E787 Data Acquisition Software Architecture

M. Burke, L. Felawka, R. Poutissou,
TRIUMF, 4004 Wesbrook Mall, Vancouver, BC,
V6T 2A2, Canada

and

S. Adler, J. Haggerty, R. Strzelinski, C. Witzig,
Physics, Brookhaven National Laboratory, Upton, NY,
11973, USA

Abstract

Brookhaven National Laboratory (BNL) Experiment 787's second generation Unix-based data aquisition system is comprised of several independent programs, each of which controls a specific aspect of the experiment. These programs include packages for reading events from the hardware systems, analyzing and reducing the data, distributing the results to various data consumers, and logging the data to tape or disk. Most of these can be run in stand-alone mode, for ease of development and testing. There are also a number of daemon processes for writing special data records to the data streams, and several monitor programs for evaluating and controlling the progress of the whole. Coordination of these processes is achieved through a combination of pipes, signals, shared memory, and FIFOs, overseen by the user through a Motif graphical user interface. The system runs on a Silicon Graphics 4D/320, interfaced to a Fastbus system through the BNL Fastbus/VME interface (BBFC), and runs under Irix and Motif/X-windows.

I. Requirements

Brookhaven AGS (Alternating Gradient Synchrotron) Experiment 787's first generation data acquisition system was MicroVax 3200/QBUS based, and relied on a farm of 68020-based ACP nodes for online data processing and compaction. The QBUS bandwidth of 1.5 Mb/s, and the ultimate throughput of 233 kb/s to two tapes were adequate for the first phase of the experiment, ending in 1991. Upgrades starting at that time included a new beam line, offering three times the beam intensity as before, and a detector upgrade with new instrumentation, including 1000 channels of CCD transient digitizers. The event size is expected to triple once the second phase is complete. The old data acquisition system could not handle this throughput, so we undertook to design a new online system in the spring of 1992. The goals were to maximize event throughput within the constraints of the front-end electronics, the periodic beam cycle of the AGS, and a tripled beam intensity.

A. *Hardware*

We decided on Silicon Graphics' (SGI) Power Series 4D/320 computer to serve as the computing engine of the new data acquisition system. Advantages of this system included multiple, high-performance, upgradeable CPUs, a versatile and industry standard Unix-based operating system, and a high-speed I/O subsystem including two SCSI buses and a VME bus integrated into its internal architecture. A clear upgrade path exists from this architecture to SGI's new Challenge architecture.

In our 1992 data-taking run, the computer configuration included the Irix 4.0 operating system, two R3000 CPUs, and four Exabyte 8mm tape drives (three model 8200s, and one model 8500).

The front-end electronics were not changed to accomodate the new data acquisition system. In particular, SLAC Scanner Processors (SSPs) [1] continue to be used to accumulate data from the FASTBUS crates during the one second of beam in each 3.2 second spill. A trigger SSP notifies the secondary SSPs when an event of interest has occurred, and signals them to read out data from their respective FASTBUS crates into memory. Between the bursts of beam, a master SSP takes over the process of building events and transferring them over the branch bus to the SGI. A new high-speed fastbus to branch bus interface (BBFC) [2] was designed, coupling the front end electronics to the SGI's VME backplane. Data transfer between FASTBUS and the SGI memory was measured at approximately 17 Mb/s in each direction.

B. *Software*

Communication with the BBFC was facilitated with a new device driver [3], which, along with the standard high-level IEEE FASTBUS routines, and the BNLSSP routines [4], were ported to the SGI. These packages eased the porting of other code that interacts with hardware on this system.

A philosophical decision was made to develop the online system as a suite of independent processes, rather than as a single, monolithic program. The advantages included greater exploitation of the multi-CPU SGI architecture, and greatly simplified development and testing, since each component is devoted to a single task and can in principle work as a stand-alone program. The main disadvantage, namely the management and synchronization of numerous processes in a multi-tasking environment, is largely handled by the Unix operating system and its built-in facilities for inter-process communication.

Our software wish list included an X11/Motif user interface to the system, in order to enhance ease of operator training, idiot-proofing, and overall trendiness. The *XDe-*

signer CASE tool [5] was selected as a Motif user interface development tool, and facilitated rapid prototyping and changes to the visual appearance of the system. Networking support was also desired, to off-load non-essential, but CPU intensive, tasks to remote computers.

These features were all available under Unix/Irix. By using the built-in facilities of this operating system, we minimized the amount of low-level system programming and development that was necessary, and were able to focus our programming efforts exclusively on the higher-level online tasks.

