E787 Technical Note #264
The E787 UNIX Based Event Buffer Manager

C.Witzig, S.Adler
Submitted to the Eigth Conference on Real-Time Applications
in Nuclear, Particle, and Plasma Physics
June 8-11, 1993
Vancouver, Canada

Abstract

The E787 data distributor - a UNIX based event buffer manager for AGS rare kaon decay experiment E787 - is presented.

I. Introduction

The E787 data distributor (called DD) is a UNIX based event buffer manager for AGS rare kaon decay experiment E787. It is written entirely in C and currently runs on a Silicon Graphics 4D/320 under Irix 4.0.5.
The data buffers that are received from the front electronics are distributed to different consumer processes for data logging, online event analysis, data reduction, and monitoring. A link over the network to outside computers is also possible. Internally the bookkeeping is based on a software implementation of FIFOs using shared memory and semaphores.
In the following we first describe this FIFO system, the advantages of its use in a buffer manager, and then show how it is implemented in the DD system. It should be noted that the FIFO software is a standalone package and the DD system is only one possible application of it.

II. The FIFO System

A. The FIFO Software Package

A FIFO (first in, first out) is a ring buffer with a read and write pointer. At initialisation time they both point to the beginning of the buffer. With every write/read operation they are incremented modulo buffer size, with the condition that the read pointer is not allowed to overtake the write pointer.
In the C programming language FIFOs can be realized by the type definition

```
typedef struct a_fifo{
        struct fifo_header fhdr;
        int cnt;
        int wrptr;
        int rdptr;
        struct fifo_entry fentry[FIFO_DEPTH];
};
```

where the fifo header contains information about the nature of the FIFO, its name, the PID of the creating process etc., *wrptr* and *rdptr* are the pointers for the next fifo read/write operation, *cnt* the number of entries in the FIFO, i.e. the difference between the write and read pointer. *fentry* is the proper FIFO, an array of entries represented by a C structure. E.g. in the case of the DD system it is the following structure

```
typedef struct fifo_entry{
        int shmid;
        int dboff;
        int *p2da;
        int len;
        int ctlw;
};
```

where *shmid*, *dboff* and **p2da* are the shared memory id, the offset within that shared memory segment and pointer to the shared memory location where the event resides, *len* the event length and *ctlw* a control word about the nature of the event.
Four basic operations are needed for accessing the FIFOs, namely
FIFO_MAKE(char *fname),
FIFO_CLOSE(int which_fifo),
FIFO_READ(int which_fifo, struct fifo_entry *) and
FIFO_WRITE(int which_fifo, struct fifo_entry).
The FIFOs reside in shared memory and any process after an initial attachment to the FIFO system can read and write into any FIFO. A set of semaphore operations (*FIFO_LOCK*(int which_fifo) and *FIFO_UNLOCK*(int which_fifo)) guarantees that only one process accesses one FIFO at a time.
Two exceptions can occur, namely when a process attempts to write into a full FIFO or tries to read an empty FIFO. In the former the process falls asleep until another process takes another event out of that FIFO. In the latter the process either returns with an error status or falls asleep until another process puts an event into that FIFO. These situations are also handled using semaphore operations.

B. The Implementation of the FIFO System in the DD

The bookkeeping of an event buffer manager can be done

using this FIFO scheme. Each event which is injected into the DD system, is copied into shared memory. A FIFO entry (see above), containing necessary information about the event like its exact location, length etc, is created and written into the first FIFO.
A data buffer system comprising different stages can be realized by associating every stage with a given FIFO. There are several advantages in using such an approach for event distribution among an arbitrary number of processes:

