TICKETRON—A successfully operating system without an operating system

by HARVEY DUBNER and JOSEPH ABATE

Computer Applications Incorporated
New York, New York

INTRODUCTION

In recent years, industry has witnessed the proliferation of complex on-line systems. More and more, computer management is recognizing the need to employ scientific methods to assist in the complex tasks of hardware/software selection and evaluation. This is especially true for real-time computer systems. As is well known, the distinguishing feature of real-time systems is that they are prone to the most spectacular failures ever witnessed in the computer industry. In many installations, real-time systems have become “hard-time” systems. The specter of potential failure has caused users to realize the importance of designing first, installing later. The sophisticated user has become aware of the fact that the rules of thumb and intuition that adequately described simple batch-type systems do not suffice when one is concerned with real-time systems. Real-time automation demands a certain amount of expertise on the part of the designer and implementor. In fact, systems which have been installed without adequate pre-analysis, more often than not, wind up with:

- Too expensive a central processor
- Too many ancillary components
- The wrong number of I/O channels
- Too elaborate a Supervisory System
- Poor communications interface

Clearly, the salient point we wish to establish is that real-time systems have a tendency to cost far in excess than necessary. Typically, the inefficient use of hardware is the staggering cost factor which most dramatically degrades the performance per dollar of a real-time system. Of course, our concern of performance per dollar would be an academic issue if the effect of improper design were to cause increases in hardware costs of the order of 10%. However, we maintain that the situation is much more drastic and such systems suffer excessive hardware costs in the order of 100%.

The reason for this state of affairs is that the design of real-time systems is not an art but rather a scientific discipline. One must bring analysis to bear on the problems. To be sure, it is not the purpose of this paper to give a full treatment of this discipline. Rather, it is the purpose of this paper to demonstrate certain techniques and their application to a real-life system, TICKETRON.

TICKETRON is a real-time ticket reservation and distribution system for the entertainment industry. In many respects it resembles most other real-time systems, therefore, the discussions concerning this system are by no means unique to it. That is to say, the approach and attitudes developed in the design and implementation of TICKETRON represent our philosophy toward real-time systems in general. We believe that using a successful system such as TICKETRON as the vehicle for presenting our philosophy concerning real-time systems, adds substance to our arguments.

The ultimate aim of our arguments is the concern for maximum performance per dollar of a system. TICKETRON is successful because it did achieve excellent performance per dollar. Specifically, the “industry standard” for this type of system priced the central facility hardware at over $60,000 per month. Through proper design, TICKETRON was able to accomplish better performance for less than $30,000 per month.

At the heart of the problem is the frenzy associated with multiprogramming in real-time systems, causing the need for supervisory programs. There has been a tendency in the past few years to implement operating systems which are so elaborate that the amount of computer time used for message processing can be matched or exceeded by the amount of time required...
by the supervisor to maintain the job flow. In addition to the exorbitant overhead in time, there is the extra hardware cost associated with the inordinately large amount of dedicated core storage required by such operating systems. Further, a typical, modern-day, operating system presents itself as a labyrinth to the user who is required to make his application programs function in the unfamiliar and complex environment of the operating system. In most instances, the added burden on the user to cope with this labyrinth during program development and debugging is so costly in terms of manpower effort that it would have been far cheaper for him to have avoided trying to take advantage of the “standard supervisory package.”

In short, these problems reflect a major paradox associated with third generation computer systems: “How can an operating system that costs nothing be so expensive?”

At this point, the reader might feel that we have overstated our position. No doubt he is able to point to many systems having constraints such that they require an elaborate operating system. We agree. Certainly a large on-line system which must perform a multitude of tasks cannot function without a complex supervisory system. Our point, however, is that too often simple systems are designed as if they were complex systems.

To summarize our approach, we believe that simplicity is the keynote of a good system design. If there is no need to multiprogram, don’t! This is why TICKETRON is a successfully operating system without an operating system!

It is the intent of this paper to put forth the system design story for TICKETRON. The second section presents the design and the third section explains the design. The fourth section analyzes the design.

SYSTEM OVERVIEW

In addition to giving a functional description of the system, it is the purpose of this section to broadly specify the architecture of the system.

To begin, what is TICKETRON? It is a fully computerized ticket reservation and distribution system offering actual printed tickets at remote terminals. In short, it provides access to box offices from remote locations. In that sense, TICKETRON is an extension of the box office. It was originally intended to sell tickets for the entertainment industry. However, today it is also selling train tickets for the Penn Central Metroliner. The system is practical in any application which involves the issuing of tickets. Remote sales terminals are installed at high-traffic points such as shopping centers, department stores, etc., and, of course, at box offices. It is a nationwide service having separate systems, each serving a geographical area. There are three central facilities at present: New York, Chicago and Los Angeles. Each central facility can support almost 500 terminals which can accommodate sales of 50,000 tickets per hour without any difficulty, under certain conditions (see the fourth section).

