A SUBSYSTEM THAT IMPROVES RESPONSE TIME FOR APPLICATIONS WITH LARGE NUMBERS OF TERMINALS

RALPH HENNEN and BOYD CARLSON

OHIO STATE UNIVERSITY RESEARCH FOUNDATION

and

SYSTEM RESOURCES GROUP, INC.

INTRODUCTION

Time-sharing has always plagued users with poor system response time and relatively high expense, though systems eventually evolved which alleviated both conditions. Chip processors and chip memory added to the attraction of time-sharing and it began to develop as a system rather than an option to a batch system.

As the systems became more sophisticated, however, so did the applications which were run on those systems. Because of this, time-sharing systems frequently frustrated the user.

Distributed systems, too, offered a breakthrough for response times in large systems. They allowed more users to access stored information without excessive response time. Their drawback was that they only applied to a large data volume/low transaction environment.

So it seems that there is a third need for improved service in the system-and-user market: frequent transactions with associated low volume of data and processing. Though both the time-sharing and the distributive systems are overelaborate for this need, their response to this type of application can still be extremely sluggish.

Applications which belong to this class of fast transaction-low data are: plant production control, computer auctioning of goods, security monitoring systems, data entry, data inquiry, and monitor control. One such design is currently in production on the HP3000.

THE NEED

This design is the result of a research project which required a CPU to service, with a response time of two seconds or better, at least 60 terminals. The project budget complicated the order; it would not allow for a large, main frame computer. The salesmen doubted that it was possible to accomplish a task of this scope, and the hardware search increased their doubts. But the HP3000 turned out to have the system tools we needed to develop the required system. Much of the system design work had been done on what was known as a "run-time system."

This system was to be used for another almost unheard-of application: the electronic market. An electronic market facilitates trading between buyers and sellers over the terminal using the CPU as auctioneer, bookkeeper, banker, statistician, and marketing information source. Several markets trade simultaneously. Other users can do other things during trading, such as getting detailed information on the goods to be sold, getting price information for goods already sold, or even looking at the regional weather forecast.

The user audience in this case was to be diverse. There was no guarantee that all terminals which might potentially talk to the system would have identical characteristics. Not only were authorized buyers and sellers to use the system, but several news services were interested in tapping into the marketing information for their customers.

Communicating with the users was not going to be a simple task. We had little control over users who already had a communication network and who simply wanted to connect to our system. Our communications would have to accommodate the Bell System as well as several other vendors. We were to serve both leased line and dial-up users. Some of the users were on point-to-point lines and asynchronous multipoint lines.

As information flowed into our offices regarding what we needed to do to bring the users onto the system, it became obvious that we needed all the tools we could find to make the system work. The puzzle became larger and more difficult daily.

THE RESEARCH

The system design task for the run-time system took about five months to accomplish. During that period, we evaluated five different systems with regard to their potential efficiency, programmability, maintainability, and practicality. In time we developed a design that seemed potentially compatible with the HP3000 hardware and some of MPE III.

We designed the system with structured and modular programming techniques. The modules were separated into two groups: system and application. The application modules solved a particular problem between user and data; the system modules connected the hardware to the user. The communication modules were coded to be tested apart from the rest of the subsystem. This separation gave them adaptability in case we had to change them or add other codes. Many field situations required considerable tuning of the communication modules.

The true application programs were lowest in the hierarchy of this subsystem because of convenience. The sponsor who had the final word on how the application programs would run did not always agree with us about how they should work. Often the sponsor could not foresee future needs to an extent that would allow us to write definitive specifications for the application programs. Fortunately, the application modules had little bearing on the system code.

After we attached the application programs to the completed subsystem, we began testing and tuning. With alteration of some subsystem parameters and changing of some MPE III configuration parameters, the run-time system performed much better than our team or anyone else had expected. For our application, the response time was better than 1.5 seconds. We inserted pauses in the code to slow the execution so the users could expect uniformity of response and to let them get used to the cadence of activities at the terminal.