- Once an event is copied into the data buffer, only a C structure is passed between the processes.
- Once a process takes such a C structure out of a FIFO, it is guaranteed automatically that no other process accesses this event by mistake, as it "disappears" from the FIFO system as long as this process uses it. Once the process is finished analysing the event, it puts it back into the FIFO system by writing it into the next desired FIFO. This has the favorable effect that the load of semaphore operations is distributed among several processes.
- A set of FIFOs can be used to route the events through a set of stages. Different paths for different classes of events can easily be realized. Once an event is no longer needed, it is simply eliminated by freeing its shared memory space and making the latter available for a new event.
- It is not necessarily desirable that every event ends up in every FIFO. A set of control words in each FIFO header, which is initialised as the FIFO is created, have to match control words in the event header in order that a given event is put into the FIFO.
  E787 uses the trigger bits as the control words in order to distribute the different event types to the proper FIFOs.
- As any given number of processes can attach to the same FIFO, a CPU intensive stage can be done by several processes in parallel, thus avoiding bottlenecks in a multiprocessor architecture.
- As FIFOs can be dynamically created and removed from the DD system, processes can attach/detach at any time to/from the DD system, creating a very flexible system for the coordination of the processes.

III. The DD System

The DD system is set up by a control program DD_CTL, which creates the FIFO system and the event buffer, consisting of a given number of fixed length slots, into which the data buffers will be copied. A first FIFO ($INPUT\_FIFO$) is created and filled with pointers to these empty event locations for future use.
The DD_CTL then either resides in the background until the system is removed or alternatively accepts monitoring commands through its standard input. An X11-OSF/MOTIF interface was developed for easy interactive use.

Processes create/remove their FIFOs as they join the DD system. The environment variable FIFONAME specifies which FIFO to attach to. If a FIFO of this name doesn't exist, it is created, otherwise the process will start to share this FIFO with the processes that are already attached to it. Processes attached to the same FIFO receive the events on a round robin basis. The nature of the events that are expected to be put into a FIFO is specified by the control word in the FIFO header.
Empty events can be obtained at any time from the $INPUT\_FIFO$. If the fixed length buffer is too small, additional memory is allocated "on the fly" and destroyed if no longer used.
If an event at a given stage has to be rejected, all there is to do is to put its pointer back into the $INPUT\_FIFO$. Modifications of the event can be done at any stage. If the modified event is too big, additional memory can be requested.

Figure 1 shows the current E787 use: Data is injected by the data reducers and distributed to the data loggers, online event monitoring, online event display etc. As the DD software can be linked to the offline E787 package, any offline code is able to run online, enabling among other things easy online calibration.
The data is routed through the DD system depending on the trigger word, which is part of the FIFO entry header for every event. Different classes of events are passed to different data loggers writing to an arbitrary number of tape devices. This schemes avoids the bottleneck of writing to one single tape device and enables the separation of the different event (trigger) types before logging to tape, i.e. at the earliest possible stage (see fig. 1).
A broadcast mechanism exists in order to syncronize all the processes attached to the DD system and/or to pass a special event to all of them. After sufficient copies of the broadcast event have been written into the "broadcast FIFO", every process attached to the DD system receives a signal upon which it reads this broadcast FIFO. Examples for this operation are "begin run", "end run" and other special records.

A process can also attach to the DD system over the network by attaching to a FIFO of name "FIFONAME@host". In this case a server program is started by the INET demon on the host computer and a socket connection between the two processes is established. The server then receives the events from the DD system and sends it over the network.

A number of X11-OSF/MOTIF based debugging and monitoring tools have been developed. They are forked off by the DD_CTL program if desired. Currently supported are a floating bar diagram for monitoring the throughput and contents of the FIFOs, and programs for continuous monitoring of the processes that are attached to the DD system. An event dumper is also available which retrieves events of the desired type from the DD system for online debugging.

The DD system was used for the first time in the E787

1992 data run. The performance is limited by the event size. For large events (the average event size in the E787 experiment is 20-30 Kby) the performance depends on the data transfer time into the DD event buffer or the processing time of the processes. In the latter case a bottleneck by a CPU intensive stage can be avoided in a multiprocessor environment by attaching several identical copies of the process to the same FIFO and processing the events in parallel. In the case of very small events the system performance is limited by the competing FIFO access, i.e. semaphore operations. An overhead of 1ms per event has been measured in that case.