A remote terminal consists of a dedicated keyboard, a ticket printer and a receive-only teletype. A customer desiring tickets approaches a remote station and makes an inquiry concerning the availability of a performance. The terminal operator interrogates the system via the dedicated keyboard and receives a response in seconds at the teletype. The teletype message indicates what seats are available, if any. Then, if the customer is pleased with the selection, the operator will cause the system to “sell” the seats. Within seconds the actual tickets are printed out by the ticket printer. These are real tickets and the customer pays for them as he would at a box office. Therefore, in a genuine sense the remote station is an extension of the box office. Direct-access to the total ticket inventory guarantees remote buyers the best available seats at time of purchase (this is done automatically by a seat selection program). In addition to selling tickets, the system provides certain key reports for management information and also accurate accounting of ticket sales.

TICKETRON is a typical real-time system in that it is composed of four major constituents:

1. the remote terminals with their communications network
2. the line controller and buffers
3. the processing unit and associated core storage
4. the auxiliary storage with its connecting data channels

Knowing that these functional elements are required in the system, one must then determine what hardware is best suited for the job. Hopefully, this selection should be made on a performance per dollar basis. In short, this is what systems design is all about.

In the third section, we discuss certain procedural concepts that we consider important for accomplishing an effective system design. Further, we present some findings obtained by executing these procedures for the TICKETRON system. The remainder of this section will be devoted to an overview of the hardware and software architecture of the system.

An important result is the actual hardware configuration that was decided upon for TICKETRON. It was found that the system should be dedicated solely to the on-line, real-time tasks required of it.
Further, it was found that the tasks were such that a process control type computer afforded the best performance per dollar in this situation. The computer system selected was a Control Data Corporation 1700. Figure 1 shows the central facility configuration. Reliability deemed it necessary that essential hardware be duplexed. The application is such that the system must be operative at certain critical times, for example, the peak ticket selling period just before a ballgame.

A result of the design shown in Figure 1 indicates that the TICKETRON system has two processors; one processor acts as a communications controller, while the other processes the messages. The front-end only handles the communication functions and contains the input and output line buffers. It does not examine the contents of the message. This last function is done by the message processing program which is resident in the central processor, which requests messages from the front-end. The communications program is resident in the front-end.

In addition to specifying the hardware, Figure 1 indicates the approximate characteristics of each device. The total monthly rental for the central facility hardware (including duplexing and maintenance) is about $30,000. We maintain that this is an achievement of understanding the characteristics of real-time systems and their design consequences.

As previously stated, the hardware configuration is a result of our design analysis. To be sure, the software design is not divorced from the performance analysis. In fact, one establishes certain programming considerations by analyzing their effect on the performance of the system. As a result, we decided on a single-thread program design. That is, at any one time there will be no more than one message in the system which is partially processed. There may be additional messages in the system, but these will be in one of two states; either awaiting processing (in input queue) or having completed processing (in output queue). We can use a single-thread programming concept because of the timings involved.

There are three major program modules: the communications program, the message processing program and an on-line utility program. The software design is such that each subroutine calls the next required subroutine. In essence, the system has one big program.

In considering the flow of a message through the system, we have the processing program receiving its messages from the input queue and after processing, delivering them to the output queue. The processing program is a single-thread program which deals with one message at a time. That is, it only accepts another message for processing after it has delivered one to the output queue. The processing program determines the next message to be processed as follows: it scans the input queue. When it finds a full buffer, it first checks the output queue to see if a buffer corresponding to that line is empty; if not, it will carry out the procedure for another line. If the processing program finds no candidates to be processed, it will then exit to what one might call a main scheduler program which does nothing but loop-the-loop. When the processing program has completed a message, the procedure is for it to loop back on itself. When it has work, it starts its cycle over again and does not return control to the main scheduler. That is, during a busy period, the processing program is continually looping; further, it is in complete charge since it has no open branches. In fact, during this time, the processing program has all the characteristics of an executive routine. However, in actuality the concept of “the executive” is foreign to this system.

The communications program, which is resident in the front-end processor, simply fills and empties the input and output line buffers, respectively, for each communications line. The details of its operation are given in a later section, because the communication discipline was an integral part of the system design.
All programming on the system was done in FORTRAN because it offered the following advantages:

(a) Minimize programming costs and time.
(b) Machine independent.
(c) Partially self-documenting, no patching.
(d) Easy to modify.
(e) Easy communication between subroutines (and between programmers writing the subroutines).

History and evolution of the system

The TICKETRON system was conceived in January, 1967. A pilot system using a CDC 160A computer and modified teletype as ticket printer was used to demonstrate the feasibility of the system and to gain practical experience with automated ticket selling. The pilot ran from July to November, 1967.