THE SYSTEM

The system is a collection of processes which are run under MPE III. These processes are organized to communicate data and to synchronize

connections between many users and many application programs, as shown in Figure 1. Much of the work which would be done by MPE in the time-sharing environment is managed under the subsystem. The disadvantage of this structure is that users do not have the use of the system hardware that they would have in a time-sharing environment. To the contrary, the hardware resources are transparent to the user on the subsystem; they are available through the applications level. This organization is an attempt to relieve the overhead of managing the system resources to control the I/O to the user.

This subsystem minimizes the movement of data throughout the processes. Data bound for system I/O to the user are stored in a memory buffer. The characteristics of the data are passed through the subsystem and the data are accessed only at the point where the application must use them. In the same way, data produced by the application program are placed in a buffer; they are not moved until they leave the system.

Figure 2 illustrates the design of the subsystem, showing it between the MPE environment and the application environment. Figure 3 illustrates in more detail the core processes and programs of the subsystem. The system is loaded and executed at a terminal which becomes the run-time console. This console is a session under MPE; it receives information from both the run-time system and MPE. The console gives the operator control over the rest of the subsystem. The operator may create initial conditions of the system. These conditions may vary, depending upon the application environment the subsystem is serving. The operator can also monitor certain aspects of the system and intervene when necessary.

The I/O Controller handles all the communication activities and problems. This module works closely with the I/O driver provided by MPE to intercept the I/O from external devices. It handles the communication protocol information that the drivers pass to the subsystem and checks errors on the communication lines. The I/O Controller regards the remote I/O device as the primary initiator of the input. It intercepts input as it comes; when the input is complete, the data are accounted for and ready for processing. The I/O Controller then sends a message to the next process on the system. That message contains all the characteristics of the data and in most cases also indicates which application program should be used to process the data. In much the same way, the application program then returns data through the system and back to the terminal.

The I/O Controller controls several resources within the subsystem. It initializes and maintains tables that track activity within the system and maintains and keeps records of all I/O activities on all devices. All information collected by the I/O Controller can be made available throughout the subsystem.

The I/O Controller distinguishes between clock-dependent activities and those which are not clock-dependent. Clock-dependent activities may require a specified delay between request and response. These activities have the highest priority for response in the system. The attached application programs must then run faster than user-

specified response time. Other clock-dependent transactions receive data at the remote device without solicitation. These transactions include countdown information, timer processes, pulse bursts, and monitor requests. Activities which are not clock-dependent pass through the other part of the system and may include application programs with variable execution times between user requests.

These two classes of activities have separate Interrupt Processors. Both processors queue requests that may use the same system resource or application program. They wake up the application program appropriate to the requests and issue errors for bad syntax and unknown requests. All usable applications for each generation on the system are known to the processors. It is necessary to reconstruct the system before changes in the available programs can be attached.

Generally, each terminal or remote device request will stimulate a response from some application level program. This transaction will cause a complete pass through the system (down and back) and one pass through the application program. In some instances, the application program level may be in a conversation-like mode with the user device or terminal: for example, when the user must choose among the items of a menu produced at the terminal. The Interrupt Processor must then remember which terminals are in that mode, and the application program must find the position in the code to which the transaction must return. This position is stored and becomes part of the communication structure between the interrupt process level and the application level. The application program may tell the Interrupt Processors that the transaction is incomplete and indicate the code location to begin execution upon further input from the device. This allows the application program to process another request while the user (or user device) is deciding which option to take. Should the user choose not to continue the chain of activity to the particular application program, the communication between the I/O Controller and the Interrupt Processors will flag that case. All pending activity for the device is then cancelled and the new request becomes a new transaction. This type of communication allows the subsystem to be somewhat transparent to the user and gives the user interactive capability within the program.

