1986

Analysis of a Class of Real-Time Control Systems

Dan Cristian Marinescu

Report Number:
86-572

ANALYSIS OF A CLASS OF REAL-TIME CONTROL SYSTEMS

Dan Cristian Marinescu

CSD-TR-572
January 1986

ANALYSIS OF A CLASS OF REAL-TIME CONTROL SYSTEMS

Dan Cristian MARINESCU

ABSTRACT

Real-time control systems with a subset of processes subject to deadlines are investigated. The structure and the performance of a class of such systems, namely the data acquisition and analysis systems are analyzed in case of procedure oriented as well as in the case of message oriented design. A methodology for system modeling and the approximations needed are discussed.

INDEX TERMS

Approximate solution, real-time systems, data acquisition and analysis systems, process, performance evaluation, parallel processing, message oriented system.

1. INTRODUCTION

Real-time systems in which all processes must meet strict timing deadlines are called hard real-time systems. Most of the systems of this type are homogeneous, since they consist of a set of processes which execute similar functions upon different input data streams. Another characteristic of this type of system is that the processing associated with each process is relatively non-sophisticated and consequently the system can be implemented either in low level languages or in special real-time languages. Timing analysis through system modeling can be performed without major problems every time the system undergoes significant modifications. Scheduling strategies for this type of system have been investigated in relation to the problem of assigning priorities to different processes, in order to satisfy all deadlines.

This paper discusses a particular class of real-time control systems which are non-homogeneous and combine the real-time requirements of control systems with the complexity and sophistication of knowledge processing systems. Since only selected processes need to satisfy strict timing deadlines these systems will be called semi-hard real-time systems. An important characteristic of this type of system is that they have to be open ended, allowing a user

to add own processing procedures to the system. Also, the system must be implemented in a high level language, since the processing performed can be very sophisticated.

Examples of this type of systems are the data acquisition and analysis systems used in nuclear and high energy physics as well as in other fields where complex experiments produce one or more streams of input data which must be captured and analyzed [6]. The data acquisition system performs a time critical function, the system must be able to process the streams of input raw data at the rate at which they are produced. Its main objective is to store safely all the data captured, with a minimum amount of transformations done to it. The analysis system is a knowledge processing system. Its function is to assist one or more users to control the experiment and to evaluate the results. Access to large data bases of experimental data, theoretical models, model parameters, as well as a wide spectrum of tools to compare different type of objects must be supported. In contrast to other types of real-time systems where all processes must meet predefined deadlines, in a data acquisition and analysis system only the data acquisition function is usually subject to a hard deadline. The data analysis function is user specific and need only process statistical samples of the input data.

A data acquisition and analysis system can be designed either as a procedure oriented system or as a message oriented system depending upon the hardware architecture of the supporting system and upon the environment provided by the host operating system. Most of existing systems are based upon a procedure oriented design and often they support only one analysis. Due to the increased availability of multiprocessor systems a number of design principles need to be reconsidered. Multiple analysis procedures are easily supported since the host system can be expanded. A new possibility is to design the system in such a way that one or more analysis procedures meet the timing constraints and are able to analyze all input data. In this case a considerable reduction of the volume of data recorded and of the effort for post experiment data analysis is expected. This paper argues that the semantics of message oriented system is more natural for

the design of a distributed data acquisition and analysis system and multi-processor systems provide a very good support for this type of design.

An important performance characteristics of a data acquisition and analysis system is the highest data rate the system can sustain during steady-state operation. It is non trivial to determine this parameter in a concurrent processing environment. In case of a procedure oriented design, a number of logical resources need to be defined and even if no contention for physical resources is present in the system, the effect of contention for logical resources must be estimated. For message oriented systems the only synchronization between processes is done via messages. The system may be designed to closely resemble a data flow environment.

Different architectural options as well as the underlaying implementation problems encountered in the design of such systems are analyzed. A methodology for system modeling in order to determine the performance of the system, in particular the highest data rate the system may sustain, is described. A number of approximations are necessary in order to model this type of system to determine its performance. The central theme is the compromise between a well structured system and an efficient one which satisfies the timing requirements of real-time processing.

The two alternatives, procedure oriented and message oriented system design are examined considering both the system structure and its performance.

2. PRIMITIVE FUNCTIONS AND INFORMATION STRUCTURES FOR DATA ACQUISITION AND ANALYSIS SYSTEMS

For many years it has been debated if a general purpose data acquisition and analysis systems can be built around a set of primitive functions to provide the necessary support for a large class of experiments. The set of primitive functions must be complete, the functions must be orthogonal to each other and they must allow composition. These requirements guarantee that the system is simple, open-ended and multi-functional.

