SDC-92-397

SDC
SOLENOIDAL DETECTOR NOTES

DEVELOPMENT OF DATA ACQUISITION SYSTEM USING RISC/UNIX™ WORKSTATION

February 16, 1993

Y. Takeuchi and T. Tanimori
Tokyo Institute of Technology

Y. Yasu
KEK

Development of Data Acquisition System using RISC/UNIX™ workstation.

Y.TAKEUCHI and T.TANIMORI
Department of Physics, Tokyo Institute of Technology, Meguro, Tokyo 152, Japan

Y.YASU
National Laboratory for High Energy Physics, Tsukuba, Ibaraki 305, Japan

Abstract

We have developed a compact data acquisition system on RISC/UNIX workstation. SUN™ SPARCstation™ IPC was used, in which extension bus "SBus™ " was linked to VMEbus. The transfer rate achieved was better than 7 Mbyte/sec between VMEbus and SUN. A device driver for CAMAC was developed in order to realize an interruptive feature in UNIX. In addition, the list processing of CAMAC function has been incorporated in order to keep the high priority of the data handling process in UNIX. The successful developments of both device driver and list processing have made it possible to realize the good real-time feature on RISC/UNIX system. Based on this architecture, a portable and versatile data taking system has been developed, which consists of graphical user interface, I/O handler, user analysis process, process manager and CAMAC device driver.

1 Introduction

UNIX workstation combined with RISC architecture has advanced to such a level that its performance has become as good as or even better than a main frame computer. In addition, the cost performance has been decreased drastically. Due to this high performance, RISC/UNIX has been widely used in the high energy physics community as a personal workbench for off-line analysis instead of main frames. However, RISC/UNIX has not been adopted yet to an on-line data acquisition as an interruptive processor, except as a monitoring computer, since UNIX does not provide good real-time features required for a data acquisition system in contrast to VAX™ /VMS™ . Although a few attempts have been reported toward that direction, none of them seemed to have established real-time feature required for data acquisitions.[1] The high performance of RISC processor, however, may make it possible to realize the "quasi-real-time" performance on UNIX system. Furthermore, the direct link between the data bus of front-end electronics and the high speed extension bus of RISC processor, for example SBus of SPARC or TURBOchannel™ of DECstation™ , will provide high performance of the data transfer rate up to more than 10 Mbyte/sec. It will open a new stage of ultra high speed data acquisition.

UNIX Operating System (OS) has two process regions; one is the User mode region in which users can access and execute processes, and the other is the Kernel mode region in which only process for the system management is the executed by the OS. "System call"s are the only way for users to execute a process in the Kernel mode. A process in the Kernel mode is assigned higher priority than that in the User mode. In UNIX, the priority of a process is not fixed. It is automatically changed by UNIX OS depending on the CPU time consumption. For example, a process that has spent a lot of CPU time moves to the waiting queue. One cannot control the priority of the processes. This is a serious problem

for a data acquisition system where the data handling process should always be kept at the highest priority. The only process that is executed in the Kernel mode can be kept at higher priority than those in the User mode and thus prevented itself from waiting. It naturally leads to the development of device drivers, because one can put the process in the Kernel mode.

Handling the interruption in the UNIX OS is another crucial problem in realizing the real-time feature on UNIX. An interruption starts the interrupt handler in the Kernel mode. Then, the handler wakes up and activates the user routine. This sequence depends on when the interrupt happened. There are two cases in Fig.1. When a process is being executed in the User mode, the interrupt handler of the Kernel mode is immediately executed. On the contrary, when a process is executed in the Kernel mode, the interrupt handler must wait for finishing the current Kernel process. In the latter case, the latency of the interruption depends on the execution time of the running process in the Kernel mode. A "quasi-real-time" feature can be realized on UNIX if the length of this duration is kept within the allowable limit.

In this paper, we present the development and performance of our real-time data acquisition system on SUN SPARC RISC/UNIX workstation (SPARC station IPC 15.8 Mips) connecting to CAMAC, where VMEbus mediates between SBus of SUN and CAMAC. The relevant software utilities were also developed. They provide a convenient environment in applying the high performance of RISC/UNIX to data acquisition. They consist of CAMAC device driver, I/O process, monitoring process, and man-machine interface on the X Window System^{TM} .

2 System Setup

Figure 2 shows the block diagram of our data acquisition system. Our portable data acquisition system consists of a RISC/UNIX workstation (SUN SPARC IPC), a VMEbus extender and an interfaces between VME and front-end electronics system like CAMAC. In this system, SUN accesses to VME through SBus via VME extender "SFVME-100" of Solflower Co. Ltd. SFVME-100 provides a lot of useful functions; (1) VMEbus Master for accessing VME address space mapped to SUN, (2) SBus DVMA (Direct Virtual Memory Access) Master for 20 Mbyte/sec fast data transfer, and (3) software compatibility with standard SUN4 device driver.[2] Kinetic model 2917 is a VME to CAMAC interface, which provides the VMEbus master function.[3] A combined system of the RISC/UNIX workstation and VMEbus makes it possible not only to connect to many front-end systems like CAMAC, FASTBUS, and GPIB through VME interface but also to do fast data transfer between other computer systems through optical links which are in general developed on VMEbus (Fig.3). We can make the DMA transfer possible between SPARC memory and CAMAC Crate Controller (K3922) by assigning K2917 and SFVME as a VMEbus master and a VMEbus slave/SBus DVMA master respectively. CAMAC data are transferred into the memory space of the process in the User mode on SUN without intermediate software. The transfer rate is limited only by the ability of hardware in this case.

