MICSIM--THE SIMULATION MODEL OF FDNY'S COMPUTER-AIDED DISPATCH SYSTEM

Michael Geller

ABSTRACT

Firefighting units in the borough of Brooklyn are currently dispatched with the aid of FDNY's new computer dispatch system, MICS. The system is designed around dual PDP 11/45s which are supported in a "fallback" (or back-up) mode by dual INTEL 8080 micro-processors. The computer processes alarms, recommends the assignment of available units, notifies assigned units by voice and hard copy terminals located in fire stations, monitors status changes of firefighting units and incidents, and dynamically adjusts the borough's firefighting coverage by recommending schemes for the relocation of units in order to minimize future (anticipated) response times. The system has been so successful in Brooklyn that it is being expanded to cover all five boroughs through a shared centralized PDP 11/70 computer system. The city-wide system will be called CMICS.

Simulation techniques were employed during all phases of MICS system development to insure that the necessary system criteria would be met under the ever-increasing alarm rates experienced in New York.

This paper describes the primary simulation tool, the GPSS simulation model, MICSIM, which was used to study the complex relationships between real-time system software, communication software, device handlers, and the application programs (components typically found in all multi-programming real-time systems). The end product of MICSIM is an explanation and quantification of the response times achieved by the system when responding to dispatchers' CRT requests.

Even with the sophisticated performance monitors that were used to measure CPU utilization and perform I/O and CPU accouting on a task-by-task basis, it was not possible to determine the real limiting factors in the system when the rates of CRT operator actions and input messages from the field increased. The increased overhead required to extract this data internally from the operational system would cause substantial interference thereby invalidating the data. Only with the aid of MICSIM was it possible to "profile" a task and determine the nature of the events that take place within MICS as it responded to external actions. The basic understanding provided by the model made it possible to take the necessary steps to achieve the desired performance standards.

Key Words and Phrases

Discrete simulation, Monte Carlo simulation, real-time systems, communication systems, performance measurement, response time, queues, bottlenecks, utlization, GPSS, hardware, software, throughput, multi-programming, systems design, fine tuning, input/output.

1. Introduction

New York City's Fire Department (designated FDNY), placed on-line, in Brooklyn, a Management Information and Control System (MICS) which provides computer assistance for the department's dispatching and communication functions. The system replaced manual procedures which were rapidly becoming inadequate for the soaring alarm rates currently experienced.

Since time is critical in the dispatching of firefighting resources, careful attention was given to the performance aspects of MICS. Performance criteria were established for the system's response time to dispatcher CRT requests under peak alarm rates. Realizing that it would be intolerable for a dispatcher to wait long for the computer to process his request, the Fire Department specified acceptable performance parameters to insure that the system would not "limit" the number of alarms which a dispatcher could process due to internal queueing problems. Bradford National Corporation, the developer of the MICS system, was

thus contractually obligated to demonstrate that the performance requirements would be met. Additionally, this necessitated the development of artificial techniques to provide peak alarm rate conditions so the performance would be documented prior to system cutover [1].

When developing a real-time system which must meet tight performance criteria, it is essential to have the means to measure the performance of the key system components and then to understand quantitatively how these elements interact. While the measurements may be obtained through some other type of performance monitoring, full understanding will probably require the use of a simulation model.

2. The Need for a Simulation Model

The wide variety in the types of demands made for fire department services, coupled with the random nature of alarm arrivals, and constantly increasing alarm rates make it difficult to formulate a statistical description of the performance of these tasks which MICS is called upon to perform during a peak alarm period. The complexity of the real-time system's interactions and dependencies further complicates the process or analyzing the system at any "moment in time" to discover the nature of the competition between tasks and system routines for the various system resources. A simulation model is a convenient aid for use in the study of a system driven by random phenomena and composed of a variety of tasks competing for finite and interdependent resources [3, 4].

A single alarm initiates a series of operator activities at several CRTs to dispatch and notify fire units. These are followed by inputs made on special purpose panels in both the central communications office and the firehouses to update unit and incident status files. Alarms may originate from mechanical street boxes (BARS), from newer electronic street boxes (ERS) which permit voice communication as well as the transmission of digital signals and via direct telephone contact. Each source requires different initial processing with the latter two requiring computer intervention. Figures 1 and 2 show the MICS hardware configuration and the operation environment, respectively.

