Operating systems architecture

by HARRY KATZAN, JR.
Pratt Institute
Brooklyn, New York

INTRODUCTION

Operating systems architecture refers to the overall design of hardware and software components and their operational effectiveness as a whole. To be effective, however, an operating system must not only be cognizant of the collection of hardware and software modules, but must also be designed in light of the programs and data which the system processes and the people which it serves. The absence of formal theory on operating systems and the lack of standard terminology have caused much confusion among users. The problem is particularly apparent when comparing systems where the same terms are applied to a variety of concepts and levels of implementation.

The purposes of this paper are threefold: (1) to present the basic properties with which operating systems can be grouped, classified, and evaluated; (2) to identify the major categories into which operating systems can be classed and to give concrete examples of each; and (3) to discuss resource management in operating systems with an emphasis on storage allocation and processor scheduling.

First, seven properties, used to classify operating systems, are briefly described. Then, the major categories into which operating systems can be classed and to give concrete examples of each; and (3) to discuss resource management in operating systems with an emphasis on storage allocation and processor scheduling.

First, seven properties, used to classify operating systems, are briefly described. Then, the major categories into which operating systems can be classed and to give concrete examples of each; and (3) to discuss resource management in operating systems with an emphasis on storage allocation and processor scheduling.

PROPERTIES OF OPERATING SYSTEMS

An operating system tends to be classified, informally, on the basis of how the facilities of the system are managed or allocated to the user. Accordingly, the number of properties, or combinations of properties, which contribute to the classification, is very large. Seven of these properties dominate the remainder and are introduced in the following paragraphs.

Access

Access is concerned with how the user interacts with the system. Does he access the system via a remote terminal or does he submit his work in a batch processing environment? If the user is at a remote location, what is the nature of his terminal device? Is it a RJE/RJO work station or is it a keyboard or CRT type device? Is a command system available so that the user can enter into a dialogue with the operating system? Can the user initiate batch jobs or query the status of them when at a remote terminal? Does the facility exist for conversing with a problem program from a terminal device?

Utilization

Utilization is concerned with the manner in which the system is used. Is the system closed so that the user is limited to a specific programming language or is the system open allowing the user access to all of the system facilities? How must the user structure his programs—planned overlay, dynamic segmentation, single-level store? Can the user prepare and debug programs on line or is he limited to querying the system? What facilities are available for data editing and retrieval? In the area of data management, what access methods, file organization, and record types are permitted? Does the data management system have provisions for using the internal and external storage management facilities of the system? Lastly, what execution-time options are permitted by the operating system at run time as compared to compile time?

Performance

Performance deals with the quality of service to the installation and to the user. Does the operating system design philosophy attempt to maximize the use of
system resources, maximize throughput, or guarantee a given level of terminal response? What is the probability that the system will be available to the user when needed? Does the user lose his data sets if the system fails (data set integrity)? What facilities are available for system error recovery?

**Scheduling**

Scheduling determines how processing time is allocated to the jobs (or tasks) which reside in some form in the system. What scheduling philosophy is used—sequential, natural wait, priority, time slicing, demand? What is the nature of the scheduling algorithm—round robin, exponential with priority queues, table driven?

**Storage management**

Storage management is concerned with how main storage and external storage are allocated to the users. Is main storage fragmented, divided into logical regions, allocated on a page basis or are programs swapped? Is external storage allocated in fixed-size increments or on a demand basis with secondary allocations, as required?

**Sharing**

Sharing is the functional capability of the system to share programs, data, and hardware devices. Does the system permit readonly and re-entrant code so that programs can be shared during execution? Can data sets be shared without duplicating them? At what level of access? Can hardware devices be shared among users giving each the illusion that he has a logical device to himself?

**Configuration management**

Configuration management is concerned with the real physical system and the logical system as seen by the user. Physically, how is the system organized and how can this organization be varied? Does the capability exist of partitioning off a maintenance subsystem? Similarly, can a failing CPU, core box, channel, or I/O device be removed from the system? Logically, does the user have a machine to himself, a large virtual memory, a fixed partition?

Obviously, the properties are not exhaustive in the sense that all operating systems, real or hypothetical, can be automatically classified. The properties do form a basis for comparison and are used in the next section to identify the major categories of operating systems.

**CATEGORIES OF OPERATING SYSTEMS**

An operating system is an integrated set of control programs and processing programs designed to maximize the overall operating effectiveness of a computer system. Early operating systems increased system performance by simplifying the operations side of the system. Current operating systems additionally attempt to maximize the use of hardware resources while maintaining a high level of work throughput or providing a certain level of terminal response. A multitude of programmer services are usually provided, as well.