II. Architecture

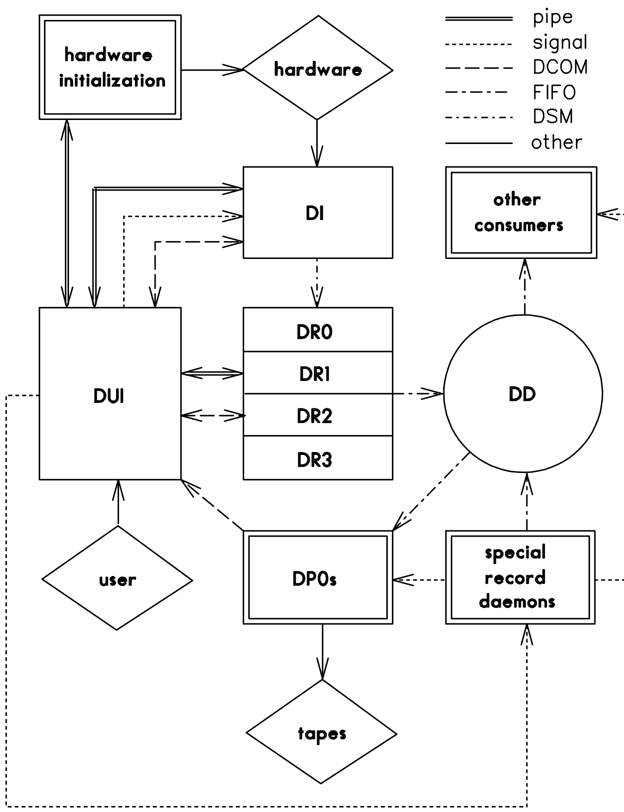

Fig. 1. The E787 online system architecture.

A. Data Flow Management

The problems of moving the data through the online system were addressed using three systems based in shared memory. The first was the DSM (data shared memory manager) whose job was to rapidly transfer data into our data reduction (DR) programs. Speed is a priority in this serial process, and little manipulation of the data is required.

Later in the system, it is necessary to gather events from these various programs (called producers, since they inject new data into the system) and then redistribute them to an arbitrary number of programs (called consumers) who request events of specific types. Speed at this point is not as much of a concern as orderly control of the data flow. This is managed by a system of FIFO buffers called the DD (data distributor) [6]. The FIFOs are used to pass event pointers around until all the consumers who want events of a class have had a chance to use them. Consumers can access the DD over TCP/IP using a special server program.

The third shared memory system in place is the DCOM (data communication), a shared memory area that is largely used for inter-process communication. Typical usage involves each process recording its current status and operating parameters in this area for other processes to examine and then adjust their behaviour accordingly.

B. Processes

B.1. DI (data input)

Only a single instance of the DI is running at any given time. Once started, it simply runs in an infinite loop, reading large (100 kb) data blocks out of the FASTBUS modules over the Branch Bus and BBFC. These data blocks can contain multiple events. They are passed to the DRs (see below) system via DSM.

B.2. DR (data reducer)

Up to sixteen instances of this producer process are permitted to run simultaneously, although four was the default number during the 1992 data run. The DRs decompress the data coming from the DI, and break it down into individual events. Online event analysis and data processing can occur at this stage, including rejection of unacceptable events (this stage is known as our Level 3 Trigger). Good events are passed into the DD.

B.3. Data Consumers

Data consumers attach to the DD and request events of certain classes (determined by the hardware triggers that are generated by each event). A consumer can therefore be configured to select every event that passes through the system, or only a very limited subset of those events, as appropriate to its task. Events are shared among consumers, so that an event will eventually get passed to every consumer that requests it.

The DP0 process manages the logging of data to a single device. Multiple instances are allowed if there are multiple destinations (eg. several tape drives) for the data. The DP0s are configured to accept only certain event classes, which are different for each instance of the process. The result is that they perform online event sorting to different tapes, a function that was previously performed off-line. This stage was formerly known as Pass 0, which gave rise to the name of this process (DPO = “data pass 0”).

Other data consumers are not essential to the data acquisition process, but can be helpful for diagnostic purposes. For example, graphical event display packages can request the occasional event coming through the system and display it for the user/operator. A Quality of Data (QOD) diagnostic package is regularly run using incoming data to quickly assess problems with the instrumentation or software. These consumers are often CPU intensive (QOD, for example, will happily impose a 100% CPU load on the system), but can be off-loaded to remote nodes since the DD is accessible via TCP/IP.

Fig. 2. The DUI (user interface).

B.4. Special Record Daemons

The special record daemons are a special type of data producer that are invoked at certain times to write special records to the data stream. Four types of special records are currently supported: comments, which are written at the beginning of every run and any other time at the user's discretion; scalers, which are written once per spill; beginruns, which are written at the start of every run; and endruns, which are written at the end of every run. These records are generated and broadcast to all data consumers by daemon-like processes that sleep until signalled by the DUI (see below).

B.5. DUI (user interface)

The user interface (figure 2) is an X11/Motif program that allows the operator to automatically start up and initialize all the various processes of the system, quickly and easily navigate the states of the online system, initialize hardware, and monitor the progress of the runs. It polls the DCOM area to track the states of the various online processes and reports relevant information to the user through a status window. It also monitors pipes that are attached to each of the processes in order to pick up messages and display them for the operator.

At any time the DUI is in one of seven states: initialization, start-up, beginrun, active, paused, endrun, or shutdown (figure 3). The appearance of the user interface changes in each state to show the operator only those options and state transitions that are available at that time.

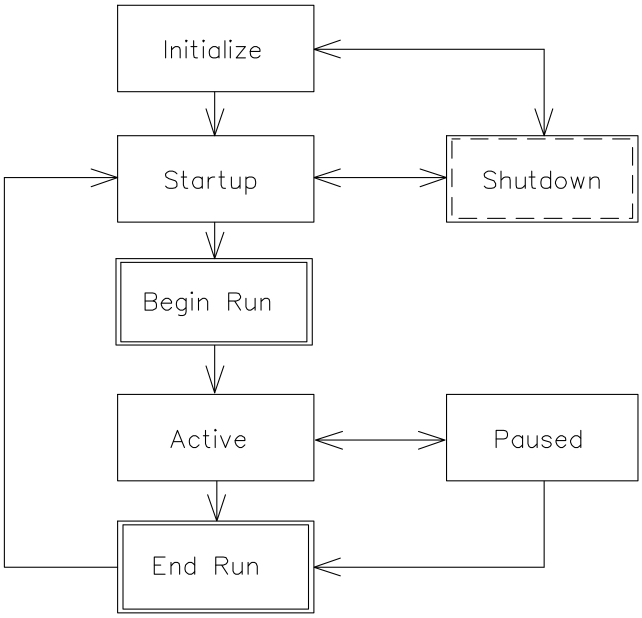

Fig. 3. State diagram for the online system.

Development and testing of the DUI was greatly simplified by the *XDesigner* user interface builder tool. It proved so effective for generating graphical user interfaces that snappy little GUIs were written for most of our monitor and testing programs. Several, including applications to monitor the DD system status and an interface to the DP0s, have become integral components of the online system.

III. Communication

A variety of inter-process communication methods are used to coordinate the online processes (figure 1).

All processes/systems started by the DUI (including DI, DD, and DRs) remain connected to it through pipes. This allows the DUI to easily monitor and display error messages and other output. It also allows the child processes to be designed as simple command-driven programs, which makes for simple development and testing. Most of these programs can therefore be run in stand-alone mode with keyboard input. When incorporated into the online system, the DUI hijacks the standard input of the program and writes the command strings itself; no special changes to the child process are needed to make it realize it is now part of a complex system of processes.

Signals are used in a few circumstances, namely to wake up the special record daemons, to get the DI to pause its data-reading loop or read a command, and to notify data consumers that a special record is available for reading.

Shared memory is used to pass data and to keep other processes apprised of a program's progress.

Semaphores are used in the DD system to access an event in a FIFO. Because Unix prevents more than one process from using a semaphore at a time, this facility provides a built-in FIFO-locking and unlocking mechanism that prevents collisions and data corruption by competing processes.

IV. Performance

Tests in the 1992 data run resulted in a maximum throughput rate to tape of 1.1 Mb/s. These tests were conducted with four 8mm tape drives logging the data, one of which was an Exabyte 8500 high-density drive. CPU usage varied from 30% (half of which was system activity) for large numbers of small events to 50% (one third of which was system activity) for smaller numbers of large events. The largest CPU consumer appeared to be the DP0s and their costly semaphore operations.

The main performance bottlenecks in the system were:

Insufficient SSP Memory. This bottleneck was the most limiting factor in the above throughput figure. Enlarged SSP memory is planned.

Tape Speed. This bottleneck is easily alleviated by writing to several tape drives at once. Performance of the model 8500 drive was observed to be only 70% of the maximum 500 kb/s due to the less than optimum record size of 8 kb that was used. Record sizes of at least 32 kb have been shown to be necessary to obtain maximum performance, and are planned in future runs.

CPU. Although the system was not CPU-limited in the 1992 configuration, future online analysis and data reduction will cause CPU power to become a factor. Some reduction of the CPU load was achieved by off-loading non-essential data consumer processes to remote nodes, and further reduction is possible by eliminating the little-endian data format that now results in a large amount of byte-swapping occuring during the online process. Otherwise, it is straight forward to simply upgrade the SGI's CPU configuration.

V. References

[1] Brafman, H., et al., "The SLAC Scanner Processor: A FASTBUS Module for Data Collection and Processing", IEEE Trans. Nuc. Science, **Vol. 32**, 1, 1985.

[2] Haggerty, J., et al., "A High Speed FASTBUS Interface for VME", Conference Record of the Eighth Conference on Real-Time Computer Applications in Nuclear, Particle, and Plasma Physics, pp. 103–104, TRIUMF, 1993.

[3] original device driver due to Michael Isely, FERMILAB.

[4] see "BNL Access Routines for the SSP User's Guide", by Robert Hackenburg, BNL.

[5] VI Corp, 47 Pleasant Street, Northampton, MA, 01060.

[6] Witzig, C. and Adler, S. "The E787 Unix Based Event Buffer Manager", Conference Record of the Eighth Conference on Real-Time Computer Applications in Nuclear, Particle, and Plasma Physics, pp. 123–125, TRIUMF, 1993.