It was difficult under these conditions to predict the work load that the MICS system would face at alarm rates projected to be as high as three alarms per minute. One alarm can result in twenty to thirty dispatcher interactions, each one causing a task which, in turn, is a series of programs required to process the particular input message.

The execution of each task requires precise coordination of key hardware and software components. Messages are transmitted over various types of multiplexer equipment from the terminals (CRT and other terminal types) directing the message to the central processing unit. Communication software controls the transmissions by polling the various terminals. In MICS, this software is called Line Control. Each message is received character-by-character in a silo where it is removed by programs (called SILO programs) into buffers, forming a

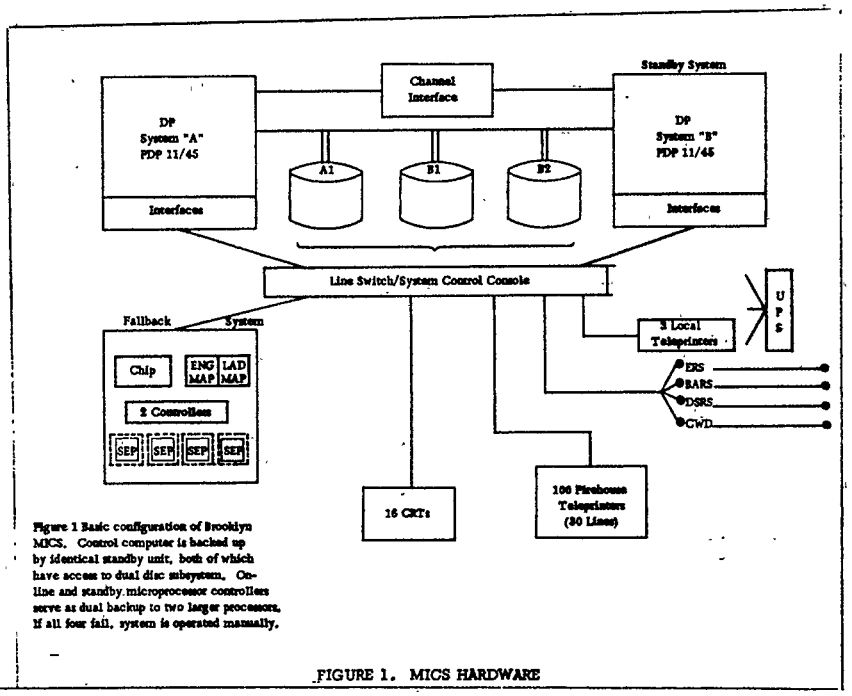


Figure 1 Basic configuration of Brooklyn MICS. Control computer is backed up by identical standby unit, both of which have access to dual disc subsystem. On-line and standby microprocessor controllers serve as dual backup to two larger processors. If all four fail, system is operated manually.

FIGURE 1. MICS HARDWARE

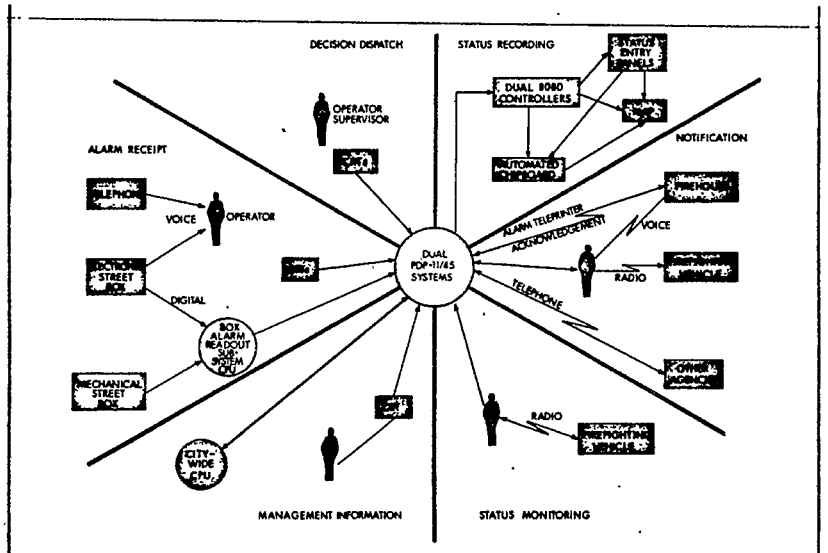


Fig 2 MICS modes of operation. All five modes are assisted by computer recommendations but operator or supervisor makes final decision. Box alarm readout system and city-wide deployment CPUs are not actually part of MICS but function with it

