Multi-processor software lockout

by STUART E. MADNICK
IBM Cambridge Scientific Center
Cambridge, Massachusetts

INTRODUCTION

As the uses of computers increase, every facility eventually becomes faced with the need to increase capacity. The solutions normally used are to replace the equipment with newer and faster models or obtain additional independent systems. The latter solution, although possibly satisfactory for batch processing, is inefficient for time-sharing and other forms of "computer utility". Technological forecast is beyond the scope of this paper, but present trends indicate that future general-purpose computers will be smaller and less costly, but not significantly faster.

Many people see multi-processor computing systems as the most promising technique for large-scale computer facilities. Projects, such as MIT's Multics, IBM's TSS/360, GE's GECOS, and UNIVAC's EXEC-8, are developing operating systems for multi-processor configurations. Papers by Corbato and Vyssotsky[1] and Dennis[2] give additional arguments for adopting such configurations.

To facilitate system scaling, reliability, and modularity, many multi-processor operating systems are designed to treat the processors as homogeneous system resources. Hence, there is no "supervisor" processor, each schedules and controls itself. To prevent critical races and inconsistent results, only one processor at a time is permitted to alter or examine certain shared system data bases; all other processors attempting simultaneous access are locked-out. This phenomenon is not strictly limited to homogeneous processor systems, similar requirements apply to any multi-processor scheme utilizing shared data bases.

Lock-out is not a novel problem. It is a familiar occurrence on multi-task single-processor computing systems. If two tasks simultaneously request use of the same I/O device, one of the tasks must be locked-out, or blocked, until the device is available. Usually this merely involves temporarily bypassing the conflicting task by continuing execution of other tasks; thus this form of lock-out has only a logical affect on the system resources not a physical affect. Processor lock-out actually results in physically idle resources, since the processor has been directed to discontinue execution of the currently active task but cannot complete the transition to another task due to lockout. At least one paper has noted this phenomenon[3].

Multi-Processor Lockout Model

For convenience and ease of understanding, a simple probabilistic model was used. Each processor is considered as independent. When executing a task, the interval of time until it becomes necessary to access a system data base is described by a negative exponential distribution function with parameter 1/E. The expected execution interval for such a task has value E time units. Similarly, the processor runs in the locked state for an expected interval of length L time units. The ratio, L/(E+L), might be loosely interpreted as the expected portion of total execution time that occurs in the locked state for a multi-task uni-processor configuration (no processor lock-out).

There are three possible states for an individual processor: (1) it is executing in the unlocked state on behalf of a task for an interval of time characterized by parameter 1/E and then will attempt to access a system data base, (2) it is executing in the locked state for an interval characterized by parameter 1/L and then will remove the lock, or (3) it attempted to access the data base but was idled by lock-out. The Markov Model for n processors is illustrated in Figure 1.

$$\begin{tikzcd}
S_0 \arrow[r, bend left, "\frac{n}{E}dt"] & S_1 \arrow[l, bend left, "\frac{1}{L}dt"] \arrow[r, bend left, "\frac{n-1}{E}dt"] & S_2 \arrow[l, bend left, "\frac{1}{L}dt"] \arrow[r, bend left, "\frac{n-2}{E}dt"] & \cdots \arrow[l, bend left, "\frac{1}{L}dt"] \arrow[r, bend left, "\frac{2}{E}dt"] & S_{n-1} \arrow[l, bend left, "\frac{1}{L}dt"] \arrow[r, bend left, "\frac{1}{E}dt"] & S_n \arrow[l, bend left, "\frac{1}{L}dt"]
\end{tikzcd}$$

State S(i) represents state with i processors attempting to access system data base (1 processor actually manipulating data base, i-1 processors awaiting access).

Figure 1. Markov Model

The transitions from state i to state i−1 have a probability of (1/L)dt representing the fact that the amount of time spent in the locked state by a processor is independent of the number of processors waiting to access a locked data base. The transitions from state i to state i+1 have probability (n−i)(1/E)dt representing the fact that each of the n−i processors in execution have probability (1/E)dt of requesting access to the system data bases.

Solving for the steady state probabilities,[4] we find that

$$Pi = \frac{n!}{(n-i)!}\left(\frac{L}{E}\right)^{i} Po \qquad i = l, \ldots, n$$

with

$$Po = \left[\sum_{i=0}^{n} \frac{n!}{(n-i)!}\left(\frac{L}{E}\right)^{i}\right]^{-1}$$

The criterion for performance with which we will be interested is the expected number of processors idled due to lock-out:

$$E\,(idle) = \sum_{i=2}^{n} (i-l)\,Pi$$

The preceeding formula becomes:

$$E\,(idle) = \frac{\displaystyle\sum_{i=2}^{n} \frac{(i-l)}{\left(\frac{E}{L}\right)^{i}(n-i)!}}{\displaystyle\sum_{i=0}^{n} \frac{1}{\left(\frac{E}{L}\right)^{i}(n-i)!}}$$

Results Obtained by Use of Model

The expected number of idle processors was evaluated for certain interesting cases and plotted in Figures 2, 3, and 4. The diagrams demonstrate that the function differs significantly from a linear approximation (expected number of idle processors = number of processors times the portion of time each would individually spend in locked state). In fact for an operating system which has a ratio L/E of .05, the linear approximation is in error by 133% for a 20-processor configuration and by 1700% for a 40-processor configuration.

Figure 3, for example, illustrates the affect of software lock-out on a system characterized by a ratio of .05 for L/E. If there were 15 processors attached to such a system, it is expected that on the average 1 processor will always be idled by software lock-out. Increasing the system to 40 processors results in an expected average of 19 idle processors, a further increase to 41 processors results in 20 idle processors.

Concluding Comments

The notion of "idle" processors must be carefully understood. It has been suggested that a multi-processor system is uneconomic if even 10% of a processor, for example, is known to be idle; therefore, a 15-processor system with 1 processor idle, as described above, is totally unreasonable! A more meaningful way to consider the situation is to note that the 15-processor system has the effective power of 14 processors, whereas a 14-processor system has the effective power of 13.25 processors. Therefore, the 15th processor has a marginal effective power of 75% (¾ of a processor). Such a configuration would be reasonable if either the marginal cost of the additional processor (and associated hardware) is less than 4% of the total system cost assuming some economy of scale, or if the necessity for additional compute power outweighs the cost. On the other hand, under the same conditions, the marginal power of the 41st processor is approximately 0%.

Software locking will be required under a variety of circumstances, such as whenever a task (1) exceeds its time quantum, (2) requests an input/output function, or (3) generates a page or segment fault. Furthermore, it has been suggested that locking will be required on memory allocation,[5] processor allocation,[6] and parameter list validation.[7]

Based upon rather crude measurements performed on the CP/40 Experimental Time Sharing System (paged memory) at the IBM Cambridge Scientific Center[8, 9] and supported by measurements extracted from the SDC system report,[10] the expected number of instructions executed on behalf of a task before locking is required is around 5000 for present-day systems. If we assume that the system remains locked for at least 100 instructions, a value of 2% is obtained for L/E. Therefore, a general range of from 0.1% to 10% might be considered reasonable.

Several operational single processor time-sharing systems have been estimated to spend up to 50% of the execution time performing supervisory functions. If such systems were adapted for multi-processing by indiscriminately locking whenever entering supervisor state, locked-time (L) might easily exceed unlocked (E). A system designed with multi-processor operation carefully planned to minimize lockout occurrences and duration might maintain low values of L/E. Butler Lampson[6] has raised some arguments for providing hardware facilities to assist the supervisor, thereby decreasing the duration of the locked state. As software expenditures increase in relation to hardware costs, this technique may gain additional support.

It is very difficult to predict how the ratio L/E will be affected by an increase in the number of processors. Depending upon effectiveness of shared procedures and amount of main storage added, the paging load might increase or decrease. Furthermore, if more tasks are allowed on the system, the amount of time required by a locked priority scheduler would be expected to increase.

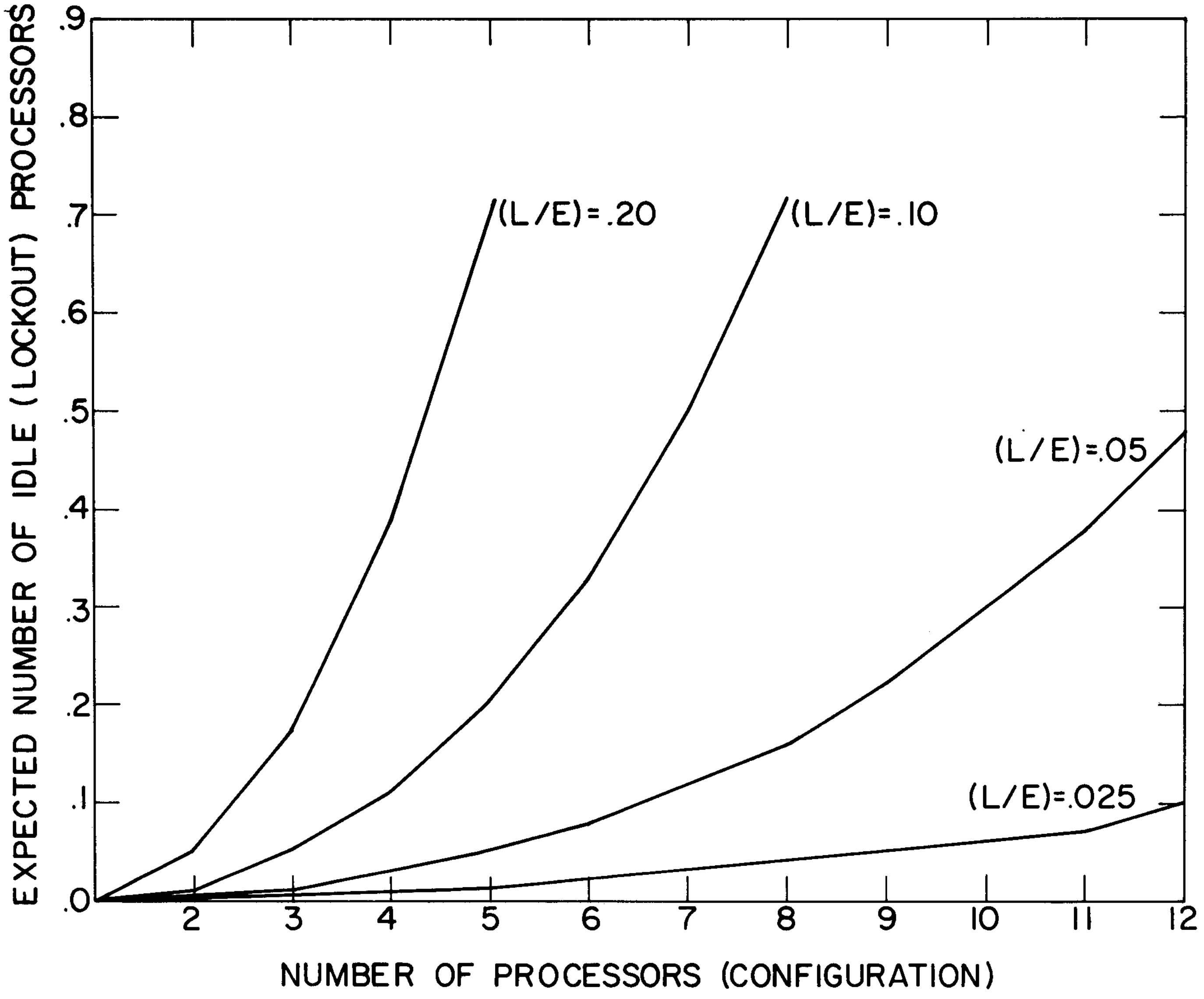
EXPECTED NUMBER OF IDLE (LOCKOUT) PROCESSORS
.9
.8
.7
.6
.5
.4
.3
.2
.1
.0
2
3
4
5
6
7
8
9
10
11
12
NUMBER OF PROCESSORS (CONFIGURATION)
(L/E)=.20
(L/E)=.10
(L/E)=.05
(L/E)=.025

Figure 2.

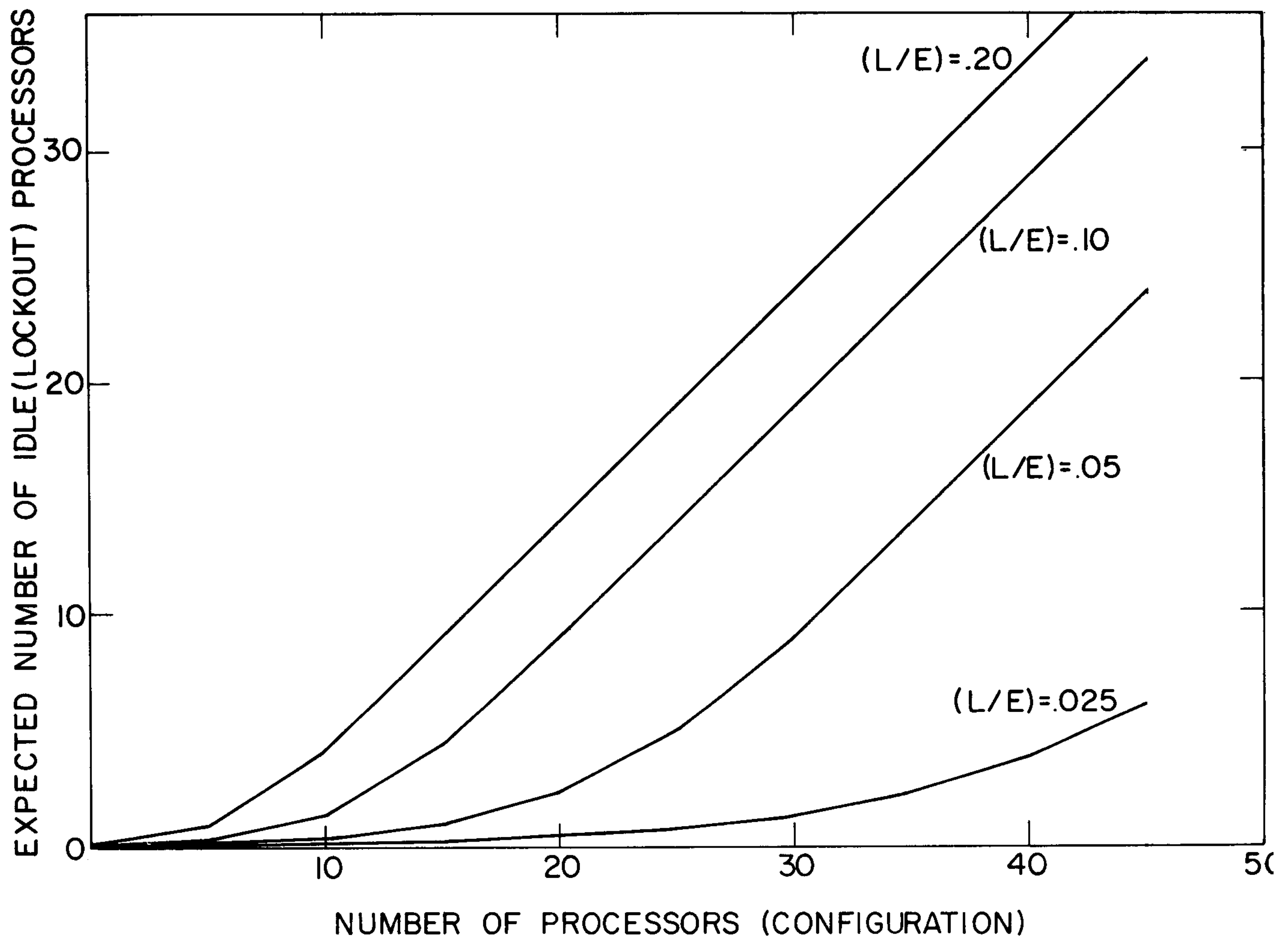
EXPECTED NUMBER OF IDLE(LOCKOUT) PROCESSORS
0
10
20
30
10
20
30
40
5(
NUMBER OF PROCESSORS (CONFIGURATION)
(L/E)=.20
(L/E)=.10
(L/E)=.05
(L/E)=.025

Figure 3.

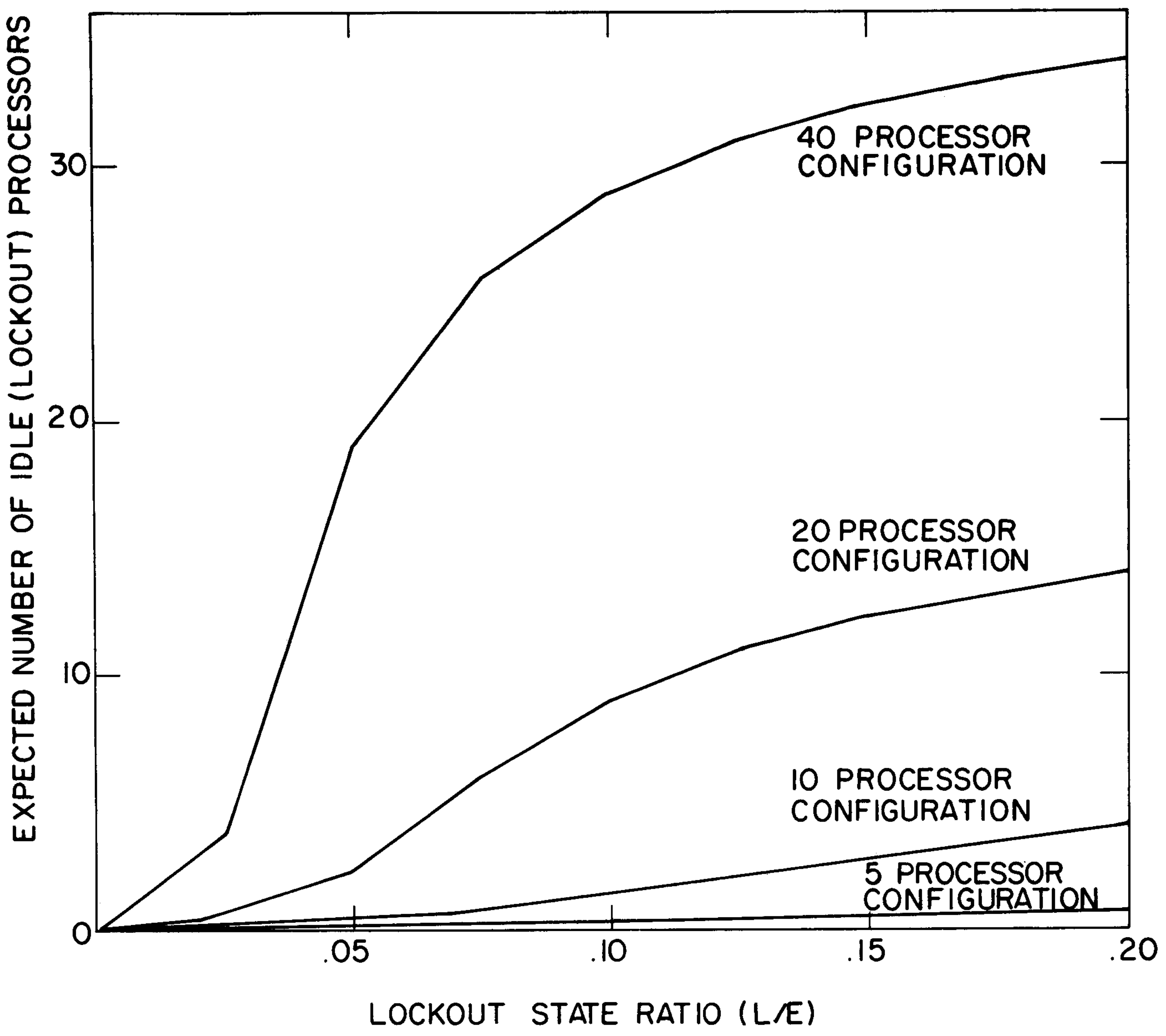
EXPECTED NUMBER OF IDLE (LOCKOUT) PROCESSORS
30
20
10
0
.05
.10
.15
.20
LOCKOUT STATE RATIO (L/E)
40 PROCESSOR CONFIGURATION
20 PROCESSOR CONFIGURATION
10 PROCESSOR CONFIGURATION
5 PROCESSOR CONFIGURATION

Figure 4.

The simple model used was not intended to produce a complete and precise mathematical solution to the entire problem of multi-processor lock-out. The presented quantitative results and accompanying elaboration on the basic phenomenon should provide further insight into its potential importance as a factor in the design of multi-processor operating systems.

ACKNOWLEDGMENTS

The author wishes to thank the many people that graciously contributed suggestions and advice that facilitated the condensed presentation of this complex and somewhat controversial topic, of special note are Bill Harrison, Steve Zilles, Don Hatfield, Tom Rosato, and Liz Levey of IBM., Ed Satterthwaite of Stanford, and Allen Moulton, Paul Mockapetris, Prof. Robert Graham, and Prof. John Donovan of MIT. The writings and teachings of MIT Professors Jerome Saltzer and Alvin Drake must be credited with providing the initial stimulus for undertaking this study.

REFERENCES

1. Corbato F J and Vyssotsky V A
   Introduction and overview of the MULTICS System
   Proceedings AFIPS 1965 Fall Joint Computer Conference Vol 27 Part 1 Spartan Books New York pp 185-196
2. Dennis Jack B
   A position paper on computing and communications
   Communications of the ACM May 1968 Vol 11 No 5
   pp 370-377
3. Saltzer Jerome
   Traffic control in a multiplexed computer system
   MIT Project MAC MAC-TR-30 June 1966
4. Drake, A W
   Fundamentals of applied probability
   McGraw-Hill New York 1962 pp 163-185
5. Fuchel Kurt and Heller Sidney
   Considerations in the design of a multiple computer system with extended core storage
   Communications of the ACM May 1968 Vol 11 No 5
   pp. 334-340
6. Lampson Butler W
   A scheduling philosophy for multiprocessing systems
   Communications of the ACM May 1968 Vol 11 No 5
   pp 347-360
7. Graham Robert M
   Protection in an information processing utility
   Communications of the ACM May 1968 Vol 11 No 5
   pp 365-369
8. Adair R J Bayles R U Comeau L W and Creasy R J
   A virtual machine system for the 360/40
   Cambridge Scientific Center Report 36.010 May 1966
9. Comeau L W
   A study of the effect of user program optimization in a paging system
   ACM Symposium on Operating System Principles
   October 1967 Gatlinburg Tennessee
10. Smith John L
    Multiprogramming under a page on demand strategy
    Communications of the ACM October 1967 Vol 10
    No 10 pp 636-646