**Multiprogramming**

A multiprogramming system is an operating system designed to maintain a high level of work throughput while maximizing the use of hardware resources. As each job enters the system, an internal priority, which is a function of external priority and arrival sequence, is developed. This internal priority is used for processor scheduling. During multiprogramming operation, the program with the highest internal priority runs until a natural wait is encountered. While this wait is being serviced, processor control is turned over to the program with the next highest priority until the first program’s wait is satisfied, at which time, processor control is returned to the high priority program, regardless if the second program can still make use of the system. The first job has, in a sense, demanded control of the system. The concept is usually extended to several levels and is termed the level of multiprogramming. A multiprogramming system is characterized by: (1) Limited access traditionally limited to tape or card SYSIN and printer or tape SYSOUT. RJE/RJO may be implemented but on-line real-time processing usually requires a specially written problem program. (2) Utilization is most frequently restricted to batch type operations with data management facilities being provided by the system. Planned overlay is usually required for large programs with most debugging being done off line. (3) Performance is oriented towards high throughput and maximum utilization of hardware resources. A given level of service is not guaranteed and the processing of jobs is determined by operational procedures. (4) Scheduling of work usually involves priority, natural wait, and demand techniques. In some systems, a unit of work may spawn other units of work providing parallel processing, to some degree. (5) Storage management techniques vary between sys-
tems with most systems using a fixed partition size or a logical region for problem programs. Paging techniques with dynamic address translation have been used with some success. (6) Sharing of system routines is frequently provided while the sharing of problem programs is a rarity. When a central catalog of data set information is provided, data set sharing, at various levels of access, is available. Otherwise, data set sharing is accomplished on an ad hoc basis. (7) Configuration management is usually limited to the existing physical system with the users being given a portion, fixed or variable, of actual storage.

**Hypervisor multiprogramming**

One of the problems frequently faced by installation management involves running two different operating systems, each of which requires a dedicated but identical machine. A hypervisor is a control program that, along with a special hardware feature, permits two operating systems to share a common computing system. A relatively small hypervisor control program (see Figure 1) is required which interfaces the two systems. Although only one processor is involved, a hardware prefix register divides storage into two logically separate memories, each of which is utilized by an operating system. I/O channels and devices are dedicated to one or the other operating system and use the hardware prefix register to know to which half of storage to go. All interrupts are indirectly routed to a common interrupt routine which decides which operating system should receive the most recent interrupt. Processor control is then passed to the hypervisor control program for dispatching. The hypervisor control program loads the prefix register and usually dispatches processor control to the operating system which received the last interrupt. Alternate dispatching rules are to give one operating system priority over another or to give one operating system control of the processor after a fixed number of interrupts have been received by the other side. Hypervisors are particularly useful when it is necessary to run an emulator and an operating system at the same time. Similar to multiprogramming systems, a hypervisor is characterized by: (1) limited access; (2) batch utilization; (3) high throughput performance; (4) priority, natural wait, and demand scheduling; (5) basic storage management techniques; (6) limited sharing facilities; and (7) configuration management determined by the operating systems that are run as subsystems.

**Time sharing**

Although time-sharing is used in a variety of contexts, it most frequently refers to the allocation of hardware resources to several users in a time dependent fashion. More specifically, a time-sharing system concurrently supports multiple remote users engaging in a series of interactions with the system to develop or debug a program, run a program, or obtain information from the system. The basic philosophy behind time sharing is to give the remote user the operational advantages of having a machine to himself by using his think, reaction, or I/O time to run other programs. Operation of a time sharing system is summarized as follows:

Time-shared operation of a computer system permits the allocation of both space and time on a temporary and dynamically changing basis. Several user programs can reside in computer storage at one time while many others reside temporarily on auxiliary storage such as disc or drum. Computer control is turned over to a resident program for a scheduled time interval or until the program reaches

---

*See reference [1], p. 190.
a relay point (such as an I/O operation), depending upon the priority structure and control algorithm. At this time, processor control is turned over to another program. A non-active program may continue to reside in computer storage or may be moved to auxiliary storage, to make room for other programs, and subsequently be reloaded when its next turn for machine use occurs.

A time-sharing system is characterized by: (1) Remote access with keyboard or CRT devices and possibly RJE/RJO work stations. (2) Varied utilization ranging from a closed system such as QUIKTRAN4, APL/3605, or BASIC6, to an open system such as MULTICS7 or TSS/3608,9. In most open systems, a single-level store, on-line debugging facilities, and an extensive file system are also available. (3) Performance in most time sharing systems is mainly centered around dividing processor time among the users and providing fast response to terminal requests. Management of other resources in a time-sharing system is usually towards this end. (4) A given level of user service is maintained by giving users a short slice of processor time at frequent intervals according to a scheduling algorithm. The most frequently used scheduling algorithms are round robin and exponential with priority queues. (5) Varied storage management techniques are used depending upon the hardware and the sophistication of the software. Swapping and paging techniques have been used with great success. In the latter category, direct-access storage devices and large capacity core storage have both been used as paging devices. (6) Most open time-sharing systems permit code sharing during execution and language processors, data management routines, and command system programs are frequently shared. If public storage is provided, then data sets sharing is also available. In closed systems, the level of sharing is determined by the programming language used and its method of implementation. (7) In a utility class time-sharing system, configuration management facilities are required for preventative maintenance and for the repairing of faulty equipment. Multiple processors, storage units, and data channels are provided with many large time-sharing systems; thus the hardware resources are available for configuring the system to meet operational needs. In some time-sharing systems, the user has a logical machine to himself provided through a combination of hardware and software facilities. This topic is covered in the next section on virtual machines.