Although TICKETRON has not varied much in concept over the years, the size and requirement of the system have undergone evolutionary changes. As a result, there have been three different computer equipment configurations.

1. Initially, TICKETRON was aimed primarily at the New York legitimate theater and advance sale for some sporting events, with 50 to 100 terminals selling 50,000 tickets per day. The first operational on-line configuration consisted of a CDC 1700 with 16K of core, 2 disks each with 3.1 million words, 2 tapes, a console teletype, and communications hardware sufficient for interfacing 16 voice grade (1200 baud) lines. This equipment was duplicated for reliability, with a printer and paper tape reader-punch added for off-line work. This system went operational in March, 1968.

2. It soon became apparent that sporting events were more important than originally anticipated, and that more terminals would be required. The number of lines was increased from 16 to 32 and the core increased from 16K to 32K. This allowed the system to handle over 500 terminals and well over 100,000 tickets per day. In addition, a drum with 256K words was added for high frequency files, in particular for the inventory needed for selling same day sporting events. This system became operational in January, 1969.

3. As terminals become more distant from the computer center, communication costs can be dramatically reduced by using communication concentrators. These city buffers are actually computers which perform "intelligent" functions such as formatting tickets. Since this required redesigning of the whole communications program, we took advantage of new technology and replaced the communication hardware by a computer front end with 20K of core at no increase in cost. This configuration can handle the equivalent of 56 phone lines and almost 900 terminals. Also, to accommodate more events two more disks are being added and the size of the drum increased to 512K words. This system will be operational in the spring of 1970. The analysis performed in this paper assumes this configuration.

SYSTEM DESIGN

We maintain that a successful system design is achieved by understanding the characteristics of real-time systems and their design consequences. The procedure for accomplishing this is essentially an iteration scheme:

(1) Specify a configuration; that is, define a prototype model of the system.
(2) Evaluate the configuration as to its operational characteristics. Essential to this step is a performance analysis which determines the capabilities and limitations of the prototype system.
(3) Make design recommendations on the basis of the evaluation.
(4) Are the recommendations substantial? If yes, continue. If no, end.
(5) Incorporate the recommendations by updating the system model. Then start again.

In this section, we shall present some results obtained by executing the above system design procedure for TICKETRON. At the heart of the procedure is the performance analysis.

TICKETRON typifies the operational characteristics of a real-time system. Namely, it is representative of a stochastic service system. The situation encountered is that the inputs to the system occur randomly and generate service requests of the central processor which are varied in type. In a poorly designed system, these random phenomena cause queueing and congestion problems. Therefore, a performance analysis which takes account of the random phenomena is essential to the design effort. The considerations of this approach manifest itself as follows: in a steady-state operation, the throughput of the system is defined as the average number of input messages to the system per second. Certainly then the throughput requirement is an important design criterion. Concurrently, with the throughput considerations is the requirement of a tolerable response time to each input message. In a given system, the situation usually encountered is that of having a desirably high throughput which, in turn, causes a system request queue to build up, thereby degrading response time. Therefore, pertinent to the system design is a knowledge of throughput versus
from the collection of the computer history museum (www.computerhistory.org)

From the collection of the Computer History Museum (www.computerhistory.org)

This front-end processor is a CDC-1774 CPU which is essentially a stripped-down CDC-1700 computer. It has 20K words of core memory available for the program and I/O line buffers. (In this system a computer storage word is composed of 18 bits: 16 data bits, 1 parity bit and 1 program protect bit.)

 Associated with each line is essentially four buffers, two for input and two for output. Actually, one of the input buffers is in an area which contains ten buffers shared by all the lines. As we shall see below, the system will only issue a poll on the line if one of the input buffers is available. That is, you only take a message in if you have room for it. As was previously discussed, the message processing program will only start processing a message if there is an output buffer available. Hence, at any one time the system generally will contain no more than four messages for a particular line.

Therefore, because of this “throttling” effect, one need not program for, nor worry about the problems associated with excessive internal queueing.

Each line is full duplex and transmission is done asynchronously. The system outputs 9-bit characters (1 start bit, 6 data bits, 1 parity bit, 1 stop bit) at 1200 baud or 7.50 msec per character, and inputs 7-bit characters (1 start bit, 4 data bits, 1 parity bit, 1 stop bit) at 800 baud or 8.75 msec per character. The reason why only 4 data bits are required on the input side, is that all input messages are in the form of a restricted fixed format. The terminal input device is a dedicated keyboard with dedicated columns allowing the entry of such pertinent information as: event code, performance date and time, and certain seat qualifiers. Associated with this data is one of three function codes: inquiry, buy, or buy alternate. The buttons on the keyboard are such that they only permit one per column or function to be depressed at any one time. Therefore, every input message to the system is of fixed size, 19 characters. The advantages of this scheme as to programming and operation are obvious.