A number of successful attempts in this direction have been reported in the literature, [1], [2]. The main benefits of this approach emerge from the higher level environment provided for the user. These benefits are: a more reliable and better controlled environment, a variety of software tools available for application design, testing and error localization. This type of system tends to evolve in the direction of knowledge processing systems. The user is relieved from the need to understand the details of concurrent programming and the system supports a sophisticated use of knowledge accumulated in previous experiments and provides the means to correlate experimental results with theoretical models.

This approach is strictly superior from a software engineering point of view to the adhoc design of experiment specific systems in which a user builds a system tailored to his needs, for each new experiment, using a library of basic routines and procedures. When the sophistication of the experiment increases or when the control functions needed are more complex, the effort to build such a disposable system becomes prohibitive.

In a data acquisition and analysis system the data acquisition function is responsible for the processing of the streams of input raw data. The input raw data consists of a sequence of experiment dependent information structures containing correlated data and called events. An event is like a snapshot of the physical system taken at a given moment of time. It consists of an array of data representing the results of measurements provided by different sensors observing the physical system which is the subject of the experiment. The events are generated randomly depending upon the physical phenomena being observed. Often the events occur in bursts; in most cases the accelerator produces a pulsed beam and only during the short pulse time accelerated particles bombard the target and produce events.

There are two basic approaches in handling the raw data. In the first approach, all events are recorded and the raw data is subject to a multiple step analysis procedure as follows: a real-time analysis is done using samples of the input data in order to monitor and control the experi-

ment. Post experiment (off-line) analysis can then be done repeatedly since all information recorded during the experiment is available.

To give an idea of the amount of information collected, it should be pointed out that an experiment may run continuously for days, generating data at a high rate. For example, an event may contain in excess of one thousand parameters (measurement points) and events may be generated at a rate of one thousand events per second or more. Typically, a measured parameter can be represented as a 16 bit integer so that the data rate of such an experiment could be higher than 16 Mbps. Fortunately, only a low percentage, for example 10 percent of the parameters of a large event are non zero, due to physical considerations, since in a large system there are many uncorrelated elements. With large, sparse events, the data compression function is most important for the acquisition system and should be built into the system as a standard feature.

A second approach is to perform some data reduction during the data acquisition and to record only events which are significant for the experiment. This approach is not used very often since usually limited computing resources are available during the experiment. The insufficient computing resources prohibits multiple analysis and limits the complexity of the analysis done during data acquisition. Another reason making this approach less popular is that any conceptual or procedural error in the filtering procedure leads to a nonrecoverable loss of data. Often, the definition of what a "significant event" means, changes considerably during the off-line data analysis.

While the task of the data acquisition part of the system is to store safely all the data captured, with a minimum amount of transformations done to it, the analysis part of the system concentrates upon the control of the experiment. Interactive control can be exercised by one or more users whose primary concern is the physical significance of the results. To determine that, a user must process in real-time statistical samples of the input raw data. The results of this processing need to be presented in a compact, meaningful format, using some graphics facilities supported

by the system. The system must provide also a number of primitive functions for the manipulation of different user defined information structures.

Another strong requirement for the analysis system is to allow a user to redefine dynamically his analysis procedure while the experiment is running and data is collected. Multiple analysis procedures running concurrently need to be supported. A considerable effort has been invested in a design to allow dynamical changes of the analysis procedure while data is being collected. To provide enough flexibility, the systems primitive functions do not manipulate the different objects in the computational space directly but through their descriptors. Such objects are events, spectra, conditions, etc. There are different types of conditions, boolean, windows, etc.; they can be altered at execution time and in this way the execution logic can be changed in a limited way. The price to pay is an increase in execution time since part of the analysis is executed in an interpretative way. Also, a considerable increase in the complexity of the design is associated with this approach, with severe implications upon the ability to find an error in a user written procedure.

If the system provides a set of tools to support editing, compilations, interactive debugging, as well as primitives to create, connect to the environment new processes and other related functions, then the requirement to change dynamically the analysis procedure can be implemented by creating a new user process which carries out a new analysis logic requested by the user. The old analysis may run until the user decides to terminate it, by disconnecting it from the environment. This approach leads to a more comprehensive and simpler design.

The analysis system must be conceived as a knowledge processing system. It should support user's access to a large data base containing results of previous similar experiments and be able to compare them with the current results. It should support access to a database of theoretical models and should be able to determine which model fits the best of the experimental data. It should also help determine the set of model parameters which are adequate for a particular type

of experiment.

In conclusion a data acquisition and analysis system must be designed to satisfy a few conditions:

- the system must be able to meet timing constraints imposed upon some of its component processes,
- the system must be open-ended, allowing a user to define dynamically one or more analysis procedures which are experimentally specific,
- it should contain a shell allowing a user to interact with the system by using a set of standard commands,
- the system should be able to recover from user errors and the data acquisition functions should not be affected by errors in the user-written analysis procedure.

