Portable Generic Data Acquisition Software

Dr P.A.Elcombe

The Cavendish Laboratory, Cambridge.

While experiments become larger and take many years from design to completion the development time for new computing hardware is continually reducing. Significant software investment can be preserved by making data acquisition software as hardware independent and portable as possible.
A Portable Buffer Manager has been developed that is POSIX compliant and can run on VMS, OS9 and UNIX systems. The design and implementation are discussed.

Introduction

It should be clear to anyone involved in large HEP experiments that it is exceedingly likely that any hardware that they may envisage NOW in which to run their software will almost inevitably be superceded - possibly many times - during the entire lifetime of the experiment. It is also likely that different subgroups of an experimental collaboration may wish to use different hardware for their own developments.

In order to minimise problems of different hardware platforms ending up with different data acquisition systems it makes sense to develop a Portable Generic Data Acquistion System. This is in the same vein as the rapid move towards 'open' systems within the CPU market. The intention is that applications related to data acquisition will be able to be moved easily between hardware platforms so as to take best advantage of available technology.

Traditionally there has been a close linking between DAQ software and the operating system and hardware on which it runs. Writing and porting device driver software is not easy. However in modern DAQ systems the proportion of software that communicates directly with external hardware is a small proportion of the whole, and a lot of that can now use standard libraries - eg for CAMAC or FASTbus access - so there is no real reason why the code should not be made largely portable.

Another reason for non portability has been the use of operating system specific calls for the special sorts of operations that real time DAQ software requires. In fact the number of such calls is not very large and by careful design they can be removed to a small library of operating system dependent calls, leaving the rest unchanged when porting to different environments.

1 Portable Buffer Manager

The heart of any data acquisition system is a buffer manager. Thus the first job in making a portable DAQ suite, is to make a buffer manager that is portable.

The Portable Buffer Manager [PBM] is based on the original OS9 buffer manager written for OPAL. It has all the things that you might want in a buffer manager including:

- Multiple streams.

- Any number of consumers per stream.
- Random and 100% consumers.
- Easy to change stream and consumer configuration.
- Farming of event to parallel copies of the same consumer.
- Sensible action if a consumer crashes while processing an event.

Data buffers are passed between consumers by using descriptors. This has a very low overhead, and also means that there is no restriction on the contents or format of the data buffer.

Thus the user has considerable flexibility in setting up PBM for his own needs. There is no design limitation in any of the configurable parameters, so it can easily be configured to as large a system as the target CPU provides and the user wants.

2 Implementation details

The goal of portability implied that PBM has to be available on OS9 systems so as to still be useful for OPAL. The ability to run on VMS and UNIX systems was also required. In the fullness of time, POSIX compliant code should be the answer, but at present not all possible target systems have support for POSIX.1, and the real-time extensions [POSIX.4] are still evolving.

The initial implementation used VOS [1] which was a good starting point. The relevant routines were then given suitable alternatives using direct OS9 calls. Later the POSIX [2] compliant calls were added as a third option.

Conditional assembly in C code is used with the main compile time options being POSIX, VOS, and OS9. The number of routines within PBM containing conditional code has been made as small as possible. User code does NOT contain any conditionals, but there are some within the standard include files provided by PBM.

To cope with the evolving POSIX.4 standard, there is a separate small library of routines that invoke the relevant system calls. Versions of this exist for POSIX.4D9, POSIX.4D10, and also for SUN and Apollo using direct system calls. This library has been kept small by using the smallest possible number of POSIX.4 functions. For example - rather than using the IPC facilities in POSIX.4 the simpler POSIX.1 signals are used instead. PBM is known to run on native OS9, OS9/VOS, VMS/VOS VMS/POSIX, SUN and Apollo systems. Porting to another environment should be very simple.

It is implemented as a linkable library - with the option of being shareable in OS9, VMS and versions of UNIX that support shareable libraries. The library is quite small and the memory saving by making it shareable is quite small, but the great gain is the fact that a new library can be made without requiring users to relink their programs.

For testing there is a small skeleton data acquisition suite that shows how to initialise a stream and provides a dummy event generator and some dummy consumers. There are also some simple control programs.

PBM is available and in use regularly by one subdetector within OPAL. It may be used in future OPAL upgrades, and in the NA48 data acquisition system. The source is maintained

with a simple text library package. Building on a new target system simply requires obtaining the source file and an appropriate build job - which may have to be modified depending on the details of the target system. For access to the source, contact the author.

3 Performance

The fact that it is portable makes it possible to compare identical suites on different systems. For the following measurements a single stream of three stages was used. The first stage generated events of gradually increasing size up to 2048 bytes, repeating the sequence. The second stage was used to discard unused space in the buffer. The third stage did nothing useful at all. Events were generated in groups of up to 20, and with a delay of about 10msec between groups. The delay and group size were optimised on each system to get the best throughput. These differences are caused by different system clock resolution and job selection methods in the different systems.

The measurements confirm what might be expected. For best throughput per unit of CPU power a small specialist real time system [OS9] is best. VMS is fairly good, and naturally enough is better without the extra overhead provided by the VOS layer. The pure UNIX systems are least good for high throughput. However the needs of a particular application should be borne in mind. If the event rate is small but the CPU requirements high, a UNIX system may well provide the best overall system.

Operating System	Processor	Dhrystones	Hz
Native OS9	MVME167 [68040]	14300	1110
Native OS9	FIC [68020]	2450	180
VOS/VMS	VAX 4000-60	16600	330
POSIX/VMS	VAX 4000-60	16600	625
SUNOS	Sparc IPX	33000	220
Apollo Domain	DN10000	25000	370

Table 1: Best throughput in different systems (Hz).

4 Pitfalls

Designing portable code is largely a matter of following the advice given in standard programming books about structured code, however there are one or two extra ones that arise in a real-time environment:

- Data structures have to be designed such that there are no C pointers in any areas of memory that might be shared between processes. A side effect of this is some rather opaque code setting up the required pointers within user routines. Different C compilers have different views as to the legality of this code.
- Closedown routines are a standard feature of PBM consumers and producers. Depending on the target system these might be called directly from genuine ASTs, or indirectly in

normal user context from a signal catching routine. Some care was needed to make sure that the user closedown routines were able to perform correctly.

- There are only two user defined Signals within POSIX.1 - they are both used and this will cause problems if linking with other libraries that also require signals.
- A further complication will arise if trying to use X11 for displays, since the thread selection that they use will almost certainly interfere with the real-time job selection built into PBM. Users will have to make a clear separation between PBM code and X11 code.

5 Future developments

PBM is only the start of a complete portable DAQ suite. In order to be able to combine the systems in several independent CPUs into a unified system additional services and utilities are required, such as: error message utilities; process creation and control; network communication services; event transport between CPUs; display programs for self diagnosis and display of simple experimental results.

I have ideas as to how to implement a lot of these, but need the time to get on with it. As far as possible all code I produce will be portable from now on.

6 Conclusion

It is highly desirable to develop data acquisition code in a portable manner. This has been shown to be possible for a buffer manager, and further components will be produced to try to make a complete suite that can be run on a large variety of target operating systems.

The existence of portable DAQ code frees the user of the restrictions that have traditionally existed that lock them into using a particular operating system and/or CPU. Once the implications of development in a portable way are understood it makes software development easier since the developer can use any handy system on which to continue working.

References

[1] VOS - a Virtual Operating System, R D Russell and G Mornacchi, ECP division, CERN.

[2] POSIX - Portable Operating System Interface, IEEE 1003.1.