M80-7
JSC 16494

Some Queuing Network Models of Computer Systems

E. S. Herndon

MARCH 1980

CONTRACT SPONSOR NASA/JSC
CONTRACT NO. F19628-80-C-0001 T-8231G
PROJECT NO. 8470
DEPT. D-74

THE MITRE CORPORATION
Houston, Texas

ABSTRACT

Queuing Network Models of a computer system operating with a single workload type are presented. Programs which operate on the Texas Instruments SR-52 programmable calculator are included.

TABLE OF CONTENTS

LIST OF ILLUSTRATIONS

LIST OF TABLES

SOME QUEUING NETWORK MODELS OF COMPUTER SYSTEMS

Queuing network models provide a basic tool for understanding computer systems and predicting how they will perform.

The use of networks of queues to describe what is going on inside the computer is a relatively old idea, but its widespread application to practical problems has only recently taken hold. In September of 1978 the ACM devoted a special issue of Computing Surveys to Queuing Network Models of Computer Systems Performance. The issue contains eight outstanding articles: the editor's overview, three tutorials, three application notes and an assessment of the field of analytic modeling. The excellent tutorial by Denning and Buzen [1] provides a point of departure for this paper.

In an earlier paper by this author [2], conventional Markov modeling techniques were used to develop a simple model of n terminals dealing with a single server system. A program for the Texas Instruments SR-52 programmable calculator was presented in that paper. The very compact algorithms presented in the tutorial by Denning and Buzen provided the inspiration to attempt more complex models on the SR-52 programmable calculator. Four programs are presented in this paper. They provide a capability to handle a large number of closed network, single workload problems.

In modeling terminology closed systems are systems in which there is a limited population of jobs; they are called closed because jobs don't enter and leave but continue to circulate within the system. Most real computer systems deal with limited job populations because there are limited facilities for handling jobs; interactive job populations are limited by the number of terminals attached to the system; batch jobs may be limited by available job input storage space; both are limited during execution by fixed amounts of main memory or software imposed multiprogramming limits. Thus models of closed systems are most appropriate to handling these real system environments.

The computational requirements for network queuing models increase with the complexity of the system being modeled. The simplest and easiest closed system models have two servers, a single workload and up to perhaps three jobs active; pencil, paper, and patience are sufficient computational resources to handle these models.

For larger job populations - up to perhaps six or seven - an inexpensive calculator with three memory registers can replace the pencil and paper. Here the limit - six or seven - is established by the stamina and dexterity of the analyst. The job population can be arbitrarily large and be accommodated on a programmable calculator with as few as 10 memory registers and 200 program steps. (This was shown in [2].) In this paper, still dealing with an arbitrary job population and a single workload, the central server system may consist of up to six separate devices ... or five devices one of which may have a load dependent service time. (The load dependent service time function is restricted to a simple function of the number of jobs in the queue.)

This size of problem can be handled with the 20 registers and 224 program steps available on the SR-52. This central system model requires two memory locations per device plus seven or eight locations for other variables and indices.

The marvelous thing about all of this is that the algorithm developed by Buzen (and used in these programs) implicitly enumerates all of the system states which can occur for n jobs visiting k devices, and solves the associated equations. A system state is any unique distribution of the number of jobs at each device in the system. The number of ways n jobs can be distributed among k devices is given by the expression:

$$L = \frac{(n + k - 1)!}{n!\ (k - 1)!}$$

The result for a central system with five devices and a population of 20 jobs, is 10,626 states. Solving the resulting 10,626 linear equations by brute force techniques would require tens of thousands of memory locations to manage the problem. With Buzen's algorithm (and a modest twist added by this author) any single workload problem can be handled with two locations per device plus about eight overhead registers. (Note: The main benefit of Buzen's fast algorithm is the reduction in numbers of arithmetic operations required to enumerate and solve the equations. From the viewpoint of storage the algorithm Buzen describes actually requires one location per device plus one location per job plus overhead. The twist added to further compact the required storage is to evaluate the matrix row by row instead of column by column. On the SR-52 this means an unlimited job population can be handled with a maximum of six devices.)

Much more powerful and sophisticated tools are required to handle multiple load dependent servers, multiple classes of jobs, and a variety of queue service disciplines. The BEST/1 program offered by BGS Systems and the CADS program offered by Information Research Associates are two such tools; they require tens of thousands of memory locations for instructions and data space, also they run on large scale computer systems.

In today's world of programmable calculators the Texas Instruments SR-52 has been replaced by the TI-59. It provides roughly twice the capacity for the same price. The programs presented in this paper can be easily converted for use on the newer TI-59. This newer calculator provides sufficient space to tackle some simple two-workload problems and will be the host for future model developments by this author.

THE CLOSED SYSTEM MODELS

Four programs have been developed to aid in the analysis of closed queuing networks.

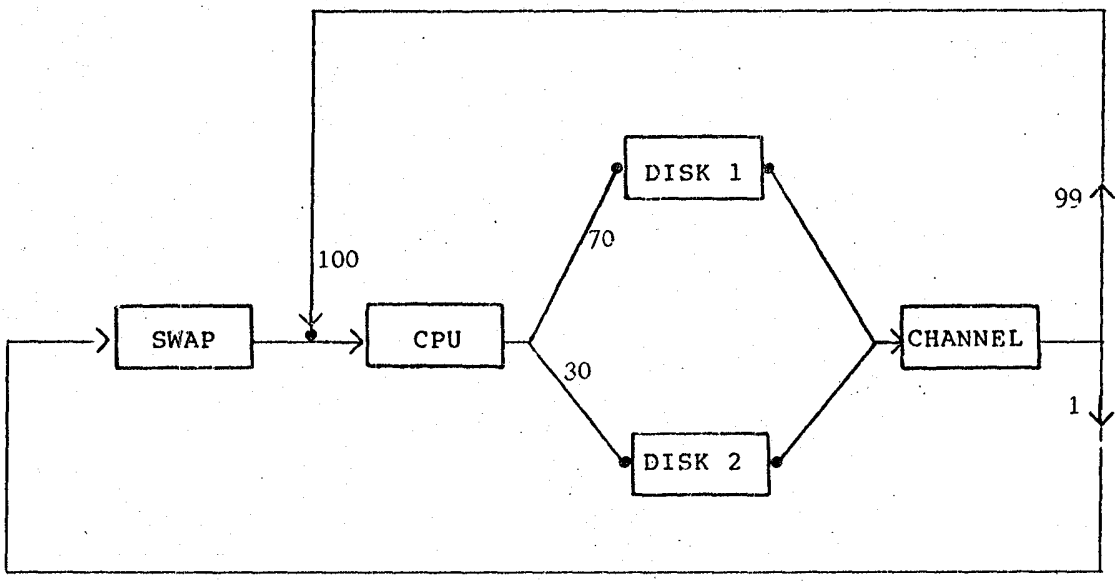
1. Batch model with homogeneous service times
2. Batch model with one load dependent server
3. Interactive model with up to five devices
4. Interactive model with a load dependent central server

The two programs for batch models will be discussed together since there are only minor variations between the two. Then the interactive models will be presented.

THE BATCH MODELS

In order to introduce nomenclature and demonstrate how these may be used a sample problem approach is taken. Figure 1 illustrates five servers in a batch processing system. At the bottom of the figure is a table showing the average job's characteristics. The typical job visits the swap device one time per job and requires 0.8 seconds to swap the job in. The job visits both the CPU and the channel 100 times; once for each disk input/output. Disk 1 gets 70% of the traffic. Disk 2 gets 30%. The service time per visit is shown for each device. The numbers which are needed in the model are the total service times for the job at each device, $Y_k = V_k S_k$. The CPU at 4 seconds of total service carries the heaviest load and will be the device which ultimately limits throughput.

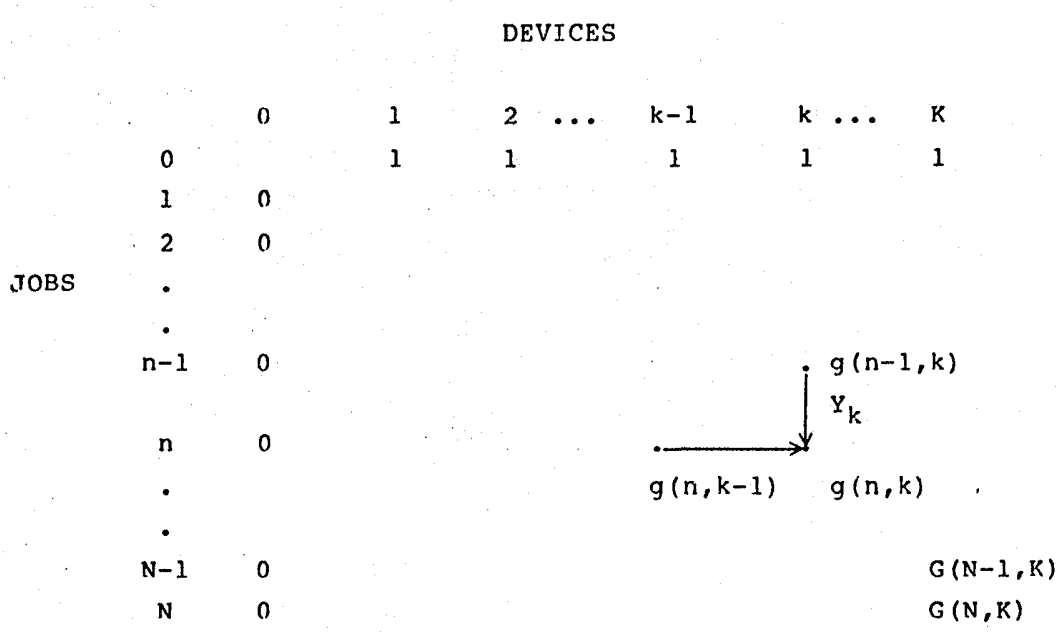
Figure 1
Sample Problem - A Batch Processor



JOB CHARACTERISTICS

DEVICE NAME	DEVICE NO. k	NO. OF VISITS V_k	TIME PER VISIT S_k	TOTAL SERVICE Y_k SEC.
Swap	2	1	.8	.8
CPU	1	100	.040	4.0
Disk 1	3	70	.030	2.1
Disk 2	4	30	.030	.9
Channel	5	100	.012	1.2

The Buzen algorithm fills in numbers in a two-dimensional matrix G. Columns in the matrix correspond to devices in the system and rows to the number of jobs. Elements of the matrix are computed from the adjacent elements, above and to the left, as shown in the figure below. Initially the first row contains 1's and the first column contains 0's.



Each element is computed as follows:

$$g(n,k) = g(n,k-1) + Y_k \, g(n-1,k)$$

where the Y_k multiplier is the service time of the job at device k.

At the end of the computation the quantity G(N,K) is found.* This is the normalizing constant for the product form equations where all devices have homogeneous service times. That is, the service time of the device is the same regardless of how many jobs are waiting in the queue. The rightmost column of the matrix contains the complete series of normalizing constants from G(1,K) through G(N,K). The performance measures of interest are functions of these normalizing constants and the device service times.

System Throughput

$$X(N) = \frac{G(N-1,K)}{G(N,K)}$$

Utilization of Device k

$$U_k(N) = Y_k \frac{G(N-1,K)}{G(N,K)}$$

Mean Queue Length at Device k

$$Q_k(N) = \sum_{n=1}^{N} Y_k^n \frac{G(N-n,K)}{G(N,K)}$$

Service Time of an Equivalent Load Dependent Server

$$S(N) = \frac{1}{X(N)}$$

An alternative way of calculating the mean queue length is given by the following recursive formula:

$$Q_k(N) = U_k(N) \ (1 + Q_k(N-1))$$

This method is particularly useful because one storage location per device is all that is needed to accumulate the mean queue length for an unlimited job population. The other expression implies storage for the complete column of n values of G(n,K).

*Note on Nomenclatures: In this paper g(n,k) denotes an intermediate value in the g matrix and G(N,K) is the final value corresponding to N jobs and K devices. Similarly h(m,k) and H(M,K) denotes intermediate and final values in the h matrix for interactive systems.

Batch Model With Homogeneous Service Times

The program for the batch model with homogeneous service times will handle up to six devices and any number of jobs specified by the user. Its short name is Batch HST-6.

The model is used where the service times for all devices are homogeneous.

The program is a straightforward implementation of Buzen's algorithm. Due to limited storage space the mean queue length is computed only for device #1. The following points cover inputs, outputs, and controls for the program:

Inputs to the model

- number of jobs — N
- device number (1 - 6) — k
- device service times — Y

Outputs in order of presentation are:

- number of jobs — N
- mean queue length at device 1 — Q
- normalizing constant — G(N,K)
- throughput with N jobs — X(N)
- mean job service time — S(N)
- up to six pairs of:
  - device number — k
  - utilization — U_k
- 99 indicating end of output

Input Controls - N, k, Y_k plus RUN

These three controls are located on function keys A,B C, respectively. Depressing the key interrupts program execution and displays the current value of the variable.

- insert a new value if required
- hit RUN to confirm your input action

Note: k is a dual purpose input.

- It indicates which device time, Y_k will be input next during input operations.

- It indicates the highest numbered device K to be modeled during execution.

Execution Controls - EXEC, RES, RUN

EXEC — Executes the program starting with an initialization of all required registers. The program will run until results are to be presented for a load of N jobs. EXEC is on function key E.

RUN — The program halts and displays its outputs in the preset order indicated above. Run is used for two purposes:
1. to obtain the next display in the cycle
2. at the end of the output cycle depressing RUN will continue the operation increasing the load to N + 1 without having to compute from scratch with a new EXECUTE.

RES — Resume is a special control which will continue the computation of g(n,k) without starting from scratch. It is intended to provide a shortcut around the logic which computes and displays the utilization statistics. It can be safely used at any point in the output cycle to advance to the next level of load.

Batch HST-6 has two main uses. The first, and most obvious, is to use it to model a batch processing system. Its second purpose is to model any subsystem of up to six devices, in order to obtain the schedule of service times for an equivalent load dependent single server. An example of its use in this role will be given in the description of the interactive model with a load dependent central server.

Recall again the sample problem in Figure 1. The CPU portion of the job is the largest component. The CPU will tend to be the limiting device so we assign it to device #1, to obtain the mean queue length.

Table I shows the results of running the program for job populations N = 1 through 5. Reading down each column the results appear in the order which the program produces them. In the output routine the device utilizations are output as a pair of numbers: first, the device number, then the utilization at that device; only the utilizations appear in Table I for each column.

Table I

Batch HST-6 Results for Sample Problem

Jobs N	1	2	3	4	5
Queue 1 (CPU)	.444	.997	1.65	2.40	3.23
G(N,K)	9.0	52.15	252	1111	4684
Throughput X(N)	.111	.172	.207	.226	.237
Service Time S(N)	9.0	5.79	4.83	4.41	4.21
Utilizations:					
5 - Channel	.133	.207	.248	.272	.285
4 - Disk 2	.100	.155	.186	.204	.214
3 - Disk 1	.233	.362	.435	.476	.498
2 - Swap	.089	.138	.166	.181	.190
1 - CPU	.444	.690	.828	.906	.949

The performance of the CPU is the main limiter in the system because the work is so CPU heavy. With five jobs active the CPU will be almost 95% busy and on the average there are 3.2 jobs at the CPU.

The appendix provides a listing of the Batch HST-6 Program for the SR-52.

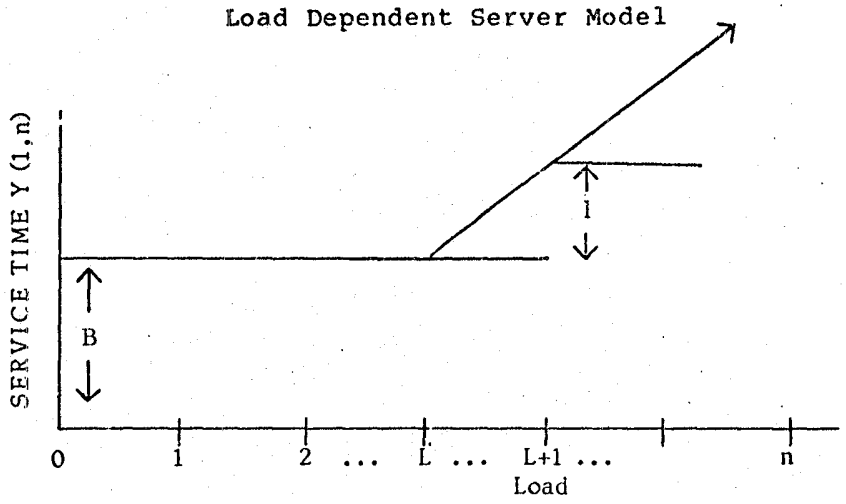
Batch Model With One Load Dependent Server

The second program is a minor variation on Batch HST-6 which allows one of the devices to be a load dependent server. When load dependent service is introduced at one of the devices the Buzen algorithm, slightly modified, can still be used to determine the throughput of the system. The device queue lengths, however, are no longer simple functions of the normalizing constants, G(N,K), and algorithms more complicated than can be easily handled on the SR-52, are required to compute these performance quantities.

In the modified program three registers are used to specify a simple model of the load dependent server. (In Batch HST-6 two of the registers were used for device #6 and one was used to accumulate the device #1 Queue length.) The net result is a program that can handle five devices. Device #1 is the load dependent server. The short name for this program is Batch LDS-5.

The load dependent server model is a simple function of the number of users in the device queue. Figure 2 illustrates the function. Base service time, B is a constant service time the job experiences up to the load at which the inflection point occurs in the function. Beyond the inflection point load, L, the service time per job increases by the increment amount, I for each additional user.

Figure 2



Stated another way:

For $n \leq L$ $\quad Y(1,n) = B$

For $n > L$ $\quad Y(1,n) = B + (n - L)I$

where:

Y(1,n) = service time at device #1 with n in queue
B = Base service time
L = Load at the inflection point
I = Increment per job in queue

The modification to Buzen's algorithm is simply to create the elements in column 1 by multiplying the previous row's value by the appropriate Y(1,n). For the device 1 column:

$$g(n,1) = Y(1,n)g(n-1,1)$$

The remaining rows and columns of the matrix are formed in the same way as previously described. At the end of the matrix computation G(N-1,K) and G(N,K) are available. These allow the following to be easily computed:

System Throughput	X(N)	$= \frac{G(N-1,K)}{G(N,K)}$
Service time of an equivalent single server	S(N)	$= \frac{1}{X(N)}$

For devices with homogeneous service times the utilizations can be computed from the relationship:

Utilization at device k $U_k = Y_k X(N)$

The following narrative covers the inputs, outputs and controls for the Batch LDS-5 program:

Inputs to the model

- number of jobs N
- device number (1 - 5) k
- device service times Y_k
- base service time B
- load at inflection point L
- increment per job I

Outputs in order of presentation are:

- number of jobs — N
- device 1 service time — Y(1,n)
- normalizing constant — G(N,K)
- throughput with N jobs — X(N)
- mean job service time — S(N)
- up to five pairs of:
  device number — k
  utilization — U_k
- 99 indicating end of output

Input Controls - N, k, Y_k plus RUN

These three controls are located on function keys A,B,C, respectively. Depressing the key interrupts program execution and displays the current value of the variable.

- insert a new value if required
- hit RUN to confirm your input action

Note: k is a dual purpose input.

- It indicates which device time, Y_k will be input next during input operations.

- It indicates the highest number of devices to be modeled during execution.

Input Controls for Load Dependent Server- B,L,I plus RUN

The parameters B,L and I are inserted as a group using function key D and the RUN key. Operation is as follows:

```
Depress function key D labeled B,L,I
     Current value of B is displayed
Insert new value if desired and depress RUN
     New value of B is displayed
Depress RUN
     Current value of L is displayed
Insert new value if desired and depress RUN
     New value of L is displayed
Depress RUN
     Current value of I is displayed
Insert new value if desired and depress RUN
     New value of I is displayed
```

Execution Controls - EXEC, RUN

EXEC — Executes the program starting with an initialization of all required registers. The program will run until results are to be presented for a load of N jobs. EXEC is on function key E.

RUN — The program halts and displays its outputs in the preset order indicated above. Run is used for two purposes:

1. to obtain the next display in the cycle
2. at the end of the output cycle depressing RUN will continue the operation increasing the load to N + 1 without having to start from scratch.

Batch LDS-5 has the same main uses as Batch HST-6, with the addition of a single load dependent server. It can be used to model a batch system or to model a subsystem of up to five devices in order to obtain an equivalent load dependent single server.

Once again let us use the sample problem of Figure 1. This time we will introduce load dependent service on the CPU to see

how it may affect the performance of the system. We will use the 4.0 sec of CPU time as the base service time. Beyond a load of two in the queue, the service time will be increased one second per job in the queue. That is to say:

B = 4.0 L = 2 I = 1.0

Table II shows the results of running the program for job populations 1 through 5. The format of the table is similar to Table 1; the queue at device 1 is not computed or presented. The device 1 service time with n in queue is presented in the second row.

Table II
Batch LDS-5 Results for Sample Problem

Jobs N	1	2	3	4	5
Y(1,n) service time	4.0	4.0	5.0	6.0	7.0
G(N,K)	9.0	52.15	268	1415	8398
Throughput X(N)	.111	.172	.195	.189	.168
Service Time S(N)	9.0	5.79	5.13	5.29	5.93
Utilizations:					
5 - Channel	.133	.207	.234	.227	.202
4 - Disk 2	.100	.155	.176	.170	.152
3 - Disk 1	.233	.362	.409	.397	.354
2 - Swap	.089	.138	.156	.151	.135
1 - CPU	.444	.690	.974*	1.135*	1.18*

Comparing Tables I and II, one can see the effects of the load dependent CPU. In the first two columns of the table the performance measures are, of course, the same; the CPU service

time is still the base value of 4.0 seconds. At the load of three jobs there is a slight loss in throughput compared with the first case study. Scanning across throughput now one finds the maximum occurs at three jobs on the system, beyond three jobs active the increased CPU time per job has a larger negative effect than the usual positive effect of adding more jobs to the multiprogramming set. This sort of thing can happen in overloaded systems, the per job device time may increase in heavily used systems; for example, as a result of increased competition for memory, more page fault interruptions may be required - resulting in more CPU and disk service time per job. The BLI function is used to represent increased system overhead past the threshold of thrashing.

One final note, the CPU utilizations marked with an asterisk in the table are the ones reported by the program. They are not correct because the CPU is a load dependent server. These utilizations were computed by multiplying maximum service time by throughput. The true value of the utilization of the load dependent server lies between this upper limit and a lower limit computed as the product of minimum service time and throughput. The algorithm for computing utilizations and queue lengths for the load dependent server would exceed the space available on the SR-52.

The appendix provides a listing of the Batch LDS program. Of particular note is the load dependent server model located at program steps 091 through 115. This section of the program can be changed to create other load dependent models. Registers 16, 17, and 18 contain the variables B, K, and I; these may assume different meanings or usage in a different load dependent server model. Any substitute function should start at the same location,

091, and place the appropriate value of service time in Register 01 prior to LABEL C, currently at location 117. Note that a completely arbitrary load dependent service time schedule can be entered dynamically by replacing the current LDS routine with a halt and display of the current value of Register 01. If the user changes the register to a new value it will be the next one used. The required routine would be:

```
RCL 01
HLT
STO 01
```

Using this routine it is possible to model any number of devices by breaking the system into device groupings of 4 to 6 devices. For example a 10 device system could be modeled as a six device subsystem plus four individual devices. First Batch HST-6 is run to obtain a schedule of load dependent service times. Then Batch LDS-5 is run using the subsystem service times for device 1 and the remaining four devices as two thru 5. This is an exact method of combining multiple devices.

One user of this program has noted that the BLI function is an approximation to what happens during thrashing. He reports two additional subsystem approximations that he has found useful. For a P processor multiprocessor, $Y(1,n) = S_1 * \min(n,P)^B$ is a reasonable form for an approximation. B = 1 represents an ideal multiprocessor. With B < 1 various amounts of multiprocessor mutual interference can be modeled. For an I/O subsystem a power curve fit of the form $Y(1,n) = A * \min(n,V)^{-B}$ provides a good approximation. In this case V is an arbitrary maximum value which is specified by the user. TI program STI-09 from the statistics library is handy for determining A and B.

THE INTERACTIVE MODELS

The tutorial by Denning and Buzen also presented a very compact algorithm for handling interactive systems. Again we will use a sample problem approach. Figure 3 illustrates such a system. It has M terminals connected to a central system. Each of the terminal users has an average think time Z. The central system has the same five devices and associated service times as the previous batch cases had. (This will facilitate using the batch results in solving the interactive systems.) For the case study we will use a think time of 10 seconds and terminal populations M = 2, 4, 6, 8, 10.

The M terminals represent M jobs in the system as a whole. Actually some number N are on the central system at any given time and M-N jobs are out at the terminals. The terminals are treated as a single subsystem whose service time is Z/n when there are n users thinking (i.e. jobs at the terminals). The terminal subsystem is thus a load dependent server. The devices in the central system all have homogeneous service times.

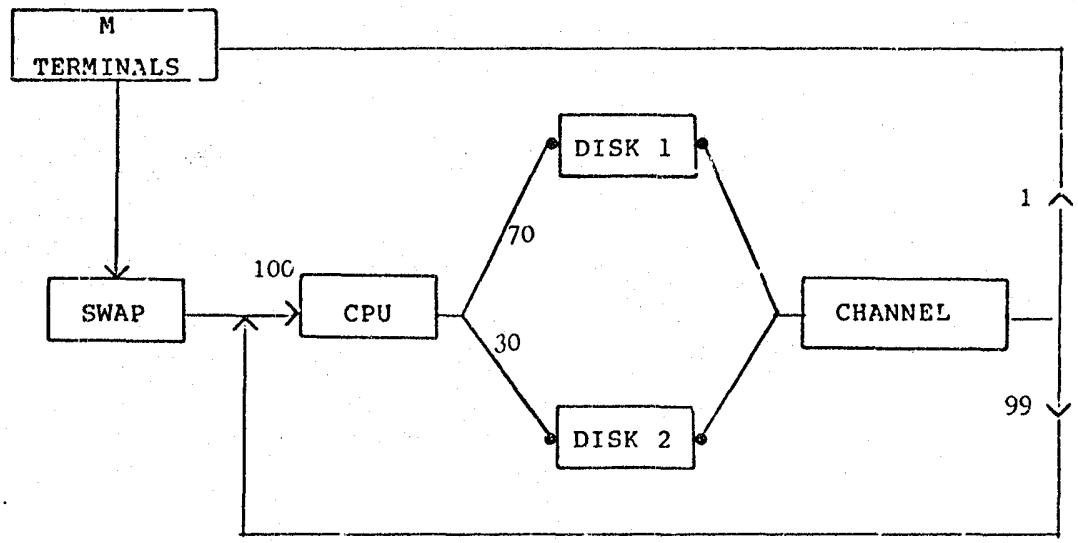
The algorithm, attributed to Williams and Bhandiwad [31, is quite similar to the Buzen algorithm described previously. The interactive algorithm fills in a two dimensional matrix h; the columns correspond to k devices; and the rows correspond to the m terminals. Elements of the matrix are computed from the adjacent elements, above and to the left as shown in the figure below. Initially row 0 and column 0 contain 1's.

Each element of the matrix is computed as follows:

$$h(m,k) = h(m,k-1) + mY_k/Z\ h(m-1,k)$$

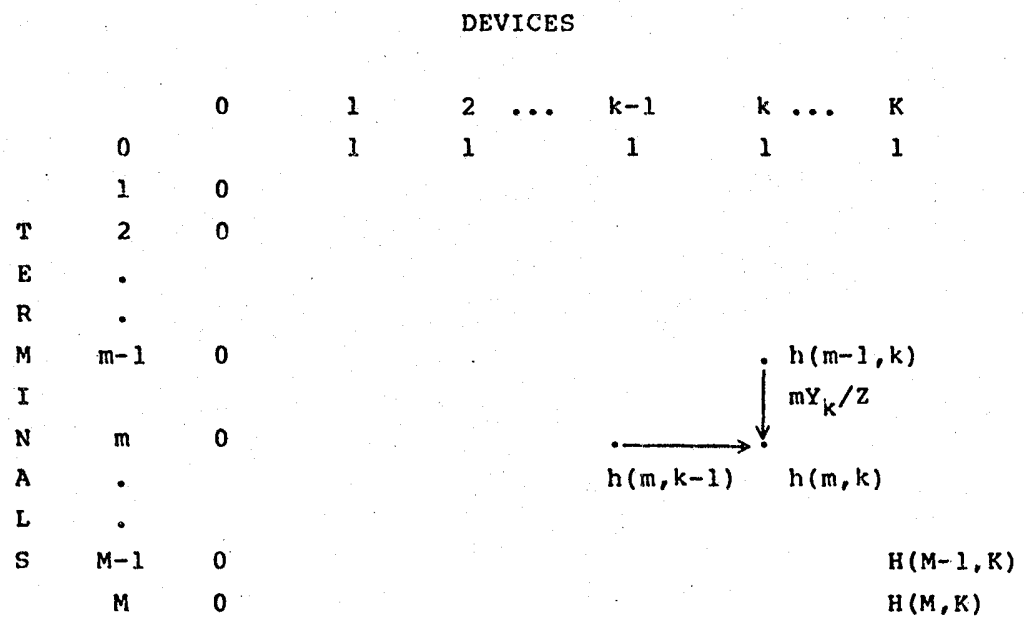
where Y_k is the service time of the job at device k ($Y_k = V_k S_k$).

Figure 3
Sample Problem - An Interactive System



JOB CHARACTERISTICS

DEVICE NAME	DEVICE NO. k	NO. OF VISITS V_k	TIME PER VISIT S_k	TOTAL SERVICE Y_k SEC.
Swap	2	1	.8	.8
CPU	1	100	.040	4.0
Disk 1	3	70	.030	2.1
Disk 2	4	30	.030	.9
Channel	5	100	.012	1.2



At the end of the computation the values H(M-1,K) and H(M,K) are available. These allow the following performance measures to be computed.

Central system idle probability $$P(0) = \frac{1}{H(M,K)}$$

Throughput $$X(M) = \frac{M}{Z} \cdot \frac{H(M-1,K)}{H(M,K)}$$

Response time $$R(M) = \frac{M}{X(M)} - Z$$

Mean active load $$Q = M - Z\ X(M)$$

Because the devices of the central subsystem are homogeneous the utilization of each device is simply the product of Y_k, the service time. and X(M), the throughput of the system.

Interactive Model With Homogeneous Service Time

The program for the interactive model with homogeneous service times will handle up to six devices and any number of terminals specified by the user. The short name of this program is Interactive HST-6.

The program is a straightforward implementation of the interactive algorithm and provides all of the performance parameters indicated above. The following cover inputs, outputs, and controls for the program.

Inputs to the model

- number of terminals M
- device number (1 - 6) k
- device service times Y_k
- think time Z

Outputs in order of presentation are:

- number of terminals M
- normalizing constant H(M,K)
- system idle P(0)
- throughput with M terminals X(M)
- response time R(M)
- mean jobs in system Q
- up to six pairs of:
  - device number k
  - utilization U_k
- 99 indicating end of output

Input Controls - M, k, Y_k plus RUN

These three controls are located on function keys A,B,C, respectively. Depressing the key interrupts program execution and displays the current value of the variable.

- insert a new value if required
- hit RUN to confirm your input action

Note: k is a dual purpose input.

- It indicates which device time Y_k will be input next during input operations.

- It indicates the highest number of devices to be modeled during execution.

Execution Controls - EXEC, RUN

EXEC — Executes the program starting with an initialization of all required registers. The program will run until results are to be presented for a load of M terminals. EXEC is on function key E.

RUN — The program halts and displays its outputs in the preset order indicated above. Run is used for two purposes:

1. to obtain the next display in the cycle
2. at the end of the output cycle depressing RUN will continue the operation increasing the load to M + 1 without having to start from scratch.

The sample problem in Figure 3 is a simple variation on the original batch problem of Figure 1 - the job characteristics on the central system are the same in both cases. The difference is in the way jobs are introduced to the system; the original case had N jobs always present, in this case study M terminals introduce the jobs to the system after a think time of 10 seconds. Table III shows the results of running the program with loads of 2, 4, 6, 8, and 10 terminals.

Table III
Interactive HST-6 Results

M Terminals	2	4	6	8	10
H(m,k)	3.843	19.56	139.8	1454	22009
P(0) - system idle	.260	.051	.007	.0006	.00004
X(M) throughput	.099	.170	.214	.236	.246
R(M) response time	10.2	13.46	17.9	23.8	30.7
Q avg jobs in system	1.01	2.29	3.85	5.63	7.54
Utilizations:					
5 - Channel	.119	.20	.26	.28	.29
4 - Disk 2	.089	.15	.19	.21	.22
3 - Disk 1	.207	.36	.45	.50	.52
2 - Swap	.079	.14	.17	.19	.19
1 - CPU	.395	.68	.86	.95	.98

The minimum response time occurs when only one terminal is active (not shown in table); the response time for one user is simply the sum of the service times on each of the devices, or 9 seconds. As the terminal load increases, the throughput of the

system rises rapidly at first and slower later on as the system approaches its saturation limit. In both the batch and the interactive cases this limit is established by the CPU component of the workload. At 4 CPU seconds per job, the throughput limit will be 1/4 = .25 jobs per second. At a load of 6 terminals the throughput is roughly 86% of this limit. Response time is roughly twice what it would be on a dedicated system. Adding more terminals will make response time worse with little gain in throughput.

A comparison of the interactive and batch cases raises an interesting question:

At a terminal load of 4 users there is an average of 2.29 terminal users in the central system and the throughput is .170 jobs per second. This is less than the .172 jobs per second throughput of the batch system with two jobs active. One might have expected that with more jobs active (2.29 is greater than 2), that the system throughput would be greater, not less. I don't know why this is so.

Interactive Model With Load Dependent Central Server

In the tutorial Denning and Buzen point out that the central system portion of an interactive system can be modeled as a single server with a load dependent service time. The article does not describe the algorithm for computing the performance quantities, but it is a simple variation on the interactive algorithm presented in the previous section. For the load dependent central server, the matrix h can be viewed as a simple one column matrix; the single column represents the single load dependent server. The service time of the central system under a given constant load of n jobs, S(n), is equal to the reciprocal of the throughput for a system with n jobs and the terminal visit shorted out.

$$S(n) = \frac{1}{X(n)}$$

One can think of such a system as a batch system with n jcbs and use the Batch HST-6 or Batch LDS-5 programs, as appropriate, to calculate the schedule of load dependent service times.

For a system with M terminals successive elements h(m) of the single column matrix h are computed by the following recursive formula:

$$h(m) = 1 + \frac{m\ S(M-m+1)\ h(m-1)}{Z}$$

where:

h(0) = 1

m is stepped from 1 up to M

Z is the think time

S(n) = service time with n jobs active.

Note: The recursion takes the service time schedule in the reverse order to increasing m. That is, S(M) is the first service time in the recursion, S(1) is the last.

At the end of the recursion H(M) is found. A second pass of the recursion is made with a terminal load of (M-1) terminals to find H(M-1). This value H(M-1) is not the same as the value of h(m-1) found on the previous pass of the recursion. H(M-1) considers the service times S(M-1) thru S(1) in its recursion. The h(m-1) of the previous recursion used S(M) thru S(2). The two values H(M) and H(M-1) are used to find the following performance quantities:

System idle	$P(0) = \frac{1}{H(M)}$
Throughput	$X(M) = \frac{M}{Z} \cdot \frac{H(M-1)}{H(M)}$
Response time	$R(M) = \frac{M}{X(M)} - Z$
Mean queue length	$Q = M - Z\ X(M)$

Also, starting from the value for P(o) found above, one can find the probability, P(n), of there being n jobs in the system from the following recursion:

$$P(n) = \frac{(M-n+1)\ S(n)}{Z}\ P(n-1)$$

The program for computing the performance quantities for an interactive system with a load dependent central server is called Interactive LDCS-1. The program implements the algorithm and computes the principal performance quantities described above. Due to lack of sufficient program storage on the SR-52 the recursion for calculating p(n) has not been included in Interactive LDCS-1.

Also due to register storage limitations no more than eleven values of S(n) can be handled. This limits the effective degree of multiprogramming of the system to serving a maximum of eleven terminals simultaneously on the central system.

Actually the limit of eleven degrees of multiprogramming is not a serious one because in most real systems the throughput changes attributable to operating above the degree of multiprogramming of eleven are usually so slight and improbable of occurrence that they can be neglected.

The Interactive LDCS-1 program offers an additional parameter setting called the multiprogramming limit, N. In some real systems the size of main memory or possible operating system parameters may limit the number of concurrent jobs that the system will consider ready for execution. When the level of multiprogramming is set to N the program will consider the service time of the system to be a constant S(N) for loads greater than or equal to N.

The rationale for modeling a fixed level of multiprogramming in this manner is treated in the article by Chandy and Sauer [4] on approximate methods. This is an approximation by use of flow equivalent methods for passive elements of the system. The passive element in this case is memory which restricts the multiprogramming to some level n which is less than the total terminal population m. The system has been collapsed from a multiple device system to an equivalent load dependent single server with a schedule of service times S(N). For example, by only considering rates S(1), S(2), S(3) and then using S(3) instead of S(4), S(5),

.... S(m), one is effectively limiting multiprogramming to level 3. That is to say the "improved" service times due to multiprogramming at levels higher than three are denied by setting them to S(3).

The following points cover inputs, outputs, and controls for the program.

Inputs to the model

• think time	Z
• number of terminals	M
• multiprogramming limit	N
• load index	n
• load dependent service time	S(n)

Outputs in order of presentation are

• number of terminals	M
• the matrix constant for M	H(M)
• system idle	P(0)
• the matrix constant for M-1	H(M-1)
• system throughput	X(M)
• response time	R(M)
• mean number in system	Q
• 99 indicating end of output cycle	

Input Controls: Z, M, N and RUN

These three controls are located on function keys A, B, and C respectively. Depressing the key interrupts the program and causes the current value of the variable to be displayed.

- insert a new value if required
- depress RUN to confirm your input action

Input Contols n, S(n) and RUN

Function key D is labelled n, S(n) and is used with RUN to enter the schedule of load dependent service times S(N).

- depress function key D
  value n is auto incremented and displayed
- enter different value of n if desired
- depress RUN
  service time, S(n), is displayed
- enter different service time if desired
- depress run to confirm
- repeat until all values of S(n) are entered

Execution Controls - EXEC, RUN

EXEC — Executes the program starting with an initialization of all required registers. The program will run until results are to be presented for a load of M terminals. EXEC is on function key E.

RUN — The program halts and displays its outputs in the preset order indicated above. RUN is used for two purposes:
1. to obtain the next display in the cycle
2. at the end of the output cycle depressing RUN will continue the operation increasing the load to M + 1 without having to start from scratch.

Note: Computation of the performance parameters, except for the case where terminal load M=1, requires two consecutive passes through the recursive formula. On the first pass the terminal load should be set at M-1 and run through the complete output cycle. [On this "primer pass" only H(M-1) and P(0) are guaranteed to be correct. H(M-1) is saved for the next pass.] After the "99" display at end of the cycle depress RUN. This will increment the terminal load from M-1 to M and cause the second pass through the recursive formula. The output displays will be correct for load M.

Depressing RUN at the end of any cycle executes the next pass and provides results for the next higher terminal load: i.e., M+1, M+2, ... etc.

Once again we turn to the sample problem in Figure 3. We are interested in studying the performance of the interactive system over a range of terminal loads from 2 through 10. We, therefore, will need the schedule of load dependent service times for the corresponding batch system with the number of jobs equal to 1 through 10. While we are at it, we might as well get the schedule of S(n) for the batch system variation in which the CPU had a load dependent service time. (Recall that this will lead to reduced throughput at higher multiprogramming levels.) Table IV shows the load dependent central server schedules, S(n), for the two batch systems.

Table IV

Load Dependent Central Server Schedules

	Constant CPU		Load Dependent CPU	
N Jobs	Y:CPU	System S(n)	Y(1,n) CPU	System S(n)
1	4 sec	9.0 sec	4 sec	9.0 sec
2	4	5.79	4	5.79
3	4	4.83	5	5.13
4	4	4.41	6	5.28
5	4	4.21	7	5.93
6	4	4.11	8	6.91
7	4	4.05	9	8.04
8	4	4.03	10	9.19
9	4	4.02	11	10.32
10	4	4.01	12	11.42

In addition to being a schedule of S(n) inputs for the model, Table IV is interesting in its own right. The columns headed by Constant CPU show the CPU time and S(n) from using Batch HST-6. Similar columns under Load Dependent CPU were calculated using Batch LDS-5. In both cases the CPU is the heaviest component of the work load and the System Service time S(n) approaches it asymptotically. What's interesting in the load dependent case is that S(n) dips below the CPU service time and then approaches it from that vantage point. It is also evident from the table that increasing the multiprogramming level in the constant CPU case beyond about four jobs will have very little payoff. For the load dependent CPU going beyond a level of three jobs is expected to hurt performance.

Table V

Interactive LDCS-1 Results

Terminals M	2	4	6	8	10
H(M)	3.84	19.55	139.7	1450	21913
System Idle P_0	.260	.051	.007	.0007	.00004
H(M-1)	1.9	8.34	50.0	429	5383
Throughput X(M)	.099	.170	.214	.237	.246
Response Time R(M)	10.22	13.45	18.0	23.8	30.7
Number in System Q	1.01	2.29	3.85	5.63	7.54

Table V shows the results of running the program for the constant CPU case. Except for minor roundoff differences, due to inserting S(N) to only three places, the results are the same as previously indicated in Table III. The results agree "exactly" when all quantities are entered to the maximum precision allowed by the calculator.

A much more interesting set of results is found by examining the effects of multiprogramming level on the performance of the system with the load dependent CPU. Multiprogramming levels, N, of 1,2,3,4, were examined for terminal loads, M, of 1 through 10. Table VI records the resulting throughputs and response times.

The best performance occurs at multiprogramming level 3, the third column of the table. This was expected because S(N) was a minimum at a load of 3. The rightmost column of the table shows how poorly the system will perform if no limits are placed on multiprogramming [i.e. the multiprogramming level N is made equal to the number of terminals M]. Here the best throughput is achieved at 5 terminals active because it is not until this point that the average number of jobs in the system get up to around 3.

Table VI

Throughput and Response Time With Load Dependent CPU

		Multiprogramming Level-N				
No of Terminals		1	2	3	4	M
1	Throughput	.053	.053	.053	.053	.053
	Response	9.0	9.0	9.0	9.0	9.0
2	X(M)	.086	.099	.099	.099	.099
	R(M)	13.3	10.2	10.2	10.2	10.2
3	X(M)	.102	.133	.137	.137	.137
	R(M)	19.3	12.5	11.9	11.9	11.9
4	X(M)	.109	.155	.164	.163	.163
	R(M)	26.7	15.8	14.4	14.5	14.5
5	X(M)	.110	.166	.181	.178	.175
	R(M)	35.2	20.1	17.6	18.0	18.5
6	X(M)	.111	.170	.189	.186	.171
	R(M)	44.0	25.2	21.7	22.3	25.1
7	X(M)	.111	.172	.193	.188	.153
	R(M)	53.0	30.7	26.2	27.2	35.4
8	X(M)	.111	.172	.194	.189	.132
	R(M)	62.0	36.4	31.1	32.3	50.4
9	X(M)	.111	.173	.195	.189	.113
	R(M)	71.0	42.1	36.2	37.5	69.6
10	X(M)	.111	.173	.195	.189	.098
	R(M)	80.0	47.9	41.3	42.8	91.6

The results from this case study are graphically presented in Figure 4 as a family of performance plots. Additional multiprogramming levels not shown in the table have been added to show how the throughput varies for the range 4 through 10. Also, to allow comparison with a system which does not have the load dependent CPU, three additional plots are shown as dashed lines in the figure. These three throughput curves were generated using the S(N) schedule from Table IV labeled constant CPU.

There is a lot of information conveyed by the figure. A few points will be made to illustrate what can be learned. A system without a load dependent CPU can be viewed as an "ideal system" because it does not require more system overhead per job to manage 10 jobs than to manage 2 jobs. The first four points discuss performance of this ideal system.

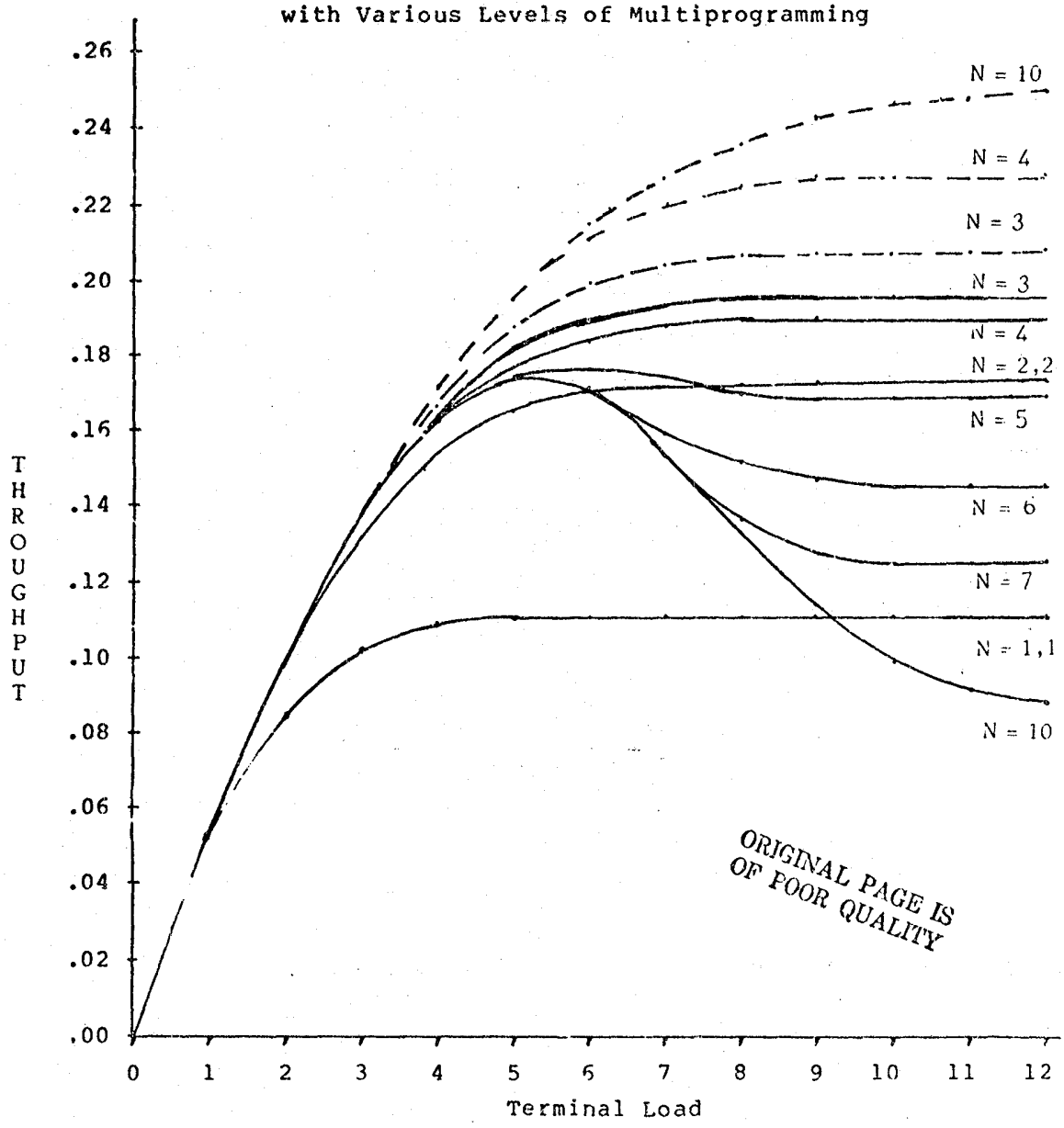
1. The uppermost dashed curve labeled N=10 shows the "best" possible throughput for the ideal system. There is no "extra overhead work" which was modeled as a load dependent CPU. There is no practical limit on multiprogramming with the limit set at 10. The system saturates at a throughput of .25 when the limiting device, the CPU, reaches 100% busy.

2. Dashed curves labeled N=4 and N=3 indicate how throughput would drop due to limiting the level of multiprogramming. Main memory size is often such a limiter of multiprogramming.

Figure 4

Throughput vs Terminal Load

with Various Levels of Multiprogramming



Key: ---- Ideal System (Constant CPU)

_____ Realistic System (Load Dependent CPU)

N Multiprogramming Level

3. Solid curves N=1,1 and N=2,2 show the throughput for levels 1 and 2 for both the ideal system and the system with a load dependent CPU. Recall that the load dependent function didn't start increasing the CPU load until 3 jobs were in the system.

4. The large difference between N=1 and N=10 dashed shows the expected gains due to multiprogramming for the "ideal" system. Forty-four percent of the potential gain is achieved by going from level 1 to 2. [Seventy percent by going from 1 to 3, 84% for going from 1 to 4.]

In most real systems there is some amount of extra overhead involved with operating at higher levels of multiprogramming. Increaced paging activity or increased swapping is such a form of load dependent behavior which could result in higher CPU activity for storage managment and page/swap support. The solid lines in this figure show a hypothetical system which is exhibiting realistic system behavior. The distinguishing character of the realistic throughput curve is that things get better up to a point where saturation occurs and then, if the load is increased, the throughput will actually get worse. Four points are made about the "hypothetical realistic" system.

1. The solid plot for N=3 shows the best throughput for the system. The service time, S(N), with three jobs in the system is at its lowest so throughput will be best if multiprogramming is at level 3.

2. The difference between the solid plot N=3 and the dashed plot N=3 indicates the difference between an ideal system and its "realistic" counterpart.

3. Increasing the multiprogramming level from 3 to 4 hurts performance a little. The difference between the ideal and realistic systems has increased.

4. Increasing the multiprogramming level to five or beyond actually results in lowering the throughput. In all of these cases the throughput approaches a limit which is 1 - S(N).

This is an example of paradoxical behavior which occurs from time to time. Conventional wisdom says increasing the multiprogramming level is good. Conventional wisdom also says that the benefits of increased multiprogramming are progressively diminishing. Conventional wisdom does not predict that throughput will drop with increased multiprogramming, as this case seems to indicate. Paradoxically conventional wisdom is correct, if we are trying to distinguish causal realtionships. The root cause of the poor performance is the increased overhead for storage management, modeled in this case by a load dependent CPU. Incrasing the multiprogramming level merely allows the storage management problem to manifest itself.

SUMMARY

Starting from the tutorial by Denning and Buzen [and that is an excellent place for anyone to start] the algorithms for handling closed networks with a single job class were adapted for use on the SR-52 programmable calculator. Along the way it was found that by slightly altering the Buzen algorithm to process the G and H matrices row by row instead of column by column, that six devices and an unlimited job/terminal population could be handled on the SR-52. Techniques were also introduced for handling a simple load dependent server and for studying interactive systems with fixed multiprogramming limits.

The paper provides listings of the four programs and a sample case study which can be replicated on the SR-52.

Next on the agenda is conversion to the TI-59, additional load dependent servers, and some simple aids for approximating systems with parallel tasks.

APPENDIX

PROGRAMS FOR THE SR-52

TITLE Batch HST-6 PAGE 1 OF 2

PROGRAMMER E. S. Herndon DATE March 1980

SR-52 Coding Form

LOC	CODE	KEY	COMMENTS	LOC	CODE	KEY	COMMENTS	LOC	CODE	KEY	COMMENTS	LABELS
000 112		LBL				STO				0		A N-Jobs
		E'				1	Q_1=0			8	k is	B k-devices
		STO		040 152		7				STO	used for	C Y_k-service
		1				7				0	LOOP	D RESUME
		9				STO		080 192		0	COUNT	E EXECUTE
005 117		IND	General			0	DSZ=7			8		A'
		RCL	Input			0				STO	G Array	B'
		1	Routine	045 157		8				1	POINTER	C'
		9	-			STO				9		D' ROW
		HLT	Display			1	SET	085 197		0	0=	E GEN INPUT
010 122		IND	Old Value			9	all			STO	Y Array	REGISTERS
		STO				1	7			1	Pointer	00 LOOP
		1	SAVE	050 162		LBL	G(0,k)			8	0=G(n,o)	01 Y_1
		9	NEW			SIN	elements			LBL		02 Y_2
		rtn	VALUE			SUM	= 1	090 202		TAN		03 Y_3
015 127		LBL				1				+	+	04 Y_4
		A	INPUT			9				1		05 Y_5
		7	N	055 167		IND				SUM		06 Y_6
		E'	-jobs-			STO				1		07 N
		HLT				1	End of	095 207		9		08 k
020 132		LBL				9	Initial			IND	G(n-1,k)	09 G(n,1)
		B	INPUT			DSZ	Phase			RCL		10 G(n,2)
		8	k	060 172		SIN				1		11 G(n,3)
		E'	device no.			LBL	Continue			9		12 G(n,4)
		HLT				D	Row	100 212		X	X	13 G(n,5)
025 137		LBL				=	Process			1		14 G(n,6)
		C				1				SUM		15 G(n-1,k)
		RCL	INPUT	065 177		SUM	n=n+1			1		16 n-Count
		0	Y_k			1				8		17 Q_1
		8				6		105 217		IND	Y_k	18 IND Y_k
030 142		E'				IND				RCL		19 IND, I, G_k
		HLT				RCL	Save			1		FLAGS
		LBL	Execute	070 182		1	G(N,K)			8		0
		E				9	in			=	=	1
		0				STO	G(N-1,K)	110 222		IND		2
035 147		STO	n=0			1	for X(N)			STO	G(n,k)	3
		1				5	CALC.	TEXAS INSTRUMENTS INCORPORATED				4
		6		075 187		RCL						

TITLE Batch HST-6 PAGE 2 OF 2

PROGRAMMER E. S. Herndon DATE March 1980

SR-52 Coding Form

LOC	CODE	KEY	COMMENTS	LOC	CODE	KEY	COMMENTS	LOC	CODE	KEY	COMMENTS	LABELS
000 112		1				RCL	Outputs			0		A N-Jobs
		9				0				)		B k-devices
		DSZ	END	040 152		8	k for			÷		C Y_k-service
		TAN	ROW?			STO	LOOP			HLT	"U_k"	D RESUME
		RCL	Update			0	CONTROL	080 192		IND		E EXECUTE
005 117		1	Q			0				RCL		A'
		7	(Q_{I+1})			RCL	DISPLAY			0		B'
		+		045 157		1	"n"			0		C'
		1				6				DSZ		D' Row
		)				HLT		085 197		COS		E' INPUT
010 122		X	X			RCL				9		REGISTERS
		RCL				1				9	"99"	00 LOOP
		0	Y_1	050 162		7				HLT	the end	01 Y_1
		1				HLT	"Q_1"			D	next row	02 Y_2
		X	X			RCL		090 202				03 Y_3
015 127		RCL				1	G(n-1,k)					04 Y_4
		1	G(n-1,k)			5						05 Y_5
		5		055 167		÷						06 Y_6
		÷	÷			IND						07 N
		IND				RCL		095 207				08 k
020 132		RCL	G(n,k)			1						09 G(n,1)
		1				9						10 G(n,2)
		9		060 172		HLT	"G(N,K)"					11 G(n,3)
		=	=			=						12 G(n,4)
		STO	Q_1(n)			HLT	"X(N)"	100 212				13 G(n,5)
025 137		1				1/x						14 G(n,6)
		7				HLT	"S(N)"					15 G(n-1,k)
		RCL		065 177		1/x						16 n
		1	n			LBL	Device					17 Q_1
		6				COS	OUTPUTS	105 217				18 IND Y_k
030 142		-	-			X	U_k					19 IND I,G_k
		RCL				RCL						FLAGS
		0	N	070 182		0						0
		,				0						1
		=				HLT	"k"	110 222				2
035 147		INV				IND						3
		IF POS	DONE?			RCL		TEXAS INSTRUMENTS INCORPORATED				4
		D		075 187		0						

TITLE 5 Devices – Load Dependent PAGE 1 OF 2

PROGRAMMER E. S. Herndon DATE March 1980

SR-52 Coding Form

LOC	CODE	KEY	COMMENTS	LOC	CODE	KEY	COMMENTS	LOC	CODE	KEY	COMMENTS	LABELS
000 112		LBL				1				1	N=N+1	A N Jobs
		E'				7	Load			5		B d-Devices
		STO		040 152		E'	At INFL			IND		C Y_d-Service
		1				HLT				RCL		D B,L,I
		9	General			1	INCR	080 192		1	Save	E EXECUTE
005 117		IND	Input			8				9	G(N,K)	A'
		RCL	Routine			E'				STO	in	B'
		1		045 157		HLT				1	G(N-1,K)	C'
		9				LBL	EXECUTE			4	for X(N)	D'
		HLT				E	INITIAL	085 197		RCL		E' INDR-I
010 122		IND				0				0		REGISTERS
		STO				STO				8		00 LOOP
		1		050 162		1	n=0			STO	LOOP	01 Y_1
		9				5				0	=d	02 Y_2
		RTN				6		090 202		0		03 Y_3
015 127		LBL				STO				RCL	SET	04 Y_4
		A				0				1	Y_1 TO	05 Y_5
		7	N-JOBS	055 167		0	SET			6	LOAD	06 IND Y_k
		E'				8	ALL			STO	DEPENDEN	07 N
		HLT				STO	G(n,k)	095 207		0	FUNCTION	08 k
020 132		LBL				1	ELEMENTS			1	B	09 G(n,1)
		B				9	TO			RCL	+	10 G(n,2)
		8	K	060 172		LBL				1	(n	11 G(n,3)
		E'				SIN	1			5		12 G(n,4)
		HLT				1		100 212		–	–	13 G(n,5)
025 137		LBL				SUM				RCL		14 G(n-1,k)
		C				1				1	L)	15 n-counter
		RCL		065 177		9				7		16 B-
		0				IND				=		17 L-
		8				STO		105 217		INV		18 I-
030 142		E'	Y_k			1				IF POS		19 INDR I,C
		HLT				9				C'		FLAGS
		LBL		070 182		DSZ	END			X	X	0
		D				SIN	INITIAL			RCL		1
		1	BASE			LBL	Continue	110 222		1	I	2
035 147		6				D'	ROW			8		3
		E'				1		TEXAS INSTRUMENTS INCORPORATED				4
		HLT		075 187		SUM						

TITLE 5 Devices-Load Dependent PAGE 2 OF 2

PROGRAMMER E. S. Herndon DATE March 1980

SR-52 Coding Form

LOC	CODE	KEY	COMMENTS	LOC	CODE	KEY	COMMENTS	LOC	CODE	KEY	COMMENTS	LABELS
000 112		=	Add to			1	G(N,K)			HLT	"G(N,K)"	A
		SUM	Base			9				=		B
		0	Y_1 -	040 152		DSZ				HLT	"X(N)"	C
		1				TAN				1/x		D
		LBL				RCL	TESDONE	080 192		HLT	"S(N)"	E
005 117		C'				1	n			1/x		A'
		8				5				LBL		B'
		STO		045 157		-	-			COS	DEVICES	C'
		1	G Array			RCL				X		D'
		9	Pointer			0	N	085 197		(		E'
010 122		0				7				RCL		REGISTERS
		STO				=				0		00
		0	Y Array	050 162		INV				0		01
		6	Pointer			IFPOS	CONT			HLT	"K"	02
		0	G(n-1,0)			D'	ROWS	090 202		IND		03
015 127		LBL				RCL				RCL		04
		TAN				1				0		05
		+	+	055 167		5				0	Y_k	06
		1				HLT	"N"			)		07
		SUM				RCL		095 207		÷		08
020 132		1				0				HLT	"U_k"	09
		9				1				IND		10
		IND		060 172		HLT	"Y_1"			RCL		11
		RCL	G(N-1,k)			RCL				0		12
		1				0		100 212		0		13
025 137		9				8	k for			DSZ		14
		X	X			STO	dsz			COS		15
		1		065 177		0	CONTROL			=		16
		SUM				0				9	END	17
		0				RCL		105 217		9		18
030 142		6	Y_k			1	G(n-1,k)			HLT		19
		IND				4				D'	NEXT	FLAGS
		RCL		070 182		÷					ROW	0
		0										1
		6				IND		110 222				
035 147		=				RCL						
		IND				1		TEXAS INSTRUMENTS INCORPORATED				4
		STO		075 187		9						

TITLE Interactive HST-6 PAGE 1 OF 2

PROGRAMMER E. S. Herndon DATE March 1980

SR-52 Coding Form

LOC	CODE	KEY	COMMENTS	LOC	CODE	KEY	COMMENTS	LOC	CODE	KEY	COMMENTS	LABELS
000 112		LBL				LBL	Initial			5		A M-Jobs
		E'				E				RCL		B k-device
		STO		040 152		0				0	K	C Y_k-service
		1				STO	m=0			8	is	D Z-think
		9	GENERAL			1		080 192		STO	used	E EXECUTE
005 117		IND	INPUT			6				0	for	A'
		RCL	ROUTINE			7				0	loop	B'
		1		045 157		STO				0	control	C'
		9				0	set all 7			STO		D' Row
		HLT				0	h array	085 197		1	set	E' IND I
010 122		IND				9	elements			8	Y_1	REGISTERS
		STO				STO	to 1			8	and	00 loop
		1		050 162		1				STO	h(m,1)	01 Y_1
		9				9				1	POINTERS	02 Y_2
		RTN				LBL		090 202		9		03 Y_3
015 127		LBL	INPUT			SIN				1	k(m-1,0)	04 Y_4
		A				1				LBL		05 Y_5
		7	M	055 167		IND				TAN		06 Y_6
		E'	TERMINALS			STO				+	+	07 M
		HLT				1		095 207		1		08 k
020 132		LBL				9				SUM		09 h(m,1)
		B				SUM				1	h(m-1,k)	10 h(m,2)
		8	K-devices	060 172		1				9		11 h(m,3)
		E'				9	END			IND		12 h(m,4)
		HLT				DSZ	INITIAL	100 212		RCL		13 h(m,5)
025 137		LBL				SIN				1		14 h(m,6)
		C				LBL	ROW			9		15 h(m-1,k)
		RCL		065 177		D'	PROCESS			X	X	16 m counter
		0				1				1		17 Z
		8	Y_k			SUM	m=m+1	105 217		SUM		18 IND Y_k
030 142		E'	Service			1				1		19 IND I,h
		HLT	Time			6				8	Y_k	FLAGS
		LBL		070 182		IND	h(m-1,k)			IND		0
		D				RCL	Saved			RCL		1
		1	Z			1	for X(m)	110 222		1		2
035 147		7	think			9	calc			8		3
		E'	time			STO	in R15	TEXAS INSTRUMENTS INCORPORATED				4
		HLT		075 187		1						

TITLE Interactive HST-6 PAGE 2 OF 2

PROGRAMMER E. S. Herndon DATE March 1980

SR-52 Coding Form

LOC	CODE	KEY	COMMENTS	LOC	CODE	KEY	COMMENTS	LOC	CODE	KEY	COMMENTS	LABELS
000 112		X	X			÷	÷			0	k	A M-Jobs
		RCL				RCL				8	is loop	B k-device
		1	m	040 152		1	Z			STO	control	C Y_k Service
		6				7				0	for	D Z-Think
		÷	÷			X	X	080 192		0	U_k	E EXECUTE
005 117		RCL				RCL				RCL		A'
		1	Z			1	h(m-1,d)			0	X(M)	B'
		7		045 157		5				7		C'
		=	=			=				LBL	U_k	D' Row
		IND				HLT	"X(M)"	085 197		COS	Loop	E' IND I
010 122		STO	h(m,k)			STO				X		REGISTERS
		1				0				(		00 loop
		9		050 162		7				RCL		01 Y_1
		DSZ	test end			1/x	1/x(M)			0		02 Y_2
		TAN	row			X	X	090 202		0		03 Y_3
015 127		RCL	test			RCL				HLT	"k"	04 Y_4
		1	done			1	m			IND		05 Y_5
		6		055 167		6				RCL		06 Y_6
		-	m-M			-	-			0		07 M, X(M)
		RCL				RCL		095 207		0		08 k
020 132		0				1	Z			)		09 h(m,1)
		7				7				÷		10 h(m,2)
		=	until	060 172		=				HLT	"U_k"	11 h(m,3)
		INV	m ≥ M			HLT	"R(M)"			IND		12 h(m,4)
		IF POS	continue			RCL		100 212		RCL		13 h(m,5)
025 137		D'	rows			1				0		14 h(m,6)
		RCL				6	m			0		15 h(m-1,k)
		1		065 177		-	-			DSZ		16 m counter
		6	"m"			RCL				COS		17 Z
		HLT				1	Z	105 217		=		18 IND Y_k
030 142		X	X			7				9		19 IND I,h
		IND				X	X			9		FLAGS
		RCL		070 182		RCL				HLT		0
		1	"h(m,d)"			0	X(M)			D'	NEXT	1
		9				7		110 222			ROW	2
035 147		HLT				=						3
		1/x				HLT	"N"	TEXAS INSTRUMENTS INCORPORATED				4
		HLT	"P(o)"	075 187		RCL						

TITLE Interactive LDCS PAGE 1 OF 2

PROGRAMMER E. S. Herndon DATE March 1980

SR-52 Coding Form

LOC	CODE	KEY	COMMENTS	LOC	CODE	KEY	COMMENTS	LOC	CODE	KEY	COMMENTS	LABELS
000 112		LBL				1	Display			RCL		A Z-Think
		E'				9	next 1			1	(m-1)	B M-Load
		STO		040 152		=	to set			5		CN-Multiprog
		1				HLT	s(I)			=		D I, s(I)
		9				E'		080 192		STO	Set	E EXECUTE
005 117		IND				HLT				1	S(L)	A'
		RCL	GENERAL			LBL	MAIN			9		B'
		1	INPUT	045 157		E	EXEC			-	TEST	C' CONT
		9	ROUTINE			0	INITIALIZE			RCL	IF N	D'NEXT STEP
		HLT				STO		085 197		1	Exceeded	E' INPUT
010 122		IND				1	m=0			4		REGISTERS
		STD				5				=		00 loops
		1		050 162		1				INV		01 S(I)
		9				STO	1=h(o)			IF POS		02 S(2)
		RTN				1		090 202		C'	IF SO	03 S(3)
015 127		LBL				8				RCL	Use	04
		A				RCL				1		05
		1	Z	055 167		1				4	S(N)	06
		3				6	M			STO		07
		E'				STO		095 207		1		08
020 132		HLT				0	Loop=M			9		09
		LBL				0				LBL	Continue	10
		B		060 172		-				C'		11 S(11)
		1	L			1				1	M=M+1	12 X(M)
		6				=		100 212		SUM		13 Z Think
025 137		E'				INV	IF LOAD			1		14 N Multi
		HLT				IF ZRO	M =			5		15 m-countr
		LBL		065 177		D	1			+	1+	16 M-load
		C				1	SET			IND		17 h (m-1)
		1	N			STO	H (M-1)	105 217		RCL	S(L)	18 s(m)
030 142		4	Multiprog			1	=1			1		19 IND H, S(L)
		E'				7				9		FLAGS
		HLT		070 182		LBL	MAIN			X		0
		LBL				D'	ENTRY			RCL		1
		D				RCL		110 222		1	M	2
035 147		1	1, S(I)			1	L			5		3
		+				6		TEXAS INSTRUMENTS INCORPORATED				4
		RCL		075 187		-						

TITLE Interactive LDCS PAGE 2 OF 2

PROGRAMMER E. S. Herndon DATE March 1980

SR-52 Coding Form

LOC	CODE	KEY	COMMENTS	LOC	CODE	KEY	COMMENTS	LOC	CODE	KEY	COMMENTS	LABELS
000 112		X				2	1/X(M)			E	Do	A
		RCL				1/x					Next	B
		1	h(m)	040 152		X					Pass	C
		ℰ				RCL						D
		÷				1	m	080 192				E
005 117		RCL				5						A'
		1	Z			-						B'
		3		045 157		RCL						C'
		=				1	Z					D'
		STO				3		085 197				E'
010 122		1				=						REGISTERS
		8				HLT	"R(M)"					00
		DSZ		050 162		RCL						01
		D'				1						02
		RCL	OUTPUTS			5		090 202				03
015 127		1				-						04
		5				RCL						05
		HLT		055 167		1						06
		X				3						07
		RCL				X		095 207				08
020 132		1				RCL						09
		8				1	X(M)					10
		HLT	"h(M)"	060 172		2						11
		1/x				=						12
		HLT	"Po"			HLT	"Q"	100 212				13
025 137		÷				RCL						14
		RCL				1	h(m)					15
		1	2	065 177		8	→					16
		3				STO	h(m-1)					17
		X				1		105 217				18
030 142		RCL				7						19
		1				9						FLAGS
		7		070 182		9	"99"					0
		HLT	"h(m-1)"			HLT	END					1
		=				1		110 222				2
035 147		HLT	"X(M)"			SUM						3
		STO				1	increment					4
		1		075 187		6	L					

TEXAS INSTRUMENTS INCORPORATED

REFERENCES

[1] Denning, P. J. and Buzen J. P. "The Operational Analysis Queuing Network Models" ACM Computing Surveys Vol. 10, No. 3, September 1978.

[2] Herndon, E. S. "An Easy Computation of Terminal Response Time," MITRE Report M78-216, June 1978.

[3] Williams, A. C. and Bhandiwad R. A. "A Generating Function Approach to Queuing Network Analysis of Multiprogramming Computers," Networks 6, ' (1976).

[4] Chandy, K. M. and Sauer C. H. "Approximate Methods for Analyzing Queuing Network Models of Computing Systems," ACM Computing Surveys Vol. 10, No. 3, September 1978, pp 292-293.