Virtual systems

A virtual system is one which provides a logical resource which does not necessarily have a physical counterpart. Virtual storage systems7,4,5,8,10,11 (see Figure 2) are widely known and provide the user with a large single level store achieved through a combination of hardware and software components. A virtual storage system is characterized by the fact that real storage

---

**Figure 2—Virtual storage**

---

**Figure 3—Loaded virtual storage**

From the collection of the Computer History Museum (www.computerhistory.org)
contains only that part of a user's program which need be there for execution to proceed. The basic philosophy of virtual storage lends itself to paging (Figure 3) and is usually associated with dynamic address translation, as introduced later in the paper.

A virtual machine is an extension to the virtual storage concept which gives the user a logical replica of an actual hardware system. Whereas in a virtual storage system, a user could run programs, in a virtual machine, a user or installation can run complete operating systems. In addition to using the virtual storage concept, a virtual machine system contains a control program which allocates resources to the respective virtual machines and processes privileged instructions which are issued by a particular operating system.

Although virtual systems are usually associated with time-sharing, the concept is more general and applies equally well to multiprogramming systems. Virtual systems tend to be most effective in operating environments where dynamic storage allocation, dynamic program relocation, simple program structure, and scheduling algorithms are of concern. Virtual systems using fixed size pages and dynamic address translation also lend themselves to sharing and most systems using this design philosophy have implemented code sharing during execution to some extent.

**Tri-level operating systems**

In a conventional operating system (Figure 4), two levels of control are available, each of which corresponds to a segment of core storage. Level one contains the supervisor program and all associated routines for job control and data management. User programs, language processors, and utility programs run at level two and are regarded as problem programs by the supervisor program. Control is passed from level two to level one by hardware facilities, usually termed an interrupt, and level one services are able to completely sustain level two needs.

In a virtual system, another level of complexity is required. Logical as well as physical resources must be maintained and allocated. Thus, in virtual systems, allocation of resources is relegated to the supervisor or control program and typical job control and data management routines are included as a job monitor program (Figure 5) which exists as a second level.

* Such as CPU time, core storage, and I/O facilities.
RESOURCE MANAGEMENT

In modern operating systems, the allocation of hardware resources among users is a major task. Two resources directly affect performance and utilization: storage management and scheduling. Both topics were introduced earlier. The most widely used implementation techniques are discussed here.

Storage management

In either a 2-level or 3-level operating system, available storage is divided into two areas: a fixed area for the supervisor program and a dynamic area for the user programs. If no multiprogramming or time sharing is done, then a user program executes serially in the dynamic area. When he has completed his use of the CPU, then the dynamic area is allocated to the next user.

When more than one user shares the dynamic area, such as in multiprogramming or time sharing, then storage management becomes a problem for which various techniques have been developed. They are arbitrarily classed as multiprogramming techniques or time-sharing techniques although the point of departure is not well-defined. Multiprogramming techniques include fixed partition, region allocation, and roll in/roll out. Time-sharing techniques include core resident, swapping, and paging.

In a fixed partition system, the dynamic storage area is divided into fixed sub-areas called partitions. As a job enters the system, it specifies how much storage it needs. On the basis of the space requirements specified, it is assigned to a fixed partition and must operate within that area using planned program structure whenever necessary. In a region allocation system, a variable number of jobs may use the system. Just before a job is initiated, a request is made to dynamically allocate enough storage to that job. Once a job is initiated, however, it is constrained to operate within that region. In a logical sense, fences are created within the dynamic area. Roll in/roll out is a variation of region allocation which effectively enables one job to borrow from another job if space requirements can not be fulfilled from the dynamic area. The borrowed region is rolled back in and returned to the original owner whenever he demands the CPU or when the space is no longer needed by the borrower.

The most fundamental technique for storage management in time sharing is core resident. In a core resident system, all active tasks are kept in main storage. This method reduces system overhead and I/O activity but is obviously limited by the size of core storage. Large capacity storage (LCS) is frequently used in a hierarchical sense with main storage and provides a cost effective means of increasing the number of potential users. Large capacity storage is sufficiently fast to satisfy the operational needs of a user at a remote terminal. Swapping is the most frequently used method of storage management in time sharing. At the end of a time slice, user A's program is written out to