This brief presentation outlines the complexity of a data acquisition and analysis system and the need for a well structured system design.

3. SYSTEM STRUCTURE AND PERFORMANCE EVALUATION

A data acquisition and analysis system is usually implemented as a distributed system. A relatively common design consists of a front-end computer linked via a high speed network to a back-end mainframe. The data acquisition function as well as most of the control functions are implemented on the front-end while the analysis system is designed as a set of cooperating processes running on the back-end computer. The cooperating processes may communicate either by sharing a common memory segment in a procedure oriented design or by using a message delivery system. Procedure oriented systems are more common, but if the operating system supports an efficient inter-process communication mechanism, the message passing is advantageous.

An alternative design can be based upon the use of a multiprocessor system to host the data acquisition and analysis system. Depending upon the multiprocessor system architecture the same two options, procedure oriented with shared memory or message oriented designs may be approached.

Most of existing systems are implemented on uniprocessor systems, often on a minicomputer, and the procedure oriented approach is the most frequently encountered. A set of cooperating processes communicate via a shared memory segment. Two cases are distinguished depending upon the way the host operating system maps a process into an address space.

In the first case each process has its own address space containing process code and private data. Protection among processes is achieved and error recover is manageable. The system has the potential of being nicely structured with distinct functions mapped into different processes. System processes implement the primitive functions provided by the system. Examples of system processes are: the data acquisition process, the dialog manager processes, a shell process running for each active user, driver process for graphics devices, an arbitrator process which decides what action should be taken in case a number of users with the same authorization level request conflicting actions, etc. User analysis procedures are mapped each into a distinct user process. It is important to structure the system in such a way that each process which carries out a given service may receive as input the data produced as output by any other process, in order to create execution pipes.

The data kept in common are in this case: input data buffers, output data of one process to be consumed by other processes, control information. The management of the common storage is the most important single problem in the system design. All user processes run independently of each other, but all processes compete for different logical and physical system resources. An important design objective is to structure the system such that an error in a user process does not affect either the data acquisition system or any user process.

As opposite to this situation, is the case when all system and user processes share a large address space and there is no protection of process private code or data. Such a system is described in [1], [8]; and the different processes run as PL/I tasks in the MVS environment. The obvious problems are: the lack of protection among different processes as well as the difficulties to allow dynamic changes in analysis procedures. Since system code together with user written procedures form a load module, a user error may cause a total system shutdown. The execution is non-deterministic and deadlock prevention cannot be guaranteed. The procedure oriented systems with a number of processes sharing a large address space, tend to be less structured than the ones with isolated processes communicating via a shared segment.

To ensure serialization of global resources, in case of a procedure oriented design, the system can either define a number of monitors, one for each resource or only locks associated with each resource. Timing problems, error recovery, process synchronization, avoidance of deadlocks are all non-trivial matters in this environment. In case of overwriting a data element it is very difficult to establish the faulty procedure.

The most severe problems are the ones related to concurrency control and synchronization. Often, instead of writing monitors to guard critical sections, a number of locks are defined. The critical sections of the code become unnecessarily large, contain subsequent requests for additional locks, calls to other procedures which may also request new locks, etc. Combined with the fact that the usually large number of locks are not arranged in a hierarchy, this lack of structure makes system deadlocks a relatively frequent occurrence.

A very critical issue is the choice of the programming language used for system implementation. Data acquisition and analysis system may be very complex consisting of hundreds of thousands lines of code. For example, the system described in [8] consists of more than 130,000 lines of PL/I code. Such a system can only be implemented in a high level language which supports concurrent processing. Usually such a language has a sophisticated run time system. In this

case, four different layers of software, each providing a different environment, can be recognized: the operating system environment, OSE, the high level language environment, HLE, the presentation environment, PE and the application one, AE. The presentation layer implements the primitive functions provided for the user by the data acquisition and analysis as well as the overall environment control functions. The application layer implements user's analysis procedures. A well structured design uses at each layer only the primitive functions provided by the previous one and communicates with it only through defined channels. For example, at the presentation layer only the primitives of the high level language should be used.

The environment consistency constraints as formulated above are very often violated in the design of this class of real-time systems. There are cases when the HLE primitives are considerably less efficient than the primitives which a knowledgeable user may implement using directly OSE functions. As an example, consider the process synchronization functions provided by the PL/I multi-tasking environment. A control process performs all operations related to interprocess communications for processes residing in an address space; it also dispatches the highest priority process when the address space is activated by the system dispatcher. Since these functions are very time consuming, an alternative could be to implement directly a number of locks based upon functions provided by the hardware, using for example the compare and swap instruction. A deadlock may occur in this case when a high priority process which waits for such a lock, external to the HLE, is dispatched by the control task which is aware of his environment

There are also cases when HLE does not provide all functions needed by the PE. For example in case of a peripheral device not supported by the high level language environment, or when additional functions are needed, the only solution is to communicate directly from the presentation environment with the operating system, by using low level system primitives. The following example illustrates the dangers of this approach. Since MVS does not support multiple files on a magnetic tape and since asynchronous I/O operations involve a large overhead in PL/I, it was

decided to write a magnetic tape driver program using physical I/O operations. One of its functions was to logically close a tape file whenever the data being collected did not arrive in a specified time interval, by writing two end of file marks. In this case, if the system would crash, the last file on the tape would still be closed and could be processed normally. If data would eventually arrive, two backward file skips could reposition the tape after the last block of data and the new data would be part of the same file. Unfortunately, the procedure responsible for writing end file marks, was not designed to return a completion code. As a result, when it failed to write one of the end of file marks, the two backward skips, positioned the tape at the beginning of the current file and valuable data was overwritten. Also ignored was the fact that an end of file mark is a special block, separated from the previous data block by a larger gap, three times larger than a normal gap between data blocks. When the input data rate and the timeout were not properly matched, this strategy lead to a five time increase of the number of I/O operations due to the additional writing of end of file marks and backward skips and, in the same time, to a tape density up to five times lower than normal since most of the data blocks were separated by long gaps. This example illustrates also the need of a single design in which exceptional conditions are handled separately and the code responsible for ordinary processing implements a comprehensible logic. As a conclusion, the different environments need to be properly interfaced. The choice of a high level language should be made only after a very serious investigation of all features needed by the presentation and application environments. Mixing up lower level code in such a complex system leads very often either to errors or to an unstable system, greatly sensitive to minor changes in basic system software.

Few attempts have been made in the past to model the performance of this type of system, in particular, to determine the highest data rate a system may support. Most often it is left to the user to either measure using the experimental hardware, or to simply estimate if all input data is captured. Such an approach is not acceptable in case of large systems with multiple analysis pro-

cedures. This problem becomes even more important when it is required that one or more analysis procedures process all input data. It was previously indicated that due to the vast amount of data involved it is highly desirable to have a certain level of data reduction done during the data acquisition, based upon the results of one or more analysis. Since this is entirely possible if the parallelism of a multi-processor system is exploited, it is conceivable to think that in the near future, a significant part of data analysis will be done in real-time. This requires a careful analysis of processing time of all critical analysis processes, the processes which must complete their analysis in real-time. It should also be pointed out that in case of data acquisition and analysis systems the mapping of processes is a very simple and natural one.

The modeling methodology presented in this paper is intended to provide general guidelines and the models need to be validated in each case. A hierarchical modeling approach is suggested; first the individual processes are modeled separately and parameters as the mean processing time per data buffer are determined. In case of procedure oriented systems these values needs to be adjusted to take into account contention for logical and eventually physical system resources. Then an upper limit for the total number of processes must be established. Using these values it is necessary to determine if the system can support the required input data rate considering the conditions imposed upon the number of analysis processes which must process all input data. Essentially this analysis gives only bounds for the system performance. An exact analysis is difficult since the models do not have a product form solution. The approximation made in the next two sections are necessary in order to obtain analytical solutions which can be easily applied.

4. ANALYSIS OF TIME DELAYS DUE TO RESOURCE SERIALIZATION FOR A PROCEDURE ORIENTED DATA ACQUISITION AND ANALYSIS SYSTEM

Modeling and analysis of parallel processing and of programs with internal concurrency is of considerable interest especially for information and knowledge processing systems. Most of

the research effort in this area is directed toward modeling of systems in which processes executing in parallel require virtually no synchronization or inter-process communication. The assumption of loosely coupled processes is needed in order to have separable models for the system. To model the concurrency associated with tightly coupled parallel processes in addition to the service centers corresponding to the physical system, service centers corresponding to logical resources need to be defined when representing the system as a network of queues. Since a process executing may require simultaneously both a physical and a logical resource, the models are no longer separable. Studies as the ones reported in [4], [5] are concerned with the effect of concurrency upon the system throughput for loosely coupled systems.

In case of real-time systems, the main concern is the ability of the system to perform a given function in a specified amount of time, with less regard to resource utilization. Each process runs on a separate processor so there is a low level of contention for the physical system resources. It is assumed that there are N processes $\pi_0, \pi_1, \pi_2, \ldots, \pi_{N-1}$, which may run concurrently in such a way that their execution logic requires frequent access to some common logical resources. Such resources are control structures defining the system status, resource allocation tables, control blocks, data buffers, etc., usually kept in a shared storage such that all processes may access them. The problem of interest is how does the contention for these resources affect the execution time of each process.

To ensure serialization, access to each resource is done by means of a monitor. It is assumed that there are M such monitors, $M_0, M_1, \ldots, M_{M-1}$, each containing the critical section associated with access to a resource. A process may be in one of three states:

- *active*, which means running in a non-critical section,
- *in a monitor state*, running in the critical section of a monitor,
- *blocked*, waiting for a monitor currently held by other process.

To simplify the analysis, it is assumed that the environment is homogeneous, all N process have the same pattern of behavior, each active process requests access to a monitor at the rate of λ requests per unit of time. The requests from any process are uniformly distributed into the set of monitors. Also the monitors have similar patterns of behavior, the service rate of a monitor being μ requests service per unit of time.

The system can be modeled as a closed system with $M + 1$ service centers, an infinite server and M service centers, each corresponding to one of the monitors. The queuing network model of this system does not have a product from solution since the service times are not exponentially distributed and also priority queuing is necessary. It is assumed that process π_0 has a high priority, since it maps the data acquisition while processes $\pi_1, \ldots, \pi_{N-1}$ have equal priorities, each of them mapping one analysis procedure.

A possible solution is to model the system as a continuous time semi-Markov chain with a finite number of states. An expert analysis can be done considering a state to be defined by the pair

$$< \text{\# of active monitors, \# of blocked processes} >.$$

In this case, the total number of states is of the order $O(N \times M)$.

The approximation used in this paper is to solve the balance equations of system in which all states with the same number of blocked processes are aggregated into a single state. So a semi-Markov process models the system and the system is in state i if i processes are blocked, waiting for monitors. The following notation will be used:

- M_i is the number of busy monitors in state i. It follows that the number of active processes in state i is:

$$A_i = N - (M_i + i),$$

- α_i is the probability that in state i a request for a monitor will lead to the blocking of the requesting process since the monitor is in use,
- β_i is the probability that in state i a monitor released by a process has at least another process waiting for it.

The transition from state i to state $i+1$ has the probability: $\lambda \times A_i \times a_i = \lambda \times (N - M_i - i)\alpha_i$ while the transitions from state $i+1$ to state i has the probability: $\mu \times M_{i+1} \times \beta_{i+1}$.

The equilibrium equations give the probability of finding the system in state i as:

$$P_i = P_0 \times \rho^i \times \frac{N - M_0}{M_i} \times \prod_{j=1}^{i-1} \delta_j, \quad \text{for} \quad 1 \le i$$

with:

$$\gamma_j = \frac{\alpha_{j-1}}{\beta_j}$$

The largest number of blocked processes is:

$$k_{max} = N - M_{max}$$

This corresponds to the situation when no process is active and running in a non-critical region.

The probability of finding the system in state 0, P_0 can be determined from the condition:

$$\sum_{i=0}^{k_{max}} P_i = 1$$

and has the value:

$$P_0 = \frac{1}{\sum_{k=0}^{k_{max}} \rho^k \times \frac{N - M_0}{M_k} \times \prod_{j=1}^{k-1} \delta_j}$$

It follows that:

$$P_i = \frac{\rho^i \times \frac{N - M_0}{M_i} \times \prod_{j=1}^{i-1} \delta_j}{\sum_{k=0}^{k_{max}} \rho^k \times \frac{N - M_0}{M_k} \times \prod_{j=1}^{k-1} \delta_j}, \quad \text{for} \quad 1 \le i$$

Let us assume that the pattern in which processes $\pi_1, \pi_2, \ldots, \pi_{N-1}$, request monitors is similar. As a result the number of busy monitors is the same, in all states:

$$M_i = M_\alpha$$
$$\gamma_i = 1 \qquad \text{for} \quad 0 \le i \le N - M_\alpha$$
$$\delta_j = \frac{N - j}{M_\alpha} \quad \text{for} \quad 0 \le j \le N - M_\alpha$$

In this case the probability of having i processes blocked is:

$$P_i = \frac{\rho^i \times \prod_{j=1}^{i=1} \delta_j}{\sum_{k=0}^{N-M_\alpha} [\rho^k \times \prod_{j=1}^{k=1} \delta_j]}, \quad \text{for} \quad 1 \le i \le N - M_\alpha$$

The average number of processes blocked, waiting for each of the active monitors is:

$$\bar{N}_b = \frac{\sum_{i=0}^{N-M_\alpha} (i \times P_i)}{M_\alpha}$$

The processing time per data buffer in absence of any delays due to resource serialization, for each process $\pi_0, \pi_1, \pi_2, \ldots, \pi_{N-1}$ are denoted as $R_0, R_1, R_2, \ldots, R_N$. It is assumed that $R_1, \ldots, R_N$ are mutually independent and exponentially distributed random variables with means $1/\lambda_1, 1/\lambda_2, \ldots, 1/\lambda_N$.

Then the average time spent by process π_i waiting for monitors, during the processing of a data buffer can be approximated by:

$$\bar{w}_i = \frac{1}{\lambda_i} \times \rho \times (\bar{N}_b - 1), \quad \text{for} \quad 1 \le i \le N - M_\alpha$$

Consequently, the average processing time per data buffer for process π_i in the presence of delays

due to resource serialization is:

$$\frac{1}{\lambda_i^w} = \frac{1}{\lambda_i} [\, 1 + \rho \times (\overline{N}_b - 1) \,], \quad \text{for} \quad 1 \le i \le N - M_\alpha$$

The average time spent waiting for monitors by the data acquisition process π_0 which a high priority, can be approximated as:

$$\overline{w}_0 = \frac{1}{2} \times \frac{1}{\lambda_0} \times \rho$$

Its mean processing time per data buffer is:

$$\frac{1}{\lambda_0^w} = \frac{1}{\lambda_0} [\, 1 + \frac{1}{2} \times \rho \,]$$

The processing time per data buffer for each process, $\pi_0, \pi_1, \pi_2, \ldots, \pi_{N-1}$ in the presence of delays due to resource serialization are mutually independent and exponentially distributed random variables with means $\frac{1}{\lambda_0^w}, \frac{1}{\lambda_1^w}, \frac{1}{\lambda_2^w}, \ldots, \frac{1}{\lambda_{N-1}^w}$.

These values are used to estimate the highest data rate which an be sustained without any loss of input data, in the presence of the serialization delays, when the system is data drive. They can also be used in conjuncture with the technique to be described in the next section to estimate the percentage of data lost when additional coupling conditions between acquisition and analysis processes are imposed. Such conditions could be: the acquisition process has to wait at least for one analysis process to complete or it has to wait for all analysis process to complete.

V. ANALYSIS OF A MESSAGE ORIENTED DATA ACQUISITION AND ANALYSIS SYSTEM

Assume N processes, $\pi_0, \pi_1, \pi_2, \ldots, \pi_{N-1}$, communicating via messages. Process π_0 implements the data acquisition and interactive control functions while processes $\pi_1, \ldots, \pi_{N-1}$ each map a different analysis procedure.

The message delivery function is provided by the distributed operating system and it is fully transparent to the data acquisition and analysis system described here. It allows asynchronous as well as synchronous communication between any pair of processes and the overhead associated with message delivery is very small in comparison with the processing time per data buffer of each of the n processes. The message delivery system has the capacity to store up to m messages from any source to any destination process; so it may contain $N \times N$ mailboxes, each with maximum capacity of m messages. Whenever a mailbox is filled up to its capacity, m, any incoming message is lost. The system may have an overflow count, which will record the total number of messages lost due to the overflow, for any given destination. Process π_i may communicate with process π_j by sending a message which will be achieved as the k-th message in the proper mailbox, $M_{j,i}(k)$. In case of synchronous communication, after sending the message, π_i will wait until process π_j receives his message and sends back an acknowledgment. In case of asynchronous communication π_i will continue processing without assuming that an acknowledgment will be sent. A message may contain either data or control information and the receiver of the message needs first to identify the content of the message and take proper action. The message delivery system supports broadcasting. In case of a broadcast message only one copy of the message is maintained and kept until it is received by all partners.

No direct interaction between processes π_1 ,... Pi_{N-1} is assumed. Each of them may run on a different processor or a group of them may share a processor. Process π_0 which runs on a dedicated processor, broadcasts a block of data at a time to all of the analysis processes, $\pi_1, \ldots, \pi_{N-1}$. Each process runs in a loop and performs a specified computation for each input data block. Process π_0 essentially executes the following code:

```
do until a stop request
begin
   read data buffer
   broadcast buffer to all analysis processes
   compress data buffer
```

```
    write data buffer to output device
    if user command then
    begin
      create a process to execute the command
    end
    update system status
  end
```

Whenever a command is entered, a special command execution process is created. This process decodes the command and produces an order either for the data acquisition process or for any of the other processes. A special communication path is then established between the command process and the target process. To simplify the model it is assumed that the frequency of user commands is much lower that the input data rate and the processing associated with input commands can be ignored.

The read operation is synchronous, process π_0 waits until the next data buffer is available at the input. The write operation is asynchronous, π_0 writes a buffer to the output device and checks the completion codes at the next cycle before performing the next write operation.

The processing time per data buffer for each process $\pi_0, \pi_1, \pi_2, \ldots, \pi_{N-1}$ are denoted as $R_0, R_1, R_2, \ldots, R_{N-1}$. It is assumes that $R_1, \ldots, R_{N-1}$ are mutually independent and exponentially distributed random variables with means $1/\lambda_1, 1/\lambda_2, \ldots, 1/\lambda_{N-1}$. It is assumed that $1/\lambda_0$ is much smaller than any of the $1/\lambda_i, i = 1$ to N.

A hierarchical modeling approach [3] is considered. First processes $\pi_1, \ldots, \pi_{N-1}$ are modeled. The actual modeling methodology depends upon the mapping of processes $\pi_1, \ldots, \pi_{N-1}$ to processors. It C of them share a given processor then a closed central server model with C different customer classes is constructed and solved for the processing associated with one data buffer. For each model there are as many service centers as required by the processes using the processor. The queuing network models are separable since there is no concurrency within each process. As a result of the modeling at his level the mean processing times $R_1, \ldots, R_{N-1}$ are determined. To model the asynchronous write operation of process π_0 the

method of surrogate delays proposed in [7] may be used.

Different types of synchronization between π_0 and $\pi_1, \ldots, \pi_{N-1}$ can be analyzed. First it is assumed that the system is driven by the input data stream and does not require that any of the analysis processes $\pi_1, \ldots, \pi_{N-1}$ analyze all input data. The highest data rate which can be sustained without any loss of data is:

$$D_{\max} = \lambda_0$$

data buffers per second. This case is implemented by the π_0 code presented above. If the input data rate is equal to the maximum rate none of the analysis procedures implemented by processes $\pi_1, \ldots, \pi_{N-1}$ are able to analyze all input data buffers, but only statistical samples of it. Process π_i is able to analyze only the ratio λ_i/λ_0 of the incoming data buffers. If the actual input data rate D, satisfies the condition $D \leq D_{\max}$ then processes with $\lambda_i \geq D$ are able to analyze all incoming data buffers.

Since message queues have a limited capacity the system is self regulating, each process π_i consumes data at its own peace. The level of communication and synchronization between processes is minimal and the system is very robust.

Now we examine the case when it is required that the acquisition process waits for all of the analysis processes to complete before accepting the next data buffer. A typical code for process π_0 is:

```
do until a stop request
begin
   read data buffer
   broadcast data buffer to all analysis processes
   compress data buffer
   write data buffer to output device
   if user command then
   begin
      create a process to execute the command
   end
   for i from 1 to N − 1 do
   begin
```

```
      wait to receive acknowledgment for process termination
    end
    update system status
  end
```

The time spent by process π_0 waiting for all its satellite processes to complete is:

$$W_0^{\alpha} = \max(R_1, R_2, \ldots, R_{N-1})$$

Since it is assumed that $R_1, \ldots, R_{N-1}$ are mutually independent and exponentially distributed random variables with means: $1/\lambda_1, 1/\lambda_2, \ldots, 1/\lambda_{N-1}$, the distribution function of the random variable W_0^{α} [9] is:

$$F_{W_0^{\alpha}}(x) = \prod_{i=1}^{N-1} F_{R_i}(x) = \prod_{i=1}^{N-1} [\, 1 - \exp[\, -x \, \frac{1}{\lambda_i} \,] \,]$$

The average time spent by the acquisition process waiting for all analysis processes to complete, [4], is:

$$\overline{W}_0^{\alpha} = E[\, W_0^{\alpha} \,] = \sum_{i=1}^{N-1} \frac{1}{\lambda_i} - \sum_{i<j} \frac{1}{\lambda_i + \lambda_j} + \cdots (-1)^{N-2} \times \sum_{i_1 < i_2 \cdots < i_{N-1}} \frac{1}{\lambda_{i_1} + \lambda_{i_2} + \cdots \lambda_{i_{N-1}}}$$

When all analysis processes perform similar computations and their processing times can be considered equal:

$$\lambda_i = \Lambda, \quad \text{for} \quad i = 1 \text{ to } N-1$$

to waiting time of the acquisition process can be approximated by:

$$\overline{W}_0^{\alpha} = \frac{1}{\Lambda} \times \ln(N-1)$$

The previous results can be easily adjusted for the case when the acquisition process must wait only for a subset of the analysis processes to complete their execution. When it is required to perform data reduction depending upon the results of k critical analysis processes, the code for process π_0 could be:

```
do until a stop request
begin
  read data buffer
  broadcast data buffer to all analysis processes
  if user command then
  begin
    create a process to execute the command
  end
  for i from 1 to k do
  begin
    wait to receive acknowledgment for process termination
  end
  if significant event then do
  begin
    compress data buffer
    write data buffer to output device
  end
  update system status
end
```

Finally we examine the case when it is required that at least one analysis procedure processes all input data buffers. The code for the acquisition process can be derived directly from the previous cases.

The time spent by process π_0 waiting for the first analysis process to complete is:

$$W_0^1 = \min(R_1, R_2, \ldots, R_{N-1})$$

The distribution function of the random variable W_0^1 is:

$$F_{W_0^1}(x) = 1 - \prod_{i=1}^{N-1} [\, 1 - F_{R_i}(x) \,] = 1 - \prod_{i=1}^{N-1} \exp[-x \frac{1}{\lambda_1}] = 1 = \exp[\sum_{i=1}^{N-1} x \frac{1}{\lambda_i}]$$

Then it follows that

$$\overline{W}_0^1 = E[W_0] = \sum_{i_1 < i_2 \cdots < i_{N-1}} \frac{1}{\lambda_{i_1} + \lambda_{i_2} + \cdots \lambda_{i_{N-1}}} = \frac{1}{\lambda_1 + \lambda_2 + \cdots \lambda_{N-1}}$$

When

$$\lambda_i = \Lambda, \quad \text{for} \quad i = 1 \quad \text{to} \quad N = 1$$

the waiting time is:

$$\overline{W}_0^1 = \frac{1}{\Lambda} \times \frac{1}{N-1}$$

The highest data rate which can be sustained without any loss of data, D_{max} data buffers per second is given by:

$$D_{max} = \frac{1}{\frac{1}{\lambda_0} + \overline{W}_0}$$

with $\overline{W}_0$ either $\overline{W}_0^{\alpha}$ or $\overline{W}_0^1$ depending if the data acquisition waits for all analysis process to complete or only for the first one.

When the actual data rate D, is higher than D_{max} then only a fraction of the data is captured:

$$\eta = \frac{D_{max}}{D}$$

5. SUMMARY

Nonhomogeneous systems with processes which must meet strict timing deadlines cooperating together with processes with no deadlines form a distinct class of real time systems, called in this paper semi-hard systems. They can be implemented either as procedure oriented systems or as message oriented systems. The systems based on the asynchronous message passing paradigm are simpler to implement and their timing analysis is easier. In case of procedure oriented systems, the contention for system's logical resources increases the average processing time of each process.

The widespread use of multi-processor systems may alter drastically the design of data acquisition and analysis systems since it will eventually allow multiple analysis process with strict deadlines increases, a significant data reduction can be done. Estimations of the average delay experienced by the acquisition process, when it must wait for one or more critical analysis processes to complete, are given. Also a methodology for system modeling is sketched.

The environment consistency constraint is discussed and a simple design of the system is advocated.

ACKNOWLEDGMENTS

I wish to thank Allen Weis and Alex Chandra for their support. I also thank P. Heidelberger for helpful discussions.

AFFILIATION OF THE AUTHOR

D.C. Marinescu was on temporary assignment at IBM Thomas J. Watson Research Center, Yorktown Heights, N.Y. 10598. He is with the Computer Science Department, Purdue University, West Lafayette, IN 47907.

REFERENCES

[1] **F. Bush, H. Goeringer, W. Hartman, J. Lowsky, D.C. Marinescu, M. Richter, K. Winkelman,** *Experiment Data Acquisition and Analysis System,* vol. 1,2,3 GSI Report 83-4, ISSN 0171-4546

[2] **E. Busse,** et al. *The Data Acquisition and Monitoring System HOOPSY,* IEEE Transactions on Nuclear Science, vol. NS-28, No. 5, pp. 3674-3679, 1981.

[3] **P. J. Courtois,** *Decomposability, Instabilities and Saturation in Multiprogramming Systems,* CACM, vol. 18, pp. 371-377, 1975.

[4] **P. Heidelberger, K.S. Trivedi,** *Analytic Queuing Models for Programs with Internal Concurrency,* IEEE Transactions on Computers, vol. C-32, No. 1, pp. 73-82, 1983.

[5] **P. Heidelberger, K.S. Trivedi,** *Queuing Network Models for Parallel Processing with Asynchronous Tasks,* IEEE Transactions on Computers, vol. C-31, No. 11, pp. 1099-1109,

1982.

[6] **A. Holm** et al, *Real Time Programming in a Data Acquisition and Analysis System,* IEEE Transactions on Nuclear Science, vol. NS-28, No. 5, pp. 3731-3738, 1981.

[7] **P.A. Jacobson, E.D. Lazowska,** *Analyzing Queuing Networks with Simultaneous Resource Possession,* CACM, vol. 25, pp. 142-151, 1981.

[8] **D.C. Marinescu, F. Busch, H. Hultzsch, J. Lowsky, M. Richter,** *Extended Data Acquisition Support,* IEEE Transactions on Nuclear Service, vol. NS-31, pp. 914-924, 1984.

[9] **K.S. Trivedi,** *Probability and Statistics with Reliability, Queuing and Computer Science Applications,* Englewood Cliffs, N.J., Prentice-Hall, 1982.