message for each line. When the message for a line is complete, the line control software passes the buffer to the Queue Controller, the program which interfaces with application tasks and controls the queueing of messages back to the terminals. Terminals may be associated with any number of modes of operation depending on the function which the dispatcher is performing. This requires a highly sophisticated queue controller. Finally, the input message is processed by application programs which invoke other device-dependent tasks when they perform I/O to update files or when program modules not currently in core are needed. These applications tasks are diverse and may range from a program that takes only five to ten milliseconds of CPU time and performs no I/O, to a sequence of mathematical programs performing several I/Os and requiring thirty seconds of CPU time. Coordinating all the software is a real-time operating system, written by Bradford, which provides the appropriate multi-programming environment with emphasis on timely and efficient dispatching of tasks, memory management, interrupt handling, etc. Figure 3 shows the flow of an alarm as it is handled by each dispatcher interacting with computer generated screens and making entries into special purpose equipment. It can be thought of as a macroscopic view of the GPSS modeling effort. Figure 4, on the other hand, is a more microscopic view following a message from its initiation at a dispatcher's CRT, thru the hardware and software processing that follows, until a response is transmitted to the initiating CRT. The latter modeling effort would be similar for other real-time systems and is discussed in more detail in the next section.

Almost every developer of large real-time systems must face this type of environment and it can be very difficult to project system performance. Many questions arise during system design:

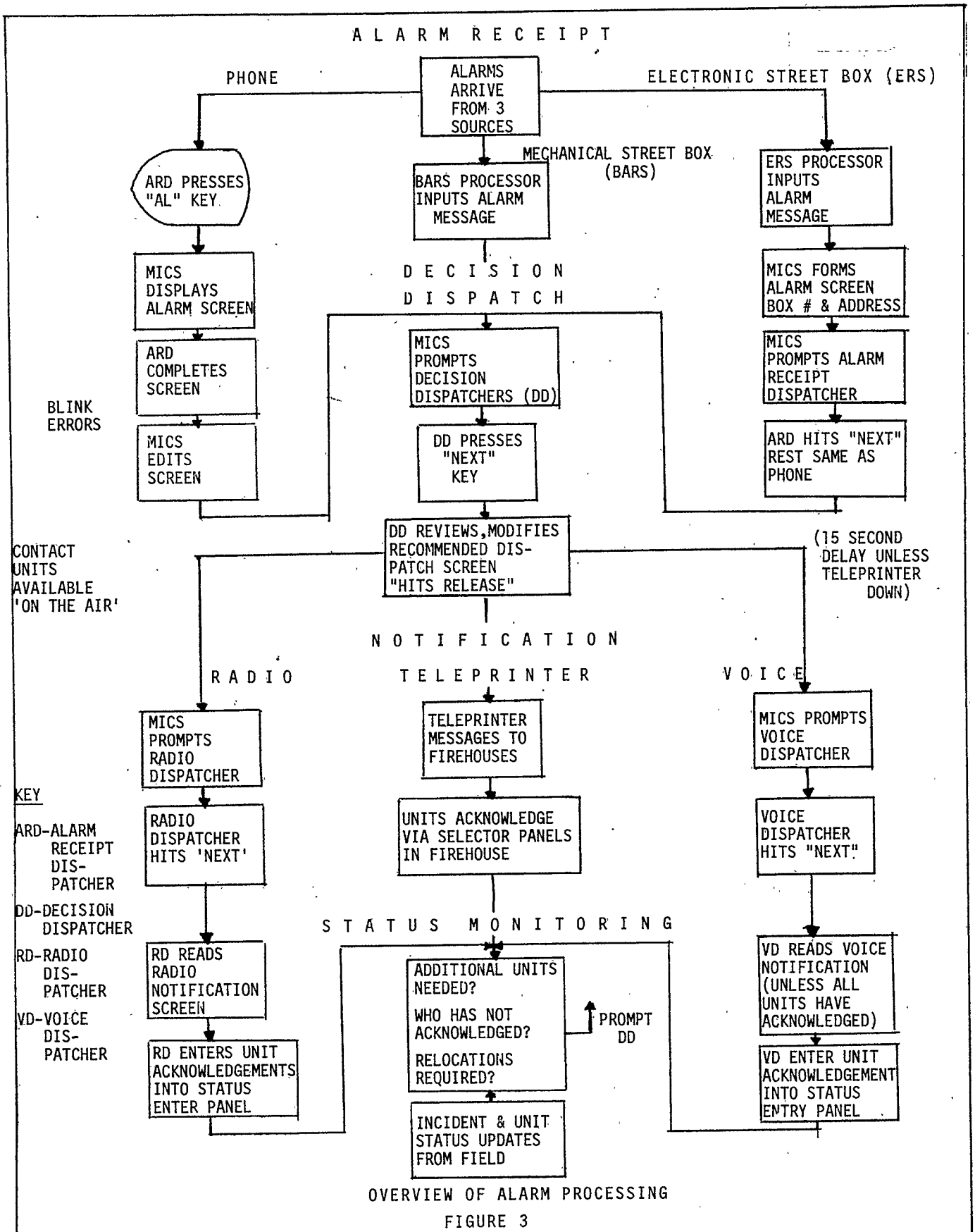
Is the CPU fast enough?

How much core is needed?

Will there be a bottleneck at the disk?

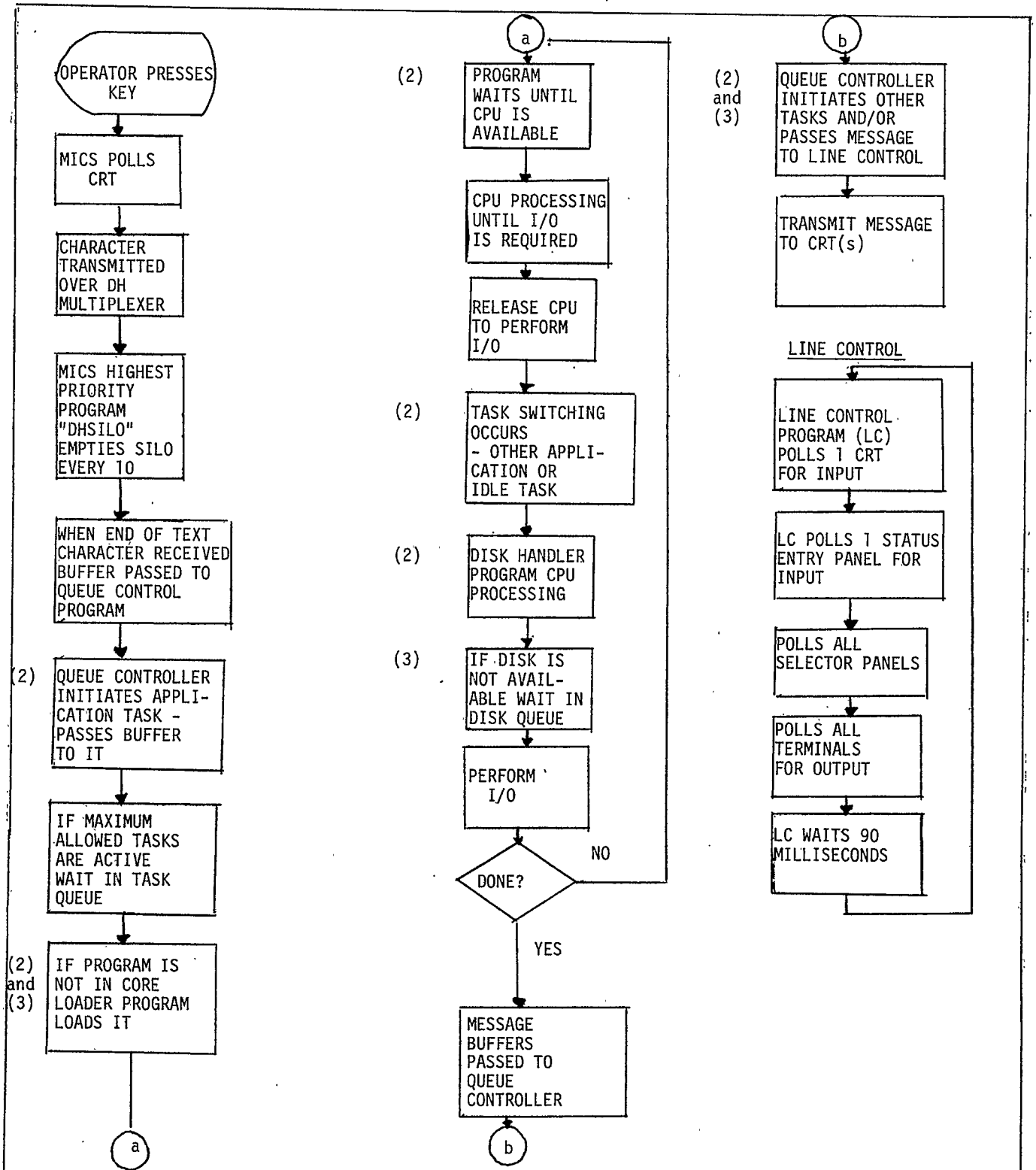
Which programs should be core resident and how much core should be allocated to task execution?

How should we prioritize the tasks, the queues?



OVERVIEW OF ALARM PROCESSING

FIGURE 3



SIMPLIFIED MESSAGE FLOW THRU MICS(1)

FIGURE 4

(1) This flow occurs each time a CRT function key is pressed.

(2) Contention for CPU by programs - lower priority tasks are preempted.

(3) Contention for disk.

The experiences described in this paper demonstrate that a simulation model of a system not only helps to answer these questions but may, in fact, be the only means to assure that the correct steps are taken to achieve the desired performance.

Even early incorporation of a sophisticated performance monitor [1] to make careful measurements of CPU utilization, task-by-task CPU and I/O accounting, and measurements of the elapsed time of tasks (response times), only enabled us to ascertain a measure of the performance and not to explain the reasons for the performance. The measurements told us where we stood, but we were frustrated in determining what steps to take to achieve our performance goals. The increased overhead required to internally extract enough data to pinpoint the cause of delays in the system would have caused enough interference to invalidate the measurements. The amount of work involved in interpreting these measurement was yet another factor leading to the simulation approach. It was at the point when several key personnel who studied the PM reports were unable to pinpoint the component(s) responsible for limiting system throughput, that the idea of a simulation model was born. Fortunately, in our case, the need for a simulation model was realized in time for us to employ this technique in order to understand the system response times, make adjustments, and ultimately achieve the desired performance. The remainder of this paper describes the MICSIM simulation model, written in GPSS H (an interactive version of GPSS), and discusses some of the studies that were performed with the model including examples of the steps that were taken to meet the performance specifications.

An interesting footnote is that the model identified the CPU as the principal bottleneck in the system, whereas previously, we thought that the CPU was no problem. It was overlooked as a bottleneck because MICS tasks ordinarily took less than 100 milliseconds so that even at peak rates of 1 to 2 tasks per second, it seemed that CPU availability would be ample. Even considering the communications overhead (line control), this seemed to be true. What we did not visualize was that the service overhead (loading tasks, I/O, task switching) which occurs during task execution would lead to such high CPU utilization at the time when the tasks require the processor. Our initial guess was that the disk was the bottleneck, and, in fact, we expected that MICSIM would prove this theory.

3. The MICSIM Model - An Outline

MICSIM is a model which simulates the flow of messages as they are transmitted over lines by various hardware devices and simulates their processing by the real-time system software in the MICS system. This model is written using GPSS which is an ideal language for simulating the flow of transactions through a system which "services" them, (i.e., a system where queues may form because of the finite resources and time required to process the transactions). By changing only a few cards, representing GPSS function and variables, it is possible to vary the alarm mix (BARS, ERS, Phone), the alarm rate, (alarm/minute), to change the processing speed of hardware, software, etc., and to change the distributions of the service and arrival times and the number of dispatchers. This made it easy to use the model for sensitivity analysis and i determining the "breaking points" in the system. For example, an analysis was done on how dispatch times vary depending on incoming alarm rates and number of dispatchers. The fire department used the resulting alarm response data and information about unit availabilities as the basis for developing special ("fallback") dispatching procedures for use during periods when alarm rates are unnaturally high (blackouts or disasters, Fourth of July, etc.). These procedures include special staffing (the use of an additional dispatcher to help work off the Decision Dispatch queue), as well as policy measures (changing the number and/or types of equipment required to respond to a particular type of incident) [2].

Communication

When an Alarm Receipt dispatcher completes entering or reviewing information on his screen,he presses the RELEASE function key. When MICS polls his CRT and determines that there is information to be transmitted to the CPU, it allows the transmission to occur. The data is transmitted over a DEC's DH multiplexer (direct memory access on output) where each character is placed in a silo used to store input from all the DH lines. Each character takes about a millisecond to be transmitted. Data from other devices such as the Status Entry Panels in the Communications Office and selector panels in each firehouse reaches MICS in a similar fashion although the polling cycles and transmission speeds are different. For example,data from the firehouse for acknowledging the receipt of an alarm or for updating a unit's status is transmitted via DEC's DJ multiplexer (character interruption output) over 600 baud lines. Each DJ line is polled each time the line control program is invoked, and the character-by-character

transmissions are stored in a silo for DJ input. CRT lines, are polled for input in a cyclical fashion (once each time line control is invoked) as are the Status Entry Panels which are used for entering incident and unit status updates and can also be logically connected to control a CRT.

The LINK and UNLINK blocks are GPSS instructions which make it simple to simulate polling. Whenever a message is generated at a CRT or one of the other devices, it is LINKED to a user chain until a corresponding UNLINK instruction is executed. Thus, the simulation of the line control software is a single loop in which UNLINK blocks are executed at the proper timing intervals. The exectuion of the UNLINK block allows the message (transaction) to be removed from its user chain and continue through a series of GPSS instructions simulating the character-by-character transmission and subsequent storage into the silo. Note that because the line control program runs at a lower priority than some other service routines, the rate of polling may decrease as the alarm rate increases. MICSIM permitted us to study this relationship.

The characters that are transmitted enter their respective silos (one silo for each type multiplexer -- DH or DJ) where they join another GPSS user chain. They wait there until one of the DHSILO or DJSILO programs removes them into a buffer for their particular line. These two programs are executed every 20 milliseconds and remove all the characters found in their respective silos. It takes approximately 40 microseconds to move each character. When an entire message of the line has been completely placed in the buffer, it is turned over to the Queue Control Program which, in turn, causes the appropriate task to be initiated ("attached"). These events are simulated in GPSS again using the LINK/UNLINK combination to cause the characters to remain in the silo until the DHSILO and DJSILO programs are executed. The use of the CPU for 40 microseconds per character removed from each silo is simulated by the PREEMPT of the CPU, the ADVANCE of V2 or V4 (number of characters on silo x 40 microseconds) and the RELEASE of the CPU. The characters pass through an ASSEMBLE block which has the effect of holding the buffer until the end of text character is removed from the silo.

Task Processing

Each input message results in the execution of an application task, a sequence of programs which process the message, update appropriate tables and files, and a formulate response messages to the dispatcher who initiated the input message and/or other dispatchers. For example, the release of a Recommended Dispatch Screen is processed by a task which starts with the program DDEDIT which can link to as many as ten other programs to update unit and incident status tables and files, cause teleprinter messages to be transmitted to the firehouses involved in the dispatch, prompt Voice and Radio dispatchers to notify the units, print a fire ticket for the incident, check for uncovered response neighborhoods. These tasks perform I/O to update the appropriate files or possibly to load programs which are not found in core when they are required. The latter is done by the LOADER program whenever programs LINK to one another. When an I/O is issued by an applications program, task switching occurs. One read/write results in 6 task switches, each of which takes .70 milliseconds. Queue services are required by each task to transmit output messages to the external devices (usually the CRTs) or to initiate independent tasks. The Queue Controller is then invoked and more I/O may occur. All the above processing was simulated in one GPSS subroutine called TASK. Whenever a task was to be initiated, the amount of CPU processing and the number of I/Os are stored in the transaction (GPSS transactions have parameters for storing this information) and it is transferred to the TASK subroutine. The loading of programs, the wait for available core, the performance of I/O including the execution of the disk handler program, RPDDT, and the seek and transfer time of the disk are all simulated. Multi-programming simulation is made possible by having each task RELEASE the CPU when performing I/O.

GPSS conveniently simulates the priority scheme of processing programs. For example, the disk handler RPDDT is given the same priority in the GPSS model that it has in the MICS system. Appropriate priorities are also given to Line Control, Queue Control, Loader, and the real-time clock handler, and each of the application tasks. GPSS automatically provides queue and utilization statistics for the disk, the CPU, and for core (the three greatest potential bottlenecks).

Through the use of other GPSS features, it was possible to develop a cross-matrix of preempting versus preempted programs showing the relative amounts of time involved. A similar matrix was developed for waiting tasks versus the task(s) or system program(s) causing the wait. It is interesting to note that only a modest number of instructions were required to simulate a multi-programming environment.

4. Making Decisions Using MICSIM

This section discusses the output of the model and describes some of the studies which we performed with MICSIM to find ways to improve performance.

The single most important piece of information that the model provided was the fact that the CPU was the major bottleneck in the system, significantly more so than either the disk or core. Table 1 is a machine component analysis extracted from the output of the MICSIM model. It should be noted that GPSS automatically provides queueing and utilization statistics as well as frequency distributions of transit times without the need of additional coding. Queue statistics for the CPU show that each time a task completes an I/O and returns to continue processing, its average wait for the CPU is 80 milliseconds. The most time-critical programs, such as DDRECM and DDEDIT, which are used to recommend dispatch assignments and notify the units in the firehouse where to go, perform more than 10 such I/Os and hence spend a significant portion of their time waiting for the CPU. The queue statistics for CORE and DISK for these programs show relatively small wait times.

The storage and facility statistics describe the utilization of the hardware and software components. The communication lines are treated as facilities as is the CPU since they process one transaction at a time representing characters and tasks, respectively. CPU utilization, for the 3-alarm per minute simulation, is shown to be 75% with the average seizure of the CPU being 10 milliseconds. From the storage statistics, we can seen the behavior of components which can serve more than one entry at a time. For example, the number of characters entering the silo, the average time they spend there, and the maximum number in the silo at one time are all shown. In this run, a maximum of 26 characters were found in the DH silo at any one time. We know a SILO can hold a maximum of 64 characters. A hardware interrupt for the DH SILO program was set to occur if the number of characters in the silo exceeded 32. The user chain statistics tell us how long messages wait to be polled on their respective lines. Histograms of task elapsed times are given for each type of task as well as one for all tasks. Finally the 'task versus task preemption' and 'CPU wait time' matrices describe why and by whom each task is preeempted from the CPU and why, how long, and for whom each task must wait for the CPU.

The CPU utilization, the polling times, and the elapsed time of task agreed very closely to those measured from the operational system. This gave us confidence in the validity of the model and trust that the bottleneck which it pinpointed (CPU) was correct. This was substantiated when we eventually cut CPU processing time of key software elements and improved performance as predicted.

TABLE 1. MACHINE COMPONENTS ANALYSIS

UTILIZATION (%)

Alarms/Min	CPU	Core	Disk
1	52.6	18.1	17.8
2	60.8	25.2	23.4
3 (1 DD)	75.3	45.7	35.6
3 (2 DDs)	72.5	44.0	33.4
6 (1 DD)	79.3	52.7	38.2
6 (2 DDs)	88.6	79.7	48.1

WAIT TIME (Sec.)

Alarms/Min	CPU	Core	Disk*
1	.04	.01	.01
2	.05	.01	.01
3 (1 DD)	.08	.07	.01
3 (2 DDs)	.09	.13	.01
6 (1 DD)	.10	1.31	.01
6 (2 DDs)	.14	.86	.02

* NOTE: CPU and Disk wait times are for each I/O done by a task.

DD = Decision Dispatcher

Figure 5 is an average profile of DDEDIT, a highly critical and time consuming task. It occurs for each alarm and any delay which it causes will hinder the Decision Dispatcher (DD), who must review the dispatch recommendations and make dispatch assignments. Since alarms are funnelled to his position by several Alarm Receipt Dispatchers, his is the position most likely to develop a queueing problem. The profile shows DDEDIT waiting for the CPU for more than a second, spending almost 0.5 seconds performing I/O and waiting for initiation for 120 milliseconds. During this particular 3-alarm per minute simulation, the number of task initiators was three, which was adequate to reduce average task wait for initiation to less than 0.13 seconds (see Table 1). The model also yields a histogram of DDEDIT elapsed times (i.e., the interval between input message receipt and the time when the first character of the response is returned).

A previous simulation run showed (in the task versus Task Wait Matrix) that the DDEDIT program was frequency "slowed down" by a less critical program, SUMCRT, which required 0.5 seconds of CPU and performed four I/Os at the very end of its processing. The SUMCRT program simply updates the summary displays of incidents currently active in Brooklyn. Lowering the priority of SUMCRT dramatically reduced the response time of the dispatch edit task. Other similar cases were found wherein a less critical task consistently 'interfered' with one of a more critical nature. Similar priority changes improved the overall response times of the critical tasks.

MICS SIMULATION MODEL

3 Alarms Per Minute

TASK PROFILE - DDEDIT

1. Waiting for Core (3 initiators)	= 120MS
2. Waiting for CPU (1)	= 1068MS
3. Waiting for Disk	= 96MS
4. Performing I/O	= 480MS
5. Executing CPU	= 100MS
6. Preempted (interrupted) by (different than waiting for)	
LOADER	= 36MS
RPDDT	= 13MS
QUEUE CTL	= 30MS
LINE CTL	= 10MS
TASK SWITCHING	= 19MS
	1.972 sec.

FIGURE 5

(1) Wait for other tasks as well as system routines.

After discovering the CPU as the prime bottleneck, a concerted effort was made to shave processing time from critical service routines which were responsible for a good deal of CPU processing during the periods of time when the applications programs also required the CPU. These routines are the disk device handler, the loader, the real-time clock handler, the scheduler, and the task switcher. By altering priorities, refining the service routines, and changing some of the applications programs, we were able to meet the specified performance goals.

Simulation modeling helped us to make the decision not to buy more core (increasing the two machines from 96K to 128K words) but rather to implement software changes to reduce CPU processing as described above.

Summary

The development of a simulation model for this real-time system was not only beneficial but necessary for developing an understanding of system performance. The insights obtained allowed us to meet specified performance goals for the system. In addition, it was possible to project the dispatch times under increasing alarm rates with different quantities of dispatchers. The saturation point of MICS was determined. This is the point where system response time becomes the limiting factor not the number of dispatchers.

Work has begun on the CMICS simulation model which will be completed for the design phase and used to make intelligent design tradeoffs and analyze projected performance. The CMICS system is even a greater challenge for simulation modeling, since it includes a communication network employing concentrators in each borough which control the message flow to the central computer. Obviously, we believe simulation tools should be considered in the development of any other large real-time system.

Acknowledgments

The author wishes to acknowledge the efforts of Dr. H. Montvilla, R. Scherma, C. Faris, J. Keenan and J. Fitzpatrick of Bradford National Corporation and J. Mohan, I. Steinberg and L. Railing of the New York City Fire Department.

Bibliography

1. An Environmental Simulator for FDNY's Computer Aided Dispatch System - J. Mohan, M. Geller - 2nd Internation Conference on Software Engineering - October 1976.

2. MICS Brooklyn GPSS Simulation - Dispatch Performance Projections - M. Geller, L. Railing - Internal FDNY document.

3. Design of Real Time Computer Systems - Prentice Hall - 1967 - J. Martin.

4. Real Time System Design - E. Yourdon - Cambridge Information and Systems Institute.