As future development a fast network link between several DD systems on different hosts is planned. It is also envisaged to expand the DD system for event distribution in the offline analysis on multiprocessor computers.

IV. Acknowledgements

The authors would like to thank John Haggerty for neverending encouragement. C.Witzig would like to thank his friends C.Bee, D.Francis, G.Heyes and L.Sacks from CPLEAR online, where he learnt the advantages of using FIFOs.

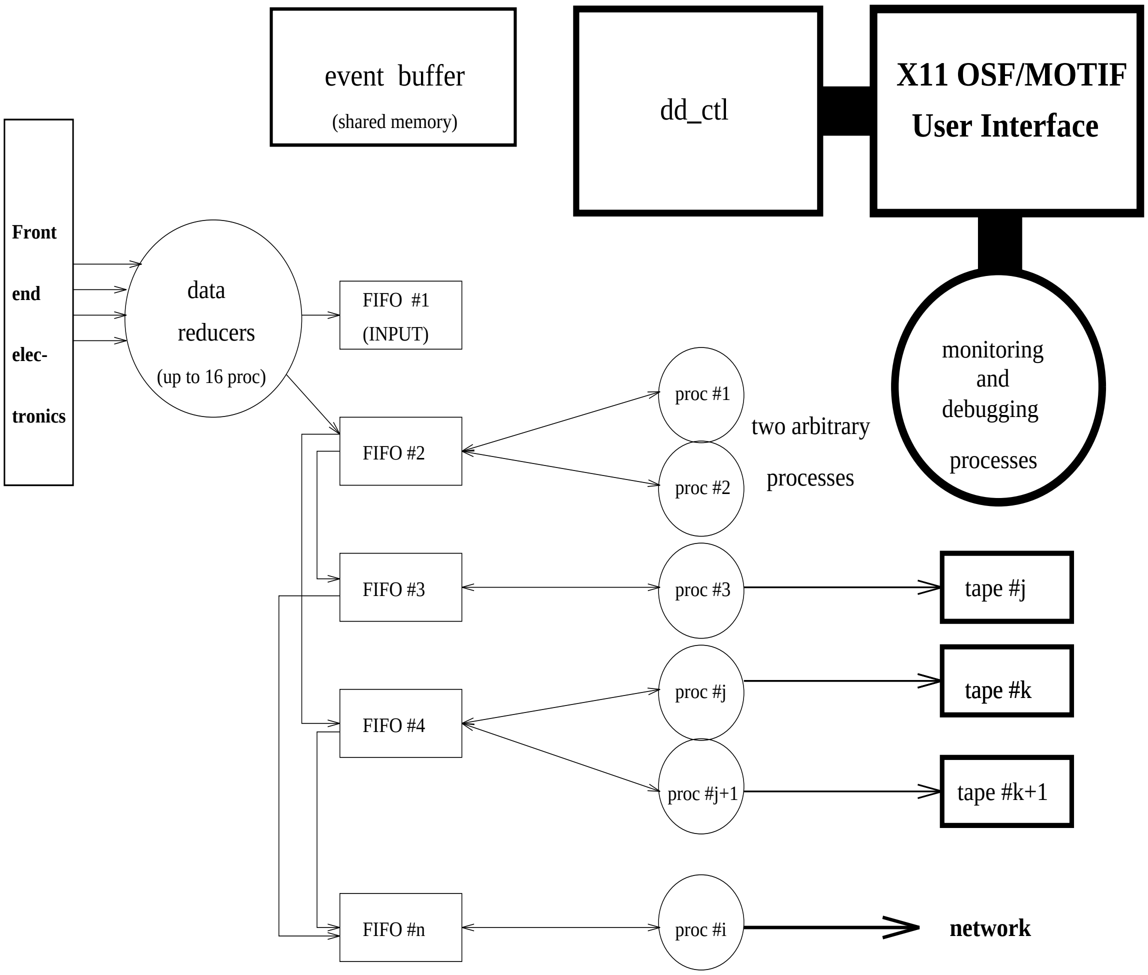

Fig. 1. Overview of the E787 DD system.