The communication program uses a polling technique that can uniquely address each terminal in the system. The poll message uses four characters. The system uses no “hand-shaking” characters such as ACK or NACK. A poll to a keyboard causes it to transmit if the transmit button is depressed. This is accomplished in about 200 milliseconds. If the transmit button is not depressed when it receives at poll, it sends no response. The communication program will allow the terminal 200 milliseconds to respond before it infers a non-acknowledgment.

The communication program will perform the following logic for each communication line on a periodic execution cycle.

1. Check disposition of the “receive” line buffer,
three possibilities exist: free, full, or busy (200 milliseconds have not elapsed since last poll).

2. If free, check if there is available space for a message in an input message buffer. If the space exists for an input message, then prepare a poll message for the next non-busy terminal on that line. Go to 5.

3. If full, check for transmission errors, then move the message to the input message queue (space will always be available). Once this is done, the "receive" line buffer is free, go to 2.

4. If busy, go to 5.

5. Check disposition of the "send" line buffer, two possibilities exist: free or busy.

6. If free, check for message in the output buffer for this line, if there is one, send it, otherwise done. (Start algorithm again at 1 for next line.)

7. If busy, done. (Start algorithm again at 1 for next line.)

We purposely did not clutter the algorithm with implementation details since it would only cause to mar the simplicity of the scheme. For instance, the output required for the printing of a ticket is transmitted in four segments. Interspersed between the segments may be poll, light and TTY messages for other terminals on the line. Further, it is clear that the scheme requires use of certain kept tables which reflect terminal and line activity.

The basic philosophy in the given design of the communication program has been to maintain the integrity of its true function. That function being, that it is simply an intermediary between the message processing program and the terminals. Speaking loosely, it should be synchronized with the actions of the message processing program. In fact, since it is ultimately the responsibility of the processing program to respond to the terminals, then the communication controller should only react to the needs of the message processing program. For example, if at any given time the message processing program has enough work, then there is no need for the communication program to poll terminals for the purpose of bringing in more messages. To do otherwise would be illogical.

PERFORMANCE ANALYSIS

In this section, we give the results of a queueing analysis of the system in order to determine its capabilities and limitations. As argued in the second section, response time versus throughput is the basis in terms of which to measure the performance of the system. The throughput capability of any system is determined by certain utilization factors.

Utilization factor is a well defined mathematical concept of queueing theory. Given a facility which has some random arrival pattern for requests such that the average input rate is \( \lambda \) arrivals per second, then let each arrival place a demand on (tie-up) the facility for some average time, \( T_s \), seconds. That is, during the time \( T_s \) (service time), the facility is not available to any other arrival. Then we have the utilization factor of the facility defined by

\[
U = \lambda T_s
\]

In short, it represents the percentage of time the facility is tied-up. Obviously, \( U \) should not exceed 100%!

For TICKETRON, the throughput is measured in terms of the number of tickets per hour that the system is capable of outputing. These determinations start with a specification of the input traffic to the system. We distinguish two types of terminals: box office and remotes. Remotes print what we call full-tickets (308 ch of data). In addition, box office terminals are capable of also selling half-tickets (119 ch of data), which are useful for same-day events. Remotes can only buy tickets after an inquiry has been previously executed, whereas, a box office may execute a direct buy. The reason for this is that box office attendants are more familiar with their own inventory and therefore, have little need to make inquiries. The characteristics of remotes are such that they average 1½ inquiries for each buy transaction. In contrast, at a box office you have on the average about 3/4 of an inquiry per buy transaction. A buy transaction requires on the average the printing of three tickets. Table I gives a distribution of the number of tickets sold per transaction. Because most tickets are bought in pairs, the distribution is "tight" about the average of three, as verified by the small squared coefficient of variation.

<table>
<thead>
<tr>
<th>Number of tickets sold per buy transaction</th>
<th>Distribution (Percent of occurrence)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>10%</td>
</tr>
<tr>
<td>2</td>
<td>49%</td>
</tr>
<tr>
<td>3</td>
<td>10%</td>
</tr>
<tr>
<td>4</td>
<td>18%</td>
</tr>
<tr>
<td>5</td>
<td>4%</td>
</tr>
<tr>
<td>6</td>
<td>4%</td>
</tr>
<tr>
<td>7</td>
<td>1%</td>
</tr>
<tr>
<td>8</td>
<td>2%</td>
</tr>
<tr>
<td>9</td>
<td>2%</td>
</tr>
</tbody>
</table>
which equals .34. Therefore, for calculational purposes, we may consider this a constant distribution and make use of the simple queueing formulas associated with constant service.

A typical operational day for TICKETRON is represented by sales of about 120,000 tickets, which is equivalent to 40,000 transactions. Unfortunately, this traffic is not distributed evenly throughout the day, hence, the system must be able to accommodate peak traffic loads. Figure 2 depicts histograms of the hourly rate of ticket sales, which we shall use to establish peak traffic loads. We note that the remote sales are evenly distributed at 4,000 tickets per hour over a ten hour day and they account for $\frac{1}{3}$ of the total load. Of these 4,000 tickets per hour, we estimate that one-tenth or 400 are for same-day events while 3,600 are for future events. In contrast, the box offices have sharp peaks for $\frac{1}{2}$ hour periods just before afternoon and evening performances. It is estimated that the box offices sell 2,000 tickets per hour for future events evenly over a ten hour period, which accounts for 20,000 tickets; whereas, the other 60,000 are for same-day events and are sold over a three hour period at an even rate of 20,000 per hour. These same-day events are usually sold as direct buys, hence, on the average we estimate that they cause only about .2 inquiries per sale. We note that since $\frac{1}{2}$ of the sales average .2 inquiries per sale and $\frac{1}{2}$ average 1.5 inquiries per sale, then on the average a box office has about $\frac{3}{4}$ of an inquiry per buy transaction. In summary, during a peak hour, the box offices sell 22,000 tickets while the remotes sell 4,000.

Because most of the peak traffic represents sales for same-day events, the system keeps this inventory on high speed drums for fast retrieval. The processing service times for the various types of transactions are given in Table II. These timings are almost a constant independent of the number of tickets, therefore, we will assume them constant. The timings include all I/O times which are not overlapped with the processing, since the main processing program is a single-thread routine.

At this point, it is of interest to demonstrate how the processing times limit the throughput of the system. As argued in an earlier section, the inputs to the system are random, in fact we maintain that they generate a Poisson arrival stream to the processor. This phenomenon causes queueing of the inputs. Therefore, the total processing time or cpu response time must include waiting time for the cpu. The simple queueing formula which determines the average waiting time for a single-server queue with Poisson input rate $\lambda$, mean service time $T_s$, and second moment of the service time $b_2$, is given

\[
W = \frac{U}{2(1 - U)} \left( \frac{b_2}{T_s} \right)
\]

where $U$ is the utilization factor which is determined by eq. (1). To be sure, $W$ becomes intolerably large as $U$ approaches 100%, which is the limitation that governs the capability of the system. Hence, to obtain the cpu response time for a particular type of input, we just add its service time to the waiting time $W$. Let us now determine the average cpu response time for three different operational environments.

**Case I. (Box office peak hour)**

Assume for this case that all inputs to the system are direct buy transactions from box offices for same-day events. One may envision this situation to prevail for a period of about an hour on a day when every baseball team has an afternoon game and the remotes are closed. (Memorial Day is an example of such a day.) It is reasonable that in this environment there will be a

<table>
<thead>
<tr>
<th>Type of Transaction</th>
<th>Processing Time in Seconds</th>
</tr>
</thead>
<tbody>
<tr>
<td>Inquiry</td>
<td>.135</td>
</tr>
<tr>
<td>Buy following an Inquiry</td>
<td>.099</td>
</tr>
<tr>
<td>Direct Buy</td>
<td>.180</td>
</tr>
</tbody>
</table>

**TABLE II—The Processing Service Times for Various Types of Transactions**
negligible number of inquiries and selling for future events. From Table II, we have the cpu service time for this case as $T_s = 0.180$ seconds. Since the service time is constant, the second moment is equal to the mean-squared. Let $\lambda$ equal the average number of input transactions per second or in this case, it is equal to average rate of buy transactions. Therefore, the cpu utilization and response time are given

\begin{align*}
U &= \lambda (0.180) \quad (3a) \\
T_{cpu} &= \frac{U}{2(1 - U)} (0.180) + (0.180) \quad (3b)
\end{align*}

Figure 3 is a graph of this result. Because each buy transaction represents an average sale of three tickets and since there are 3,600 seconds in an hour, then $(10,800)\lambda$ represents the number of tickets sold per hour. This quantity is also given in Figure 3. We observe that in this environment the system can sell in the order of 50,000 tickets per hour.

Case II. (Remote peak hour)

In this case, let us assume that all the inputs to the system are generated by remote terminals and are sales for future events. This situation may very well occur on certain rare days which have very few events and during the time that box offices are usually not active. Since remote terminals cannot make a direct buy, the input transactions in this case will consist of inquiries and buys (only after an inquiry). Hence, there will be at least one inquiry for each buy, but in fact, we maintain that on the average there will be $1.5\lambda$ inquiries for each buy transaction. Therefore, if $\lambda$ is the number of buys per second, then $1.5\lambda + \lambda$ equals the rate of input transactions to the system. From Table II, we calculate the mean cpu service time for a transaction to be $(0.6)(0.388) + (0.4)(0.248) = 0.302$ seconds. Hence, the cpu utilization in this case is given

\begin{equation}
U = (2.5\lambda)(0.302) = \lambda(0.755) \quad (4a)
\end{equation}

As before, the number of tickets per hour equals $(10,800)\lambda$. The second moment of the service time equals $0.093$ seconds-squared. Therefore, the average cpu response time to a buy transaction in this environment is given

\begin{equation}
T_{cpu} = \frac{U}{2(1 - U)} \left( \frac{0.093}{0.302} \right) + (0.248) \quad (4b)
\end{equation}

Figure 4 is a graph of this result. We observe that in this environment the system can sell in the order of 10,000 tickets per hour.

It is interesting to note the contrast of this result with that given for the environment in Case I. We may consider these two cases as extremes to which the system can respond. Let us now turn to a study of the
system which more closely corresponds to a realistic environment to which the system is subjected more often.

**Case III. (Realistic Peak Hour)**

Let us assume an input traffic mix as determined from a peak hour of the histograms given in Figure 2. Specification of this mix, as discussed above in this section, establishes the profile given in Table 3. Again, \( \lambda \) equals the average rate of buy transactions. We find that the total number of inputs per second is 1.5\( \lambda \). Also, we find that the mean cpu service time for this traffic is .209 seconds. Therefore, the cpu utilization is given

\[
U = (1.5\lambda)(.209) = \lambda(.314) \tag{5a}
\]

As before, the number of tickets per hour equals (10,800)\( \lambda \). The second moment of the service time equals .050 seconds-squared. Therefore, the average cpu waiting time for processing in this environment is given

\[
W_{\text{cpu}} = \frac{U}{2(1 - U)} \left( \frac{.050}{.209} \right) \tag{5b}
\]

In Figure 5, we graph the following two functions: \( W_{\text{cpu}} = .099 \) and \( W_{\text{cpu}} = .248 \) which represent the average cpu response times to a buy following an inquiry for a same-day event and a future event, respectively.

Figure 5 shows that as far as cpu processing is concerned, the system can accommodate peak hour sales of 26,000 tickets in the environment specified by the histograms of Figure 2 which represents possible traffic distribution for sales of 120,000 tickets in a day. In a sense, this situation is to be expected because typically, the limitations for a system of this sort are not determined by the cpu processing capability but rather by the congestion of the communication lines. Further, since the output transmission dominates the communication load in such systems, it is the traffic on the output line and the number of such lines which truly govern the throughput capability of the system. For example, since the output transmission time required to print a full ticket is about 4 seconds, then the theoretical maximum that a line can output is 900 tickets per hour. Hence, if the system only had 16 lines, it could not achieve the throughput levels required. Therefore, due to these considerations, we shall now investigate the system in terms of the activity on a single line.
We shall now determine the response time for a terminal on a specified communication line in terms of the output utilization and the output rate of tickets for that line. For this system, we shall define the response time as the elapsed time between the initiation of a request and the start of output data transmission. The timing schematic of Figure 6 illustrates this quantity. Further, the timing diagram depicts the delays encountered by the passage of a particular request through the system.

Hence, the system response time is the sum of four random variables: the waiting time for the terminal to be polled, plus the time required to transmit the input data, plus the cpu time required to process the input request, plus the waiting time for the output line to become free in order to commence transmission. The response time is given by

\[ T = W_{\text{pol}} + T_{\text{in}} + T_{\text{cpu}} + W_{\text{out}} \]  

(6)

We have a system of three queues in tandem, input, processing and output. Essentially, the response time is figured by calculating the delays encountered at each queue and convolving the results. The procedure and assumptions required for such an analysis are given in reference.\(^3\)

Unfortunately, in this analysis there is one slight complication; namely, the phenomenon of blocking for queues in tandem. For two queues in tandem, blocking occurs when a customer completes his current service but cannot move to the next queue because its limited waiting line is filled. Thus further service in the first queue is blocked until the second queue completes a service. If we recall the working of the communications program, it is such that polling is only performed if there is space in the input queue for a transaction, otherwise the system does not poll terminals for requests. That is, if at any given time the message processing program has enough work (the input queue is full), then there is no need for the communication program to poll terminals for the purpose of bringing in more messages, since to do so would serve no purpose.

This situation is unfortunate only for the analysis of the system, however, it is a real advantage in terms of system operation because it permits the system to optimize resource allocations concerning input message storage. Now because of buffer allocations, there are always enough transactions in core for processing and outputing, therefore, this blocking causes no forced idle time of the second and third queues. Hence, in the resulting system a request has the same total response time as it would have in a system where blocking of this sort were not present. In other words, if we were to calculate the individual terms on the right hand side of equation (6) assuming no blocking, the individual quantities would not represent the TICKETRON system, however, the total does, and is a valid representation for the response time of the TICKETRON system.

To this end, we now calculate the individual terms in equation (6) for a system which allows unlimited queues and continuous polling. We shall perform the calculation for two different configurations of output lines.

In the first configuration for an output line we shall assume that it has 12 remote terminals attached to it, all selling full tickets for future events. Further, we assume that the cpu is processing in the environment as specified in Case III. We want the response time to a buy transaction for a future event. Hence, for the cpu processing time we use equation (5b), giving \( T_{\text{cpu}} = W_{\text{cpu}} + (.248) \). We assume that the throughput level of the cpu is at 26,000 tickets per hour, this corresponds to a cpu utilization of .756. Then from the appropriate graph in Figure 5, we find that for the environment being considered, \( T_{\text{cpu}} = .62 \) seconds.

Let us next calculate the delays encountered on inputing. First, we have the elapse time a terminal must wait in order to receive a poll once the transmit button on the keyboard has been depressed. We recall that for calculational purposes we consider a system with continuous polling. Therefore, we may envision that the 12 terminals on the line are being polled cyclically at the same rate. Because the pointer can be anywhere in the list when the terminal initiates a request for transmission, that particular terminal must wait on the average \( N/2 \) polling times before it receives a poll, where \( N \) is the number of terminals on the line. As dis-
cussed in Section 3, the communications program will successively poll terminals on a line every 200 milliseconds, provided the output line is free to send out a poll message. Therefore, when the line is free, the wait for a poll is \((N/2)(.2)\) seconds. On the other hand, when the output line is busy transmitting tickets, the polling rate is variable as follows. The transmission of a ticket is accomplished in four parts. The first operation is to send a short message which slews the ticket into the printer. The slew time is 1.2 seconds, however, during most of this time the line is free to permit polling at the normal rate. Hence, during slew time, we can send out five polls. Next, the data required to print the ticket is transmitted in three equal segments, and between each segment a poll message is sent (in addition to teletype and light messages). In the printing of a full ticket, each segment requires transmission of 103 data characters plus 4 control characters, which takes about .8 seconds. Therefore, to estimate the average time between polls during full ticket printing, we observe that we can send out 8 polls in about 4 seconds, which gives an average polling time of .5 seconds. Then when the line is busy, the wait for a poll is \((N/2)(.5)\) seconds.

Having found the average contribution to \(W_{\text{poll}}\) for free and busy conditions of the line, we now form a weighted sum to obtain the desired result. The weighting factors are expressed simply in terms of the utilization of the output line \(U_{\text{out}}\). Since \(U_{\text{out}}\) is the average percent of time the line is busy, then \((1 - U_{\text{out}})\) is the average percent of time the line is free. Therefore

\[ W_{\text{poll}} = (1 - U_{\text{out}})(N/2)(.2) + U_{\text{out}}(N/2)(.5) \]  

(7a)

where, using \(N = 12\) gives

\[ W_{\text{poll}} = 1.2 + 1.8U_{\text{out}} \]  

(7b)

Following the poll delay, is the time required for input transmission, \(T_{\text{in}}\). As discussed earlier, the input message is of fixed size of 19 characters. At 8.75 msec. per ch., the time to transmit is .167 seconds, however, added to this is 70 milliseconds required to turn on the modem. Therefore, \(T_{\text{in}} = .237\) seconds.

The final delay to be calculated is the waiting time encountered when the system is ready to transmit the data necessary to print tickets. Naturally \(W_{\text{out}}\) is a function of the line utilization and the service time required to print three tickets. As mentioned previously, we assume for calculational purposes that each buy transaction is for three tickets. We now determine the service time on the output line caused by such a transaction. First, there is the actual printing time for a ticket. This is equal to the transmission time since the printer prints at line speed of 1200 baud or 7.5 msec. per character. One full ticket requires 308 characters of data, plus 16 control characters, plus 1.2 seconds of slew time, which totals to 3.63 seconds. Therefore, three tickets take 10.89 seconds. However, this is not the only contribution to the traffic on the output line.

In addition to tickets, there are poll, teletype and light messages also being transmitted over the output line. As discussed above in connection with polling, the transmission of a ticket is segmented into four parts. The other messages are transmitted interspersed between the segments. Therefore, between each data segment we may figure on one poll message, one light and about two teletype messages being sent. Because the teletypes print at a much slower speed than 1200 baud, each teletype has a six character buffer. Hence, a teletype message is sent out in blocks of ten characters, six data plus four control characters. The teletype traffic accounts for the response to inquiries and audit trail associated with each transaction. We estimate that a teletype response to an inquiry requires 72 characters of data or 12 blocks of TTY messages, and an audit trail requires 12 characters of data or 2 blocks of TTY messages. Therefore, each buy transaction causes 20 blocks of TTY messages to be transmitted, since there are on the average 13½ inquiries per buy. Added to this traffic are nine poll messages and nine light messages each of four characters, for each buy transaction since the three tickets have nine data segments. (Note, that the poll messages sent out during slew time are overlapped.)

Hence, in addition to the ticket printing time for a buy transaction, we must add the transmission time for 272 characters which is 2.04 seconds to take account of the other activity on the output line. Therefore, the service time on the output line is 12.93 seconds for a buy transaction. If \(\lambda\) equals the average number of such transactions per second, then the utilization factor for the output line, \(U_{\text{out}} = \lambda(12.93)\). As before, \((10,800)\lambda\) represents the number of tickets sold per hour over the line. Hence, the average waiting time in this case is given

\[ W_{\text{out}} = \frac{U_{\text{out}}}{2(1 - U_{\text{out}})} \]  

(12.93)  

(8)

Finally then, we accumulate these quantities to determine the response time as prescribed above using equation (6). We have that the average response time for this configuration is given by

\[ T = (1.2 + 1.8U_{\text{out}}) + (.24) + (.62) \]

\[ + \left[ \frac{U_{\text{out}}}{2(1 - U_{\text{out}})} \right] \]  

(12.93)  

(9)
Figure 7—Average system response time for a buy transaction as experienced by a remote on an output line which has 12 remote terminals all selling full tickets. Also included is an estimate of the 95% system response time for this case; that is, the response time which is not exceeded by 95% of the transactions.

Figure 7 includes a graph of this result. We observe the validity of the conjecture stated initially; namely, that the determination of the response time is dominated by the output transmission time. In other words, the contribution or congestion caused by the third queue (output) essentially determines the response time. Because of this, we may use a simple argument to estimate the 95th percentile of response time, that is, the response time which is not exceeded by 95% of the transactions. We argue that the system response time distribution is estimated by the waiting time distribution of the output queue which happens to be a queue with constant service time. For such a queue, we know the relationship that exists between the 95th percentile of waiting time and the average waiting time. Therefore, we use the same relationship to estimate the 95% system response time from the average value. A graph of this estimation is also given in Figure 7.

Having considered the case for an output line with 12 remote terminals all selling full tickets, let us now turn to what might be considered a more efficient situation. Assume an output line configuration with 2 box office terminals selling \( \frac{1}{2} \) tickets and 10 remote terminals selling full tickets. Again we have 12 terminals on the line, except in this case the box offices generate less line utilization per ticket sold, so that the throughput of the line is increased. Also, a box office terminal is polled twice as often as a remote terminal. (In fact, the polling sequence used by the system for this case is as follows: \( \cdots, R_1, R_2, R_3, R_4, R_5, B_1, B_2, R_6, R_7, R_8, R_9, R_{10}, B_1, B_2, R_1, \cdots \) where \( R \) designates remote and \( B \) designates box office.) Therefore, the box offices will have a better response time. Further, we assume that the ticket mix on the line is such that 50% of the sales are \( \frac{1}{2} \) tickets and 50% are full tickets. That is, the two box office terminals sell as much as the ten remotes since they are the only ones capable of selling \( \frac{1}{2} \) tickets.

We shall calculate the response time for a direct buy transaction as experienced by a box office. The calculations for this case are similar to those presented above. We find the response time given by

\[
T = (.70 + 1.05U_1 + .18U_2) + (.24) + (.47)
\]

where

\[
U_{\text{out}} = \frac{U_{\text{out}}}{2(1 - U_{\text{out}})} \left( \frac{112.9}{10.3} \right)
\]

(10a)

(10b)
That is, the output line utilization factor has two contributions, $U_1$ and $U_2$ which are the utilization factors due to transmission of full tickets and $\frac{3}{2}$ tickets, respectively. Figure 8 depicts the average and 95% response time for this case. We observe that for an average response time of 10 seconds and a 95% response time of 40 seconds the throughput of the line is limited to 630 tickets per hour. Whereas, in the previous configuration of a remote selling full tickets, the same response time only afforded the line to output 430 tickets per hour. Therefore, the use of $\frac{3}{2}$ tickets permits the system to sell more tickets per line. In fact, the more efficient configuration will allow the system to sell 26,000 tickets per hour using 41 lines such that the average response time is about 10 seconds.

REFERENCES

1 A WEINGARTEN
   *Analyzing a real-time system*

2 P H SEAMAN
   *Analysis of some queueing models in realtime systems*
   IBM Manual No F20-0007 IBM Corporation Poughkeepsie
   New York 1965

3 J ABATE H DUBNER S WEINBERG
   *Queueing analysis of the IBM 2314 disk storage facility*

4 J ABATE H DUBNER
   *Optimizing the performance of a drum-like storage*
   IEEE Trans on Computing Vol C-18 No 11 pp 992–997
   November 1969

5 W P MARGOPOULOS R J WILLIAMS
   *Characteristic problems on teleprocessing system design*
   IBM Syst Journal Vol 5 No 3 pp 134–141 1966