Other small system programs do not appear in these generalized diagrams. They manage some of the common resources in the run-time environment. These resources can be thought of as highly dependent or shared resources in the subsystem. Their use dictates the results of some transactions. For example, in the situation of the electronic market, each item offered to the market is posted on a market screen. The transaction that results in an offer is completely separate from those which play the market, though the two activities are tied together by the market screen. The market screen must have a governor to ensure that all offered items are advertised to all marketeers, each of whom is interacting with the market screen uniquely. These management processes tend to change in character and number as the run-time environment dictates; resource

management for an electronic market would not be the same as for a production control run-time system, for example, or a security monitoring system.

THE INSTALLATION

This run-time system can be thought of as an application system. Since the application environment is usually tailored to a specific user audience, run-time systems vary. Each system is designed to be installed for each new application environment. The system is modular, and, though a few core system programs occur in all instances, the remaining peripheral software can be installed by trained personnel to accommodate the application environment. Not all applications can take advantage of this type of subsystem; programs which massage a large volume of data or perform large calculations with few user transactions will receive little value from it.

Existing needs and software should be evaluated regarding their compatibility with the subsystem before installation, but ideally the application programs should be written with the subsystem in mind.

Many of the rules for efficient stand-alone programs apply to this environment. Code written with some consideration of segmentation, memory allocation, and possible thrashing problems will, of course, run more quickly. Code designed in a structured, top-down fashion with one pass through for each transaction state is ideal. Programs which use data bases efficiently by limiting the number of cross datasets searches will run faster. Single programs which do not try to accommodate every user's needs at once will be at an advantage.

The subsystem offers a host of tools which can be used successfully at the application program level. The library includes I/O statements for user device accessing, error diagnostics to test for errors in the operations, error file operations for user-defined errors, status checking for the transaction condition codes and data location, and others.

These subsystem monitors also provide tools for the system environment. Files can be allocated to save the monitor information for inspection at a later time. This log of system activity can be useful for tuning the system.

A special consideration at installation time should be the various devices attached to the subsystem. The communication modules in the I/O Controller will have to be selected and installed based on the communication protocol and hardware interface for the device. This will require different considerations for different communication subsystems.

THE PERFORMANCE

The run-time system is in production; it has been performing satisfactorily for over a year with few modifications of the code.

The largest production encompasses 55 terminals. Under normal operation, response time has shown no noticeable degradation, even with increased activity within the run-time system. The main system currently feeds to five time-sharing terminals which are used for program development and editing large data sets. They run simultaneously with the run-time system and 55 terminals. There is no noticeable degradation in response time at either the run-time terminals or the time-sharing terminals, with one exception. There is a slight delay in the time-sharing terminals when the Clock Interrupt Processor is sending countdown information every 1.5 seconds to 40 or more terminals.

Performance data are being collected from inside the run-time system as well as in the time-sharing environment under MPE. Information is being collected in the run-time system as to the average process time for different types of application programs: those which are clock dependent, those accessing external file data sets, those accessing IMAGE data sets, those broadcasting information to more than one receiving terminal. In the time-sharing environment, data are collected as to execution times for a mix of compiles, preps, edits, data entry and access, and streamed jobs. This is all done for the stand-along, time-sharing environment; stand-along, run-time environment; and the mixed environment, for the purpose of comparing the differences among the environments.

Several system configurations under MPE obviously play a significant part in the response time in the run-time system. The parameters are time quantum, max extra data segment size, and max data. These parameters will take on different values for different configurations of the run-time system. These should be monitored and tuned before the run-time system is put into production.

CONCLUSION

Many hybrid systems are appearing in the manufacturer and OEM market. These systems reduce the programming load for the installations who buy the software. It costs less to purchase an accounting system than it does to construct one in-house.

The run-time system provides more efficient use of a CPU for installations serving large numbers of transactions with data volume in standard CRT screen unite (1920 bytes). The added benefit of this subsystem is the ability to communicate to a large number of terminals attached to a single CPU.

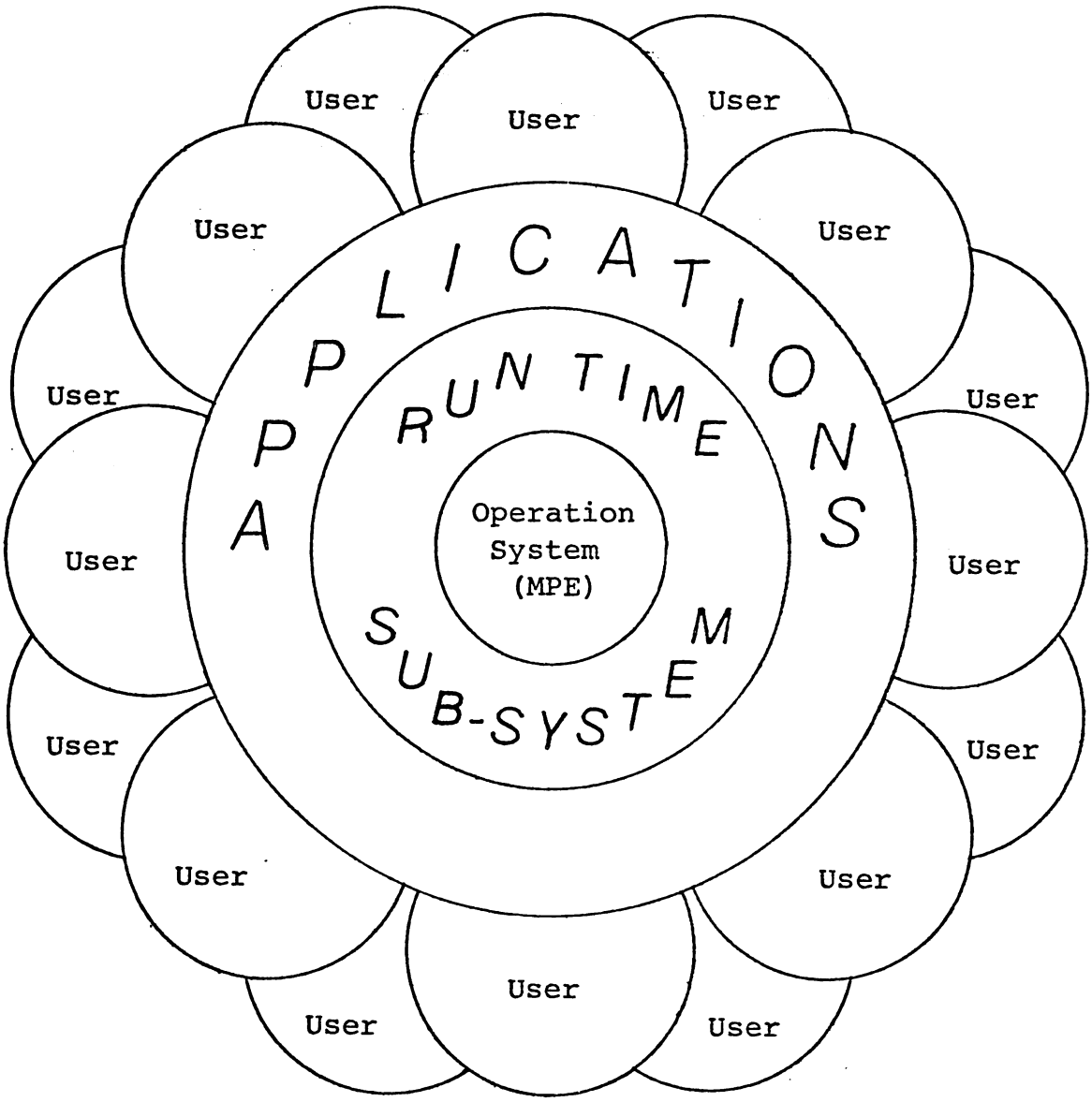
User
User
User
User
User
User
User
User
User
User
User
User
User
User
User
User
User
User
APPLICATION
RUN TIME
SUB-SYSTEM
Operation
System
(MPE)

Figure 1

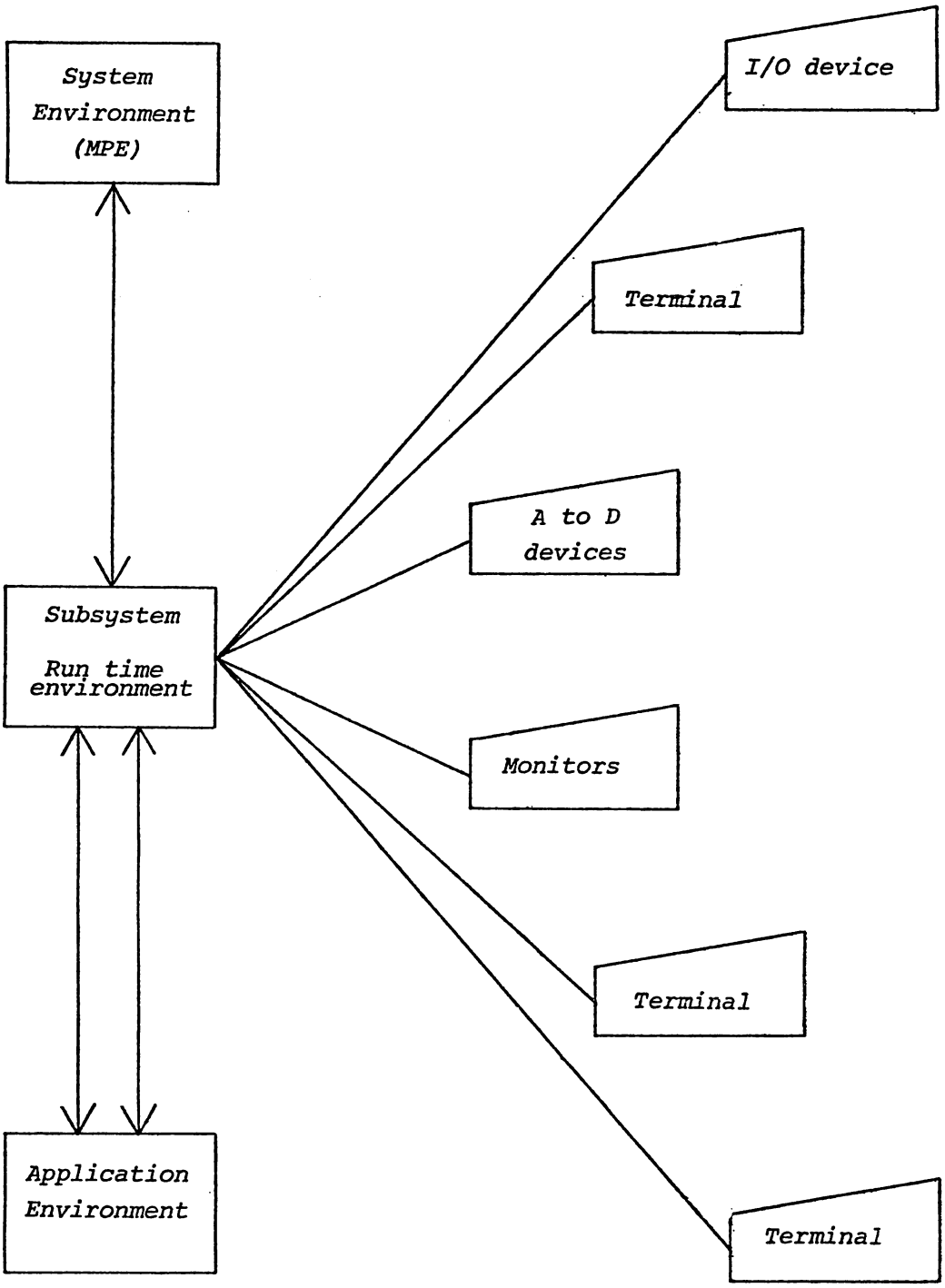
System Environment (MPE)
I/O device
Terminal
A to D devices
Subsystem
Run time environment
Monitors
Terminal
Application Environment
Terminal

Figure 2

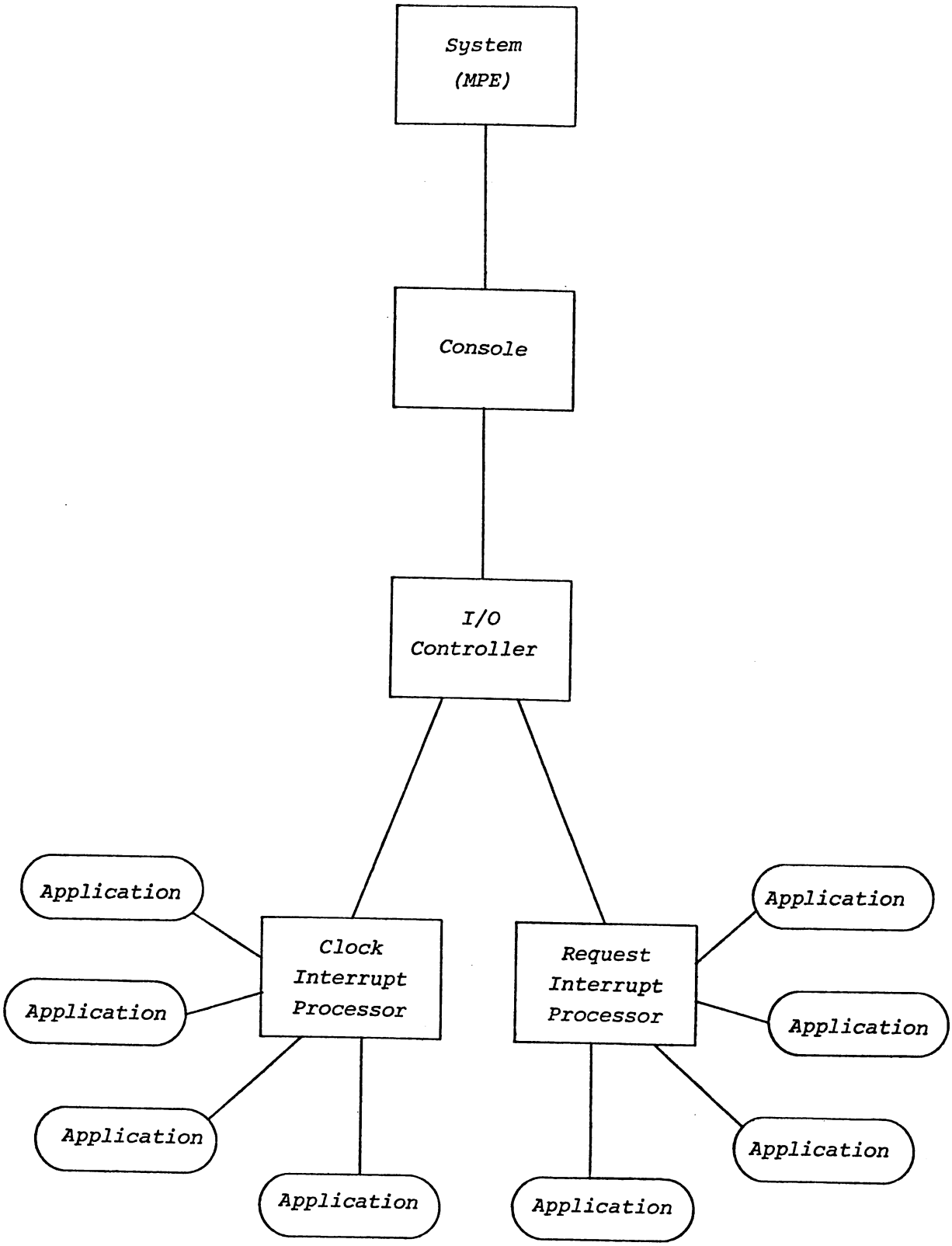
System
(MPE)
Console
I/O
Controller
Application
Application
Application
Clock
Interrupt
Processor
Application
Request
Interrupt
Processor
Application
Application
Application
Application

Figure 3