A device driver for CAMAC was developed on SPARC in order to preserve the priority of the CAMAC handling process. The ability of fast interruption was also realized by this architecture. Using this device driver on the basis, a library of CAMAC access functions CAMLIB was made. The CAMLIB makes it easy to fully utilize the CAMAC device driver. Table 1 shows some CAMLIB functions. The CAMLIB was written in the same style as that developed by the KEK on-line group for VAX/VMS, VME/OS-9 and PC98/MS-DOS^{TM} for the software compatibility.[4] One common feature of those CAMLIBs is the

SYSTEM CALL

Function	Description
CAMOPN	Open CAMAC device driver
CAMCLS	Close CAMAC device driver
CSETCR	Set crate number
CGENZ	Generate CAMAC Z
CAMAC	CAMAC single action with a 32-bit word transfer
CAMACW	CAMAC single action with a 16-bit word transfer
CDMAL	CAMAC block transfer action with 32-bit words
CDMAW	CAMAC block transfer action with 16-bit words
CWLAM	Wait CAMAC LAM interrupt
CEWAI	Execute CAMAC list in device driver and wait its termination

LIST PROCESSING

Function	Description
READ	Read CAMAC data with single action
WRITE	Write data to CAMAC module with single action
NDT	Execute CAMAC single action with no data transfer
IGQ	Execute Ignore-Q CAMAC block transfer
WAITEVENT	Wait CAMAC LAM interrupt

Table 1: Some CAMAC functions in CAMLIB on Sun SPARC. These functions have good compatibility with those on μVAX II/VMS.

SYSTEM CALL

Function	Unit	SPARC IPC	μVAX II/VMS
CAMACW read	μsec	180	1100
CDMAW overhead	μsec	1160	2000
CDMAW read	Kbyte/sec	690	800
CDMAW write	Kbyte/sec	790	
CWLAM	μsec	230	1000
CEWAI overhead	μsec	190	3200

LIST PROCESSING

Function	Unit	SPARC IPC	μVAX II/VMS
READ (to user space)	μsec	36	158
NDT	μsec	21	26
IGQ overhead	μsec	750	400
IGQ read	Kbyte/sec	690	800
IGQ read (DB)	Kbyte/sec	960	1200
WAITEVENT	μsec	180	160-190

Table 2: Performance of CAMLIB functions.

list processing, which was developed to reduce the non-negligible overhead for calling the device drivers. Figure 4(a) schematically shows the CPU transaction during the sequence of single CAMAC actions. For every CAMAC action the processing comes back from the device driver (system call) to the user level. A single CAMAC action corresponds to one system call. Thus the overhead for system call is required at every CAMAC action resulting a considerable overhead for the data acquisition. It is noteworthy that the overhead of one system call for VAX/VMS takes about 1 msec.[4] Using the list processing, a sequence of CAMAC actions in one list processing can be transacted in one system call as indicated in Fig.4(b). One list processing corresponds to one system call. As far as fast data handling is concerned, the list processing is definitely effective in decreasing the overhead. Furthermore, the list processing has another advantage in the case of the RISC/UNIX. It keeps the priority of the data handling process highest during the transaction of the CAMAC sequence.

All libraries for CAMAC functions can be used in both FORTRAN and C. It is easy to transplant these software to other UNIX workstation, because they are all written in C.

3 Performance

The performance of the present data acquisition system has been examined and compared to that of μVAX II/VMS data acquisition system. The later was developed at KEK and used in a lot of high energy experiments.[4] Table 2 shows the summary of the performance of typical CAMAC functions and list processing functions on SPARC IPC as well as on μVAX II/VMS. In this section, we give a detailed account of these measurements. Following measurements were carried out under the simultaneous operation of network management process and window system.

SYSTEM CALL

Function	Data Counts	Mean Execution Time (μsec)	R.M.S. (μsec)
CAMAC read	2000	179	22
CAMAC write	2000	178	31
CAMACW read	2000	178	20
CAMACW write	2000	179	23
CAMACW ndt	2000	171	29

LIST PROCESSING

Function	Data Counts	Mean Execution Time (μsec)	R.M.S. (μsec)
READ	2000	28.6	4.3
WRITE	2000	28.8	3.9
NDT	2000	20.9	3.4

Table 3: Performance of CAMAC single action functions.

The measured execution time for typical CAMAC single action functions are presented in Table 3. All CAMAC single actions in the list processing were observed to be much faster than those in the system call. The overhead was measured to be about 150 μsec for each system call. It is, however, 7 times faster than that of μVAX/VMS, which is 1 msec. The list processing has faster mean execution time and narrower R.M.S than the system call. Figure 5 shows the distributions of the execution time for both CAMAC word read/write function (CAMACW) in the system call and CAMAC read function in the list processing (READ). The execution time of READ is concentrated between 25 to 35 μsec. Less than 1% of READ executions is found to be delayed as much as 50 μsec. This long delay is considered to be caused by the Kernel processing like the clock interruption. The narrowness of R.M.S. of list processing is considered to be because of the high process priority of the Kernel mode.

In Table 4, the performance of the block transfer functions is given, where list processing provides only CAMAC read functions. IGQ (IGnore Q-response) was achieved to be 0.96 Mbyte/sec. This performance is close to the limit of the hardware ability of K2917 (1.14Mbyte/sec). Figure 6 indicates the good linearity between the transfer time and the data size. CAMAC crate controller K3922 has two data transfer mode. The transfer speed of IGQ in double buffer mode was 1.4 times faster than that in non-double buffer mode .

The latency of interruption generated by CAMAC LAM request is shown in Fig. 7. The latency of the WAITEVENT function in the list processing was measured to be 180 μsec on average with R.M.S. of 23 μsec. More than 99.9% of the interruption was found to be generated within 500 μsec.

The ability of the data acquisition under various environments of background processes is another thing to be studied. The execution time of the CAMLIB function may depend on the processes which are executed simultaneous to the CAMLIB function. We measured the CAMLIB performance on simultaneous execution of two kinds of processes. One is a CPU consuming process (CPU-CONSUMER). The other is I/O consuming process (I/O-CONSUMER).

SYSTEM CALL

Function	Data Counts	Overhead (μsec)	Transfer Time ($\mu sec/cycle$)	Transfer Speed ($Kbyte/sec$)
CDMAL write	899	1023	5.47	548
CDMAW write	899	1044	2.52	794
CDMAL read	899	978	4.67	642
CDMAW read	899	1160	2.88	694

LIST PROCESSING

Function	Data Counts	Overhead (μsec)	Transfer Time ($\mu sec/cycle$)	Transfer Speed ($Kbyte/sec$)
IGQ read	899	746	2.91	687
IGQ read (DB)	899	736	2.09	957

DB: Double Buffer mode of CAMAC Crate Controller K3922

Table 4: Performance of CAMAC block transfer functions.

The CAMAC block transfer time was examined on the various environments (Figs. 8). The transfer time was linear to the data size on the environment with no background processes. Figs. 8(a),(c),(d) show the result with two I/O-CONSUMERs. Fig.8(b) shows that with three CPU-CONSUMERs. Figs. 8(a),(b),(c) show the performance of the CAMAC block transfer in system call. Fig.8(d) shows that of the list processing. It was noticed that IO-CONSUMERs made crucial effects on the CAMAC block transfer system call functions. Some of data transfers are delayed by the CPU-CONSUMER. This unfavorable effect, however, was avoided by using the block transfer in list processing as shown in Fig.8(d) or decreased by means of assigning high priority to the block transfer process ("nice" system call) as shown in Fig.8(c).

The latency of the LAM interruption was examined on those environments. UNIX does not guarantee the maximum latency of the interruption. Figure 9 shows the response time of the list processing function measured under three CPU-CONSUMERs (a) or two I/O-CONSUMERs (b). From Fig.9(b), it is found that I/O-CONSUMERs cause a considerable latency, even though we use list processing functions in the Kernel mode. However, such an extreme condition may not happen in the realistic data acquisition.

4 Portable Data Acquisition System (PDAQ)

Using the CAMAC device driver, a portable data acquisition system (PDAQ) including utilities has been developed on SunOS 4.1.2. The PDAQ is divided into two parts, the "User program" and "Core program", as shown in Fig.10. The "Core program" is an ensemble of processes for the management of data handling with higher priority assignment. It includes CAMAC device driver, "Collector", "Recorder", and "Manager", in which only CAMAC device driver is executed in the Kernel mode. In this system, "Core program" looks like a black box for users.

The "User program" is an ensemble of processes for monitoring data and man-machine interface, called

Data Size (word / event)	10	100	1000	2000
Max. Acquisition rate (Hz)	360	300	120	60

Table 5: Performance of PDAQ.

"Monitor" and "Panel" respectively. Data transfer between CAMAC and SUN is done by "Collector". "Recorder" writes data on the disk or other mass storage device like an 8 mm tape. Data transfer between processes is done through "Data buffer" which is in the UNIX shared memory. The data is at first transferred from CAMAC to the "Data buffer" by "Collector". Then "Recorder" writes the data on "Data buffer" to a mass storage. "Monitor" is allowed to analyze data on "Data buffer". "Panel" is written by using OpenLook™ utilities. We can control the data acquisition processes by clicking the button on "Panel". Almost all the utilities supported by both SunOS and X11 Window System can be used with the PDAQ simultaneously. When users develop on-line programs on this system, both C and FORTRAN languages are supported.

We measured the total performance of the PDAQ system. The experimental data were taken from CAMAC with the list processing and saved to a hard disk. Table 5 gives the maximum acquisition rate measured with the PDAQ. The performance indicates that the PDAQ has enough power to be adopted for various small experiments. The PDAQ system is not fully optimized in the present stage. It seems to be possible to achieve higher rate data acquisition with SPARC IPC.

We used the PDAQ for a beam test of CsI calorimeter being developed for the Japanese B factory project. No serious trouble occurred during the continuous running of two weeks. In parallel with the data acquisition, high level analysis was done by using PAW. Although PAW requires quite large memory area, no noticeable effect on the data handling was observed. PAW is not an I/O consuming process. Therefore, it does not affect the data acquisition so much.

Network capability of UNIX operating system makes it possible to add more CPU power to the data analysis. Another RISC/UNIX workstation was used to analyze experimental data simultaneously by means of NFS™ (Network File System).

5 Conclusion

We have developed a portable data acquisition system using RISC/UNIX workstation. It enables us to use both the high performance of RISC processor and the good utilities of UNIX. Comparing with μVAX II/VMS, higher performance of data acquisition was achieved by a very compact and inexpensive SUN workstation combined with SBus-VME extender. The system is as small as a personal computer. In particular, PAW can be used in parallel with the data acquisition process on a small workstation.

The quasi-real-time data acquisition on UNIX OS (SunOS) was realized by developing a device driver which is processed in the Kernel mode. This method was found to provide the good performance of response time of interruption and execution time of data transfer on RISC/UNIX workstation.

The measurements of the basic performance verified that an online system based on RISC/UNIX will not only satisfy above requirements but also open a new stage in global data acquisition. For instance, UNIX provides a good environment of network so that the boundary between on-line and off-

line may vanish. Hierarchy of the on-line processors may possibly disappear, since the portability and inexpensiveness of RISC/UNIX workstation will enable us to use the same software and hardware from small test bench experiments to large collider experiments.

6 Acknowledgement

We are grateful to KEK on-line group for their continuous support and the presentation of the data of VAX/VMS system. In particular, we thank Dr.Nomachi for his fruitful discussion and cordial encouragement. This work was partially supported by the Tokyo Institute of Technology Research Fund and by the Grant-in-Aid For Scientific Research by the Ministry of Education, Culture, and Science, Japan.

References

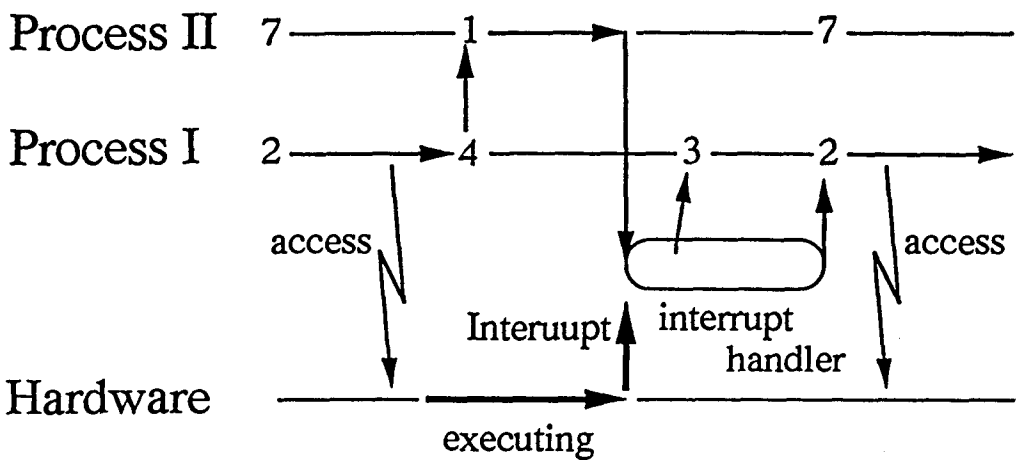
[1] M.Nomachi et al., Proceedings of the "Computing in High Energy Physics 1991", edited by Y.Watase and F.Abe, Universal Academy Press, p681 (1991)

[2] "SFVME SERIES USER'S MANUAL and INSTALLATION GUIDE", Solflower Computer, Inc.

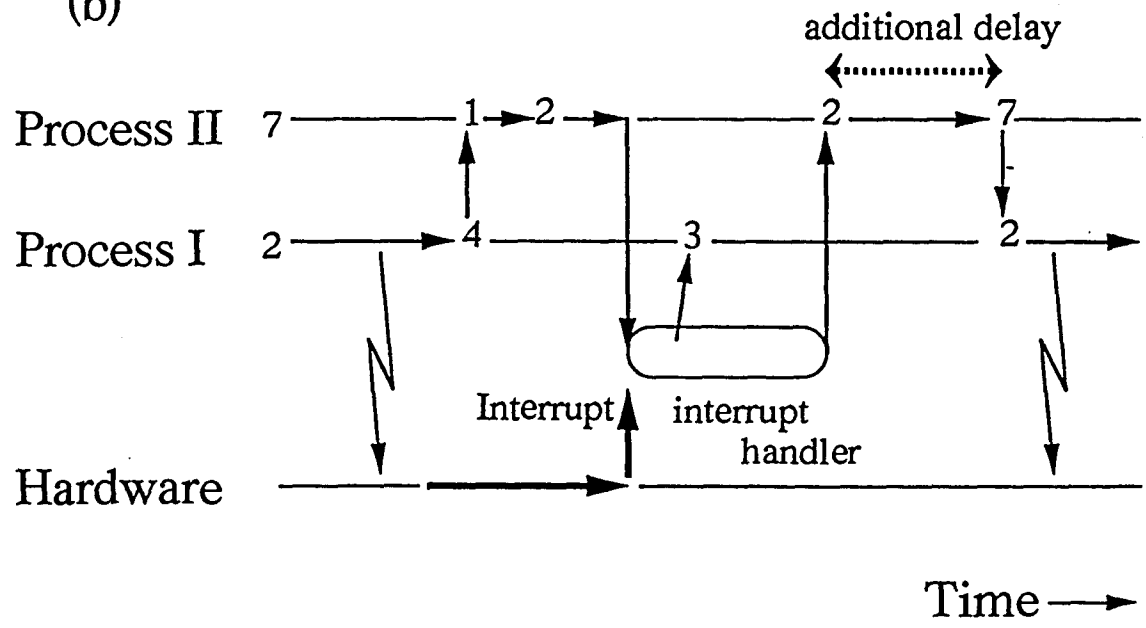
[3] "Model 2917-Z1A VMEbus Interface w/DMA INSTRUCTION MANUAL", Kinetic Systems, Inc.

[4] Y.Yasu and M.Nomachi, KEK internal 90-2 (1990)

(a)



(b)



Process status
1: executing in User mode
2: executing in Kernel mode
3, 7: waiting
4: sleeping

Figure 1:

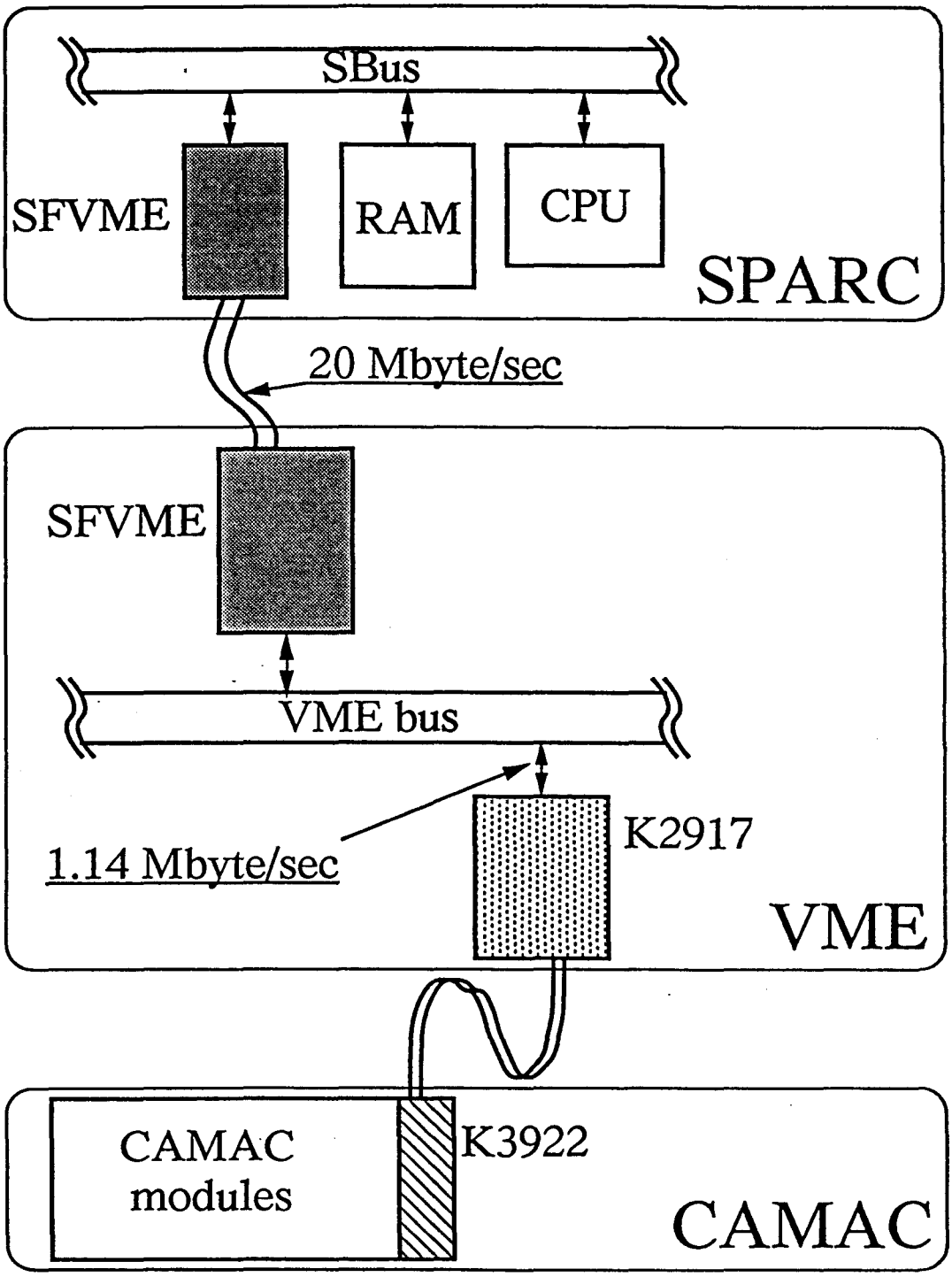

SBus
SFVME
RAM
CPU
SPARC
20 Mbyte/sec
SFVME
VME bus
1.14 Mbyte/sec
K2917
VME
CAMAC
modules
K3922
CAMAC

Figure 2:

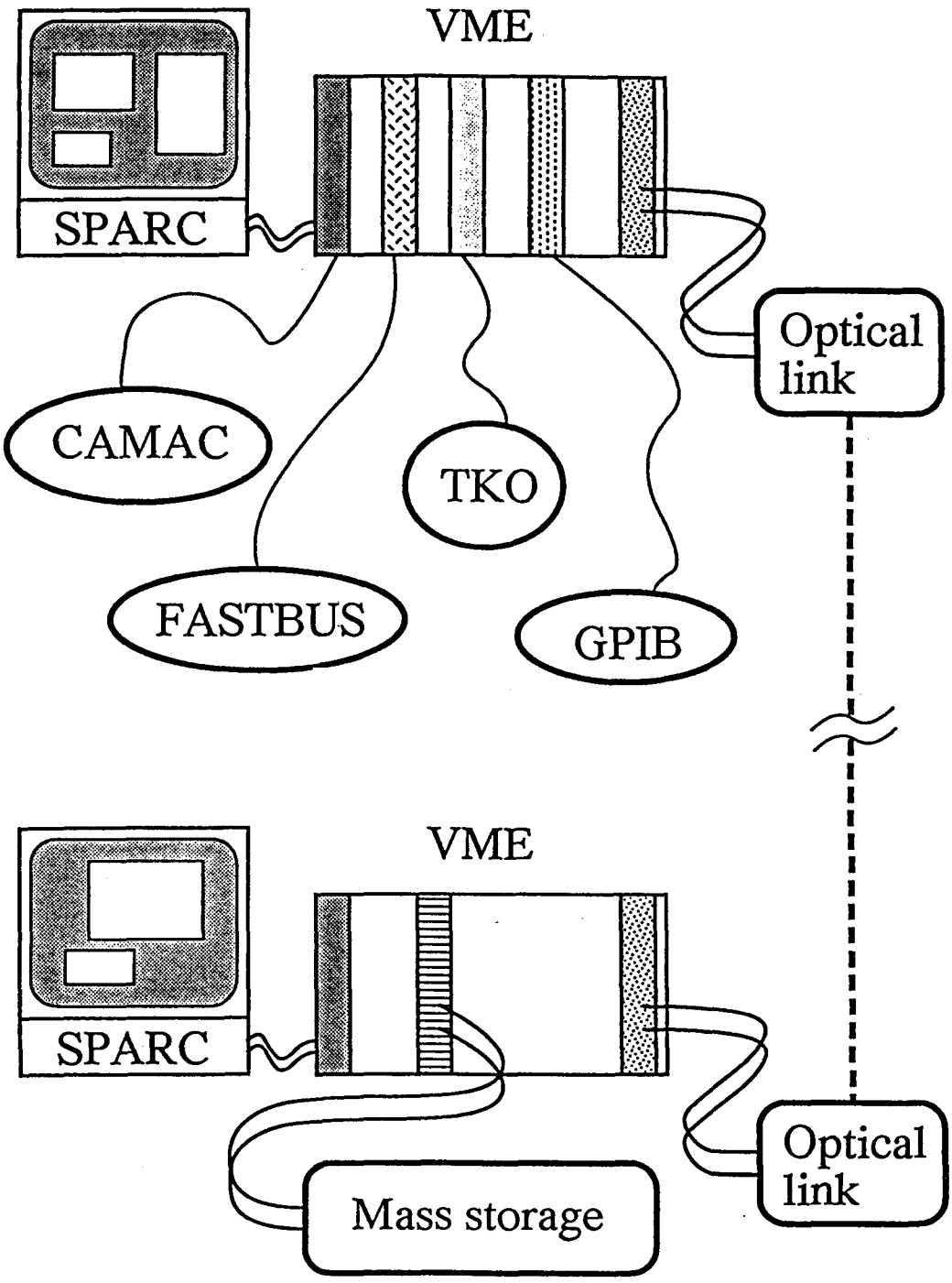
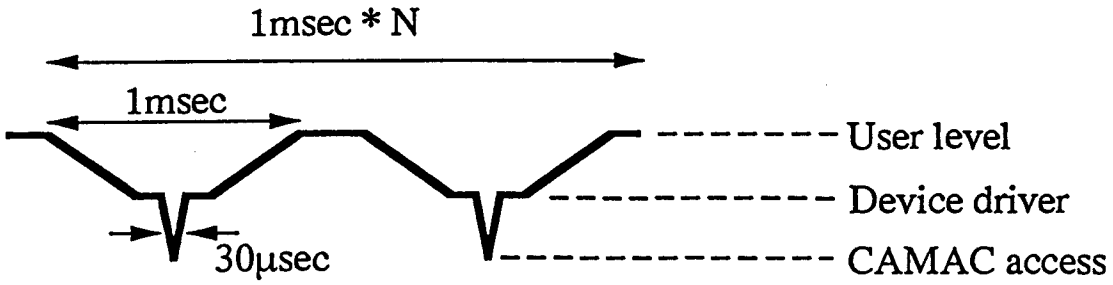
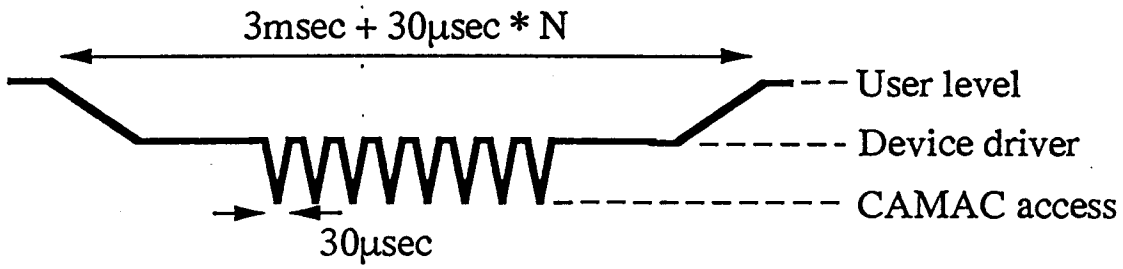

VME
SPARC
Optical link
CAMAC
TKO
FASTBUS
GPIB
VME
SPARC
Optical link
Mass storage

Figure 3:

(a) SYSTEM CALL



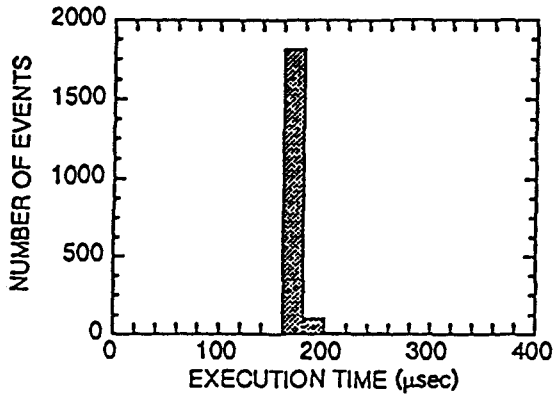
(b) LIST PROCESSING



N: number of CAMAC accesses

Figure 4:

(a) CAMACW



(b) READ

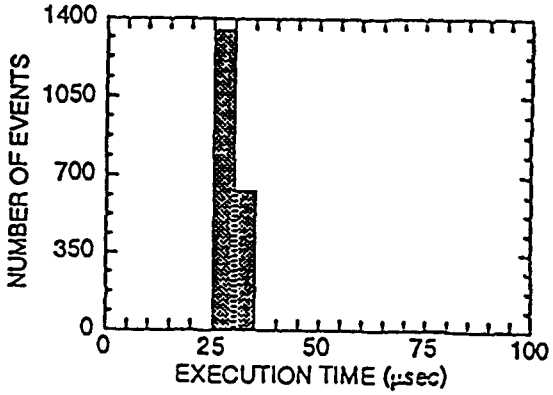
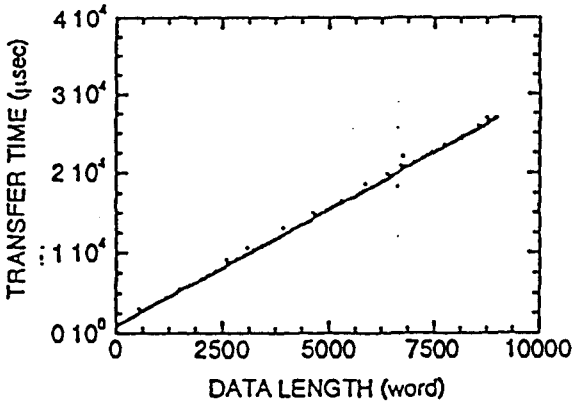


Figure 5:

(a) CDMAW



(b) IGQ (double buffer mode)

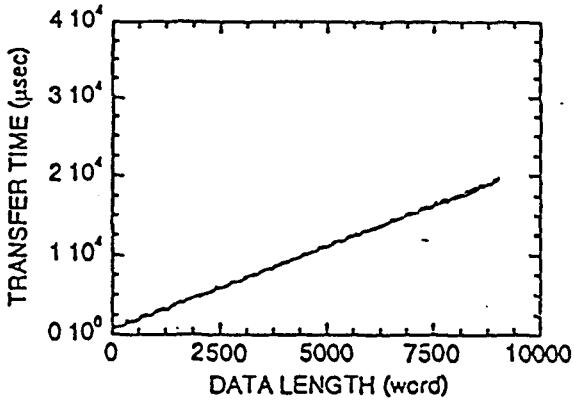
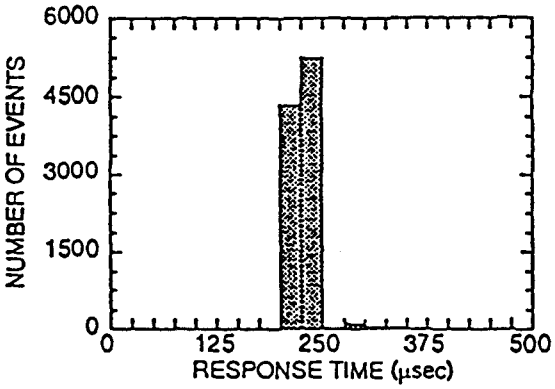


Figure 6:

(a) CWLAM



(b) WAITEVENT

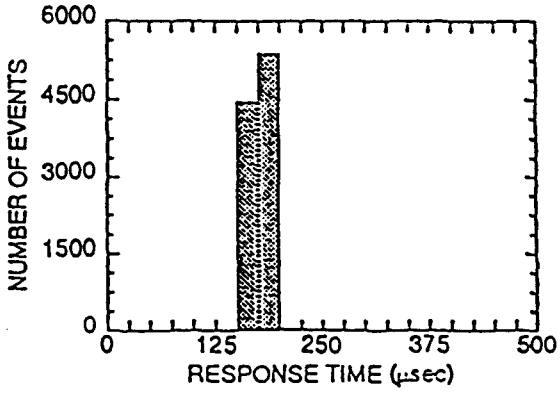


Figure 7:

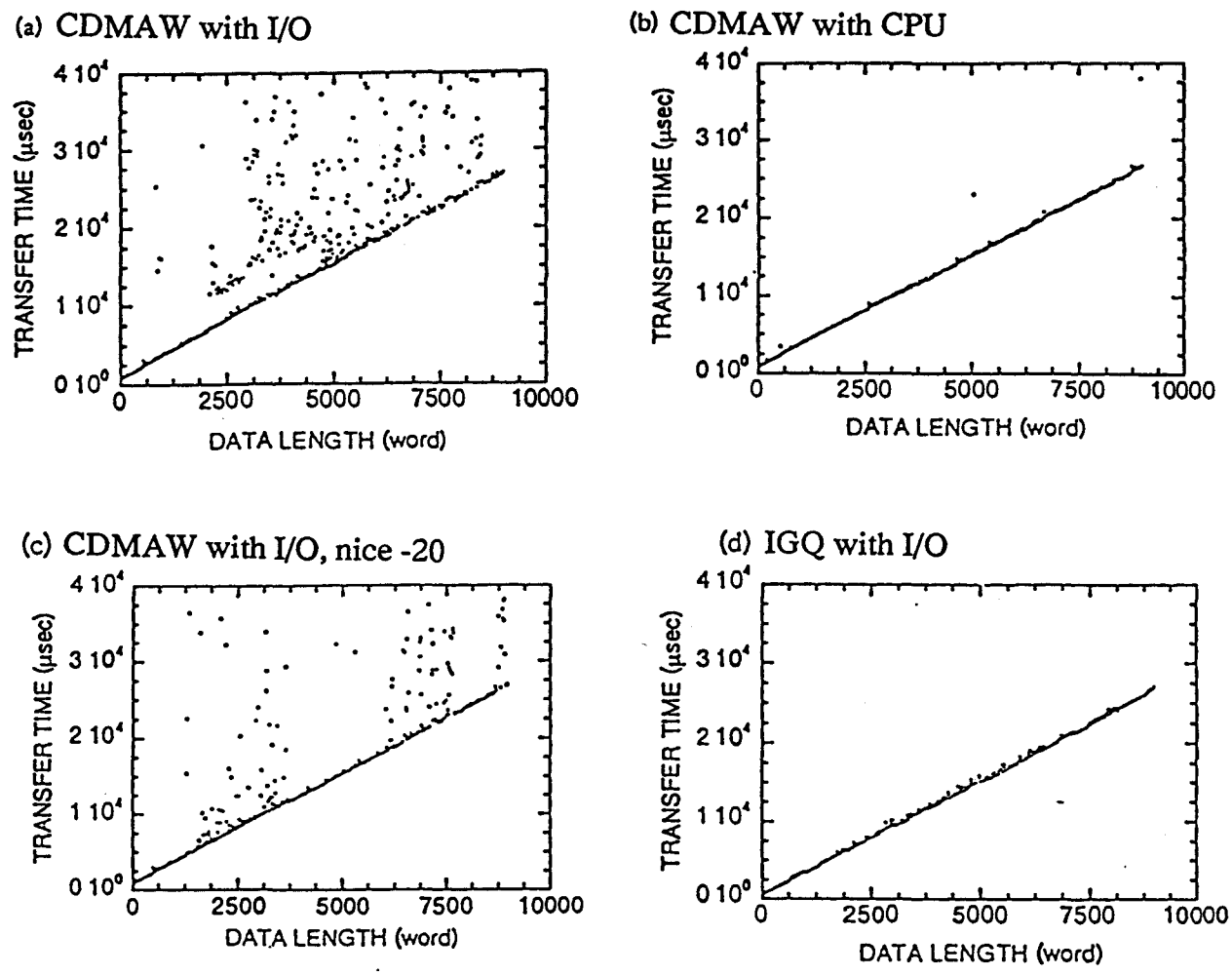


Figure 8:

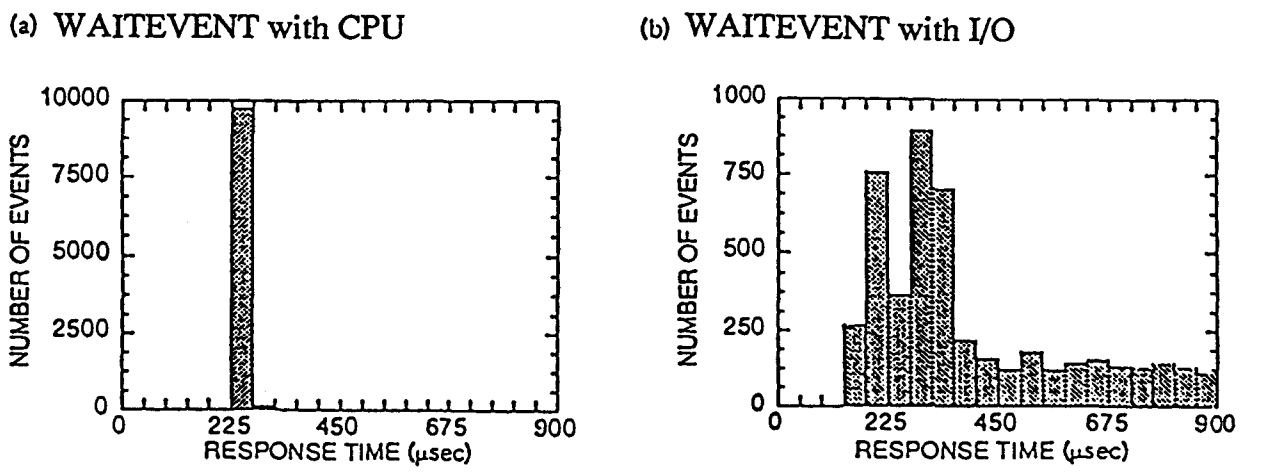


Figure 9:

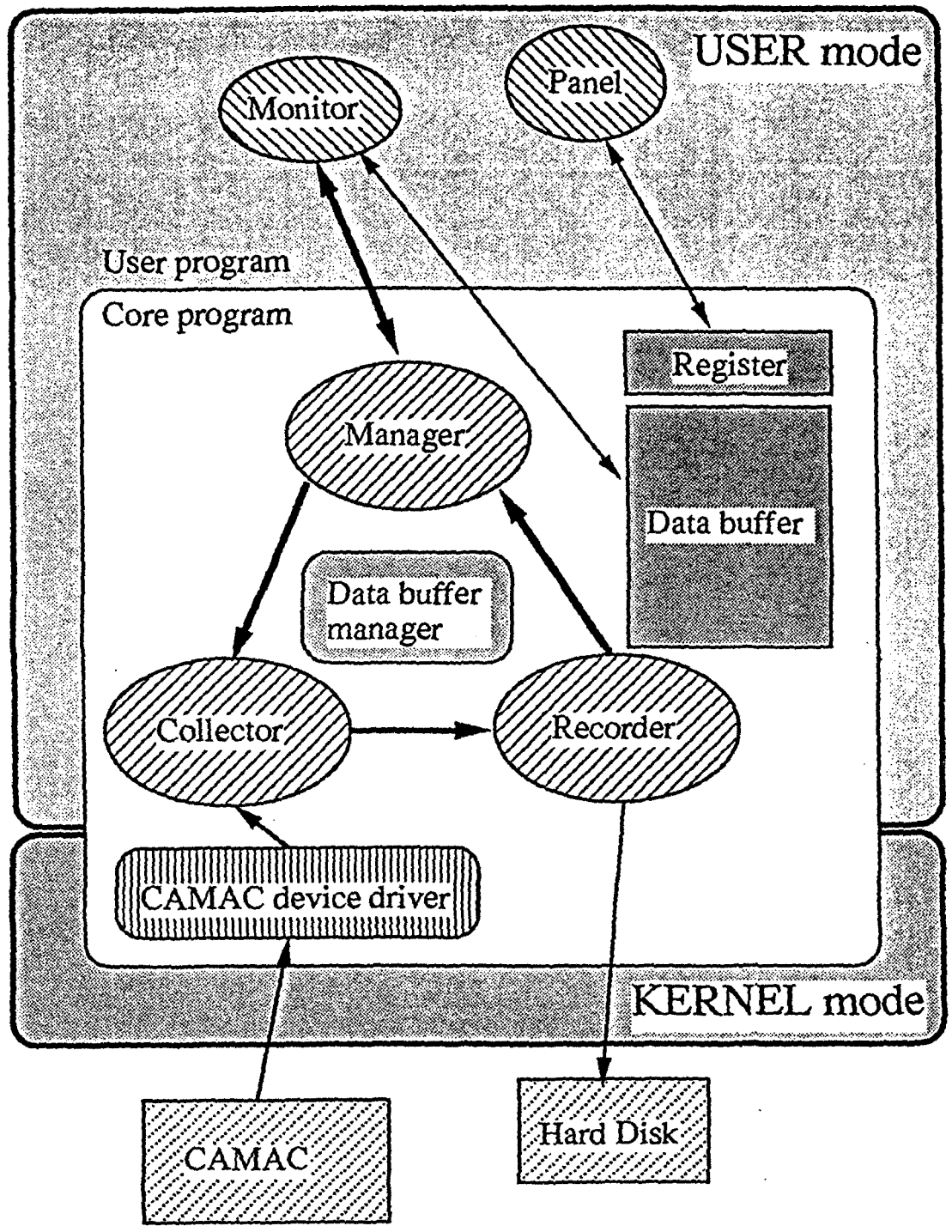

USER mode
Monitor
Panel
User program
Core program
Register
Manager
Data buffer
Data buffer manager
Collector
Recorder
CAMAC device driver
KERNEL mode
CAMAC
Hard Disk

Figure 10: