6 Performances evaluation

We measured the performances of our Linux box both for data acquisition and for simulation.

6.1 Data Acquisition

To measure the performances of our device driver we loaded the interrupt handler with a set of write instructions which set and reset continuously the first bit of an Output Register.

We showed the resulting square wave on an oscilloscope connected to the output line corresponding to that bit. A period (18 ± 1) μs was observed, leading to the conclusion that the minimum time required to perform one complete CAMAC operation is about 9 μs [6]. This allows a maximum DAQ rate slightly above 100 KHz.

The latency time between the interrupt signal and the execution of the interrupt handler was measured looking at the time needed to reset a bit on an Output Register by the interrupt handler after the same bit was set by hand. The setting of the bit was followed by a LAM signal automatically generated by the bit true status. Subtracting the time needed to set the bit previously measured we are able to set an upper limit to the latency time to be (14 ± 1) μs.

6.2 Simulation

We used as a benchmark a complete GEANT program simulating electrons impinging on a scintillating crystal and the subsequent propagation of the light inside it.

We ran the same program on a 99 MHz HP9000/735 and on a 100 MHz Pentium with Linux. The program was compiled using fort77 and g77 under HP-UX and Linux respectively. In both cases we used the default optimization levels.

The performance ratio between the time needed to execute the program on the HP and on the PC is about 1.9.

[6] This time can be slightly reduced by hard-wiring the interrupt handler routine directly into the device driver

7 Conclusions and prospects

We tested a DAQ set-up based on a CAMAC crate connected to a PC running the Linux OS. The system proved to be fast enough for all our applications while the multi-tasking OS allowed monitoring and analysis to proceed in parallel to DAQ. We provided a C and Fortran interface to the CAMAC controller to grant the end-user the widest access to the facility and the possibility to use the CERN libraries.

We also tested the performances of our system in running simulation programs for high energy physics obtaining a good cost/performances ratio when compared to a Unix workstation.

Thanks to the use of low-cost technologies and the free availability of all the software packages, the final system resulted very cost-effective when compared to more sophisticated solutions.

References

[1] M. Barabanov and V. Yodaiken, *Linux Journal* **34** (1997) 19. See also `<http://luz.cs.nmt.edu/r̃tlinux/>`

[2] A. Rubini, *Linux Journal* **23** (1996) 10.

[3] O. Streicher, anonymous ftp to `<ftp://ftp.ifh.de/pub/unix/linux/measurement/>`

[4] The latest version of our C111(A) device drivers and user libraries can be obtained via anonymous ftp to `<ftp://hpl3rm06.roma1.infn.it/pub/linux/local>`

[5] J.K. Ousterhout, "Tcl and the Tk Toolkit", Addison-Wesley, 1994, ISBN 0-201-63337-X.

[6] CN/ASD Group documentation, `<http://wwwcn.cern.ch/asdoc/Welcome.html>`.

[7] R. Brun *et al.*, "PAW – Physics Analysis Workstation", CERN Computer Center Program Library.

[8] W. Wander, available from `http://lars.desy.de/f77prob.html`.

[9] D. P. Barber, *et al.*, Nucl. Instr. and Methods A338 (1994) 166-184

[10] R. Brun, *et al.*, "GEANT 3", CERN DD/EE/84-1 (Revised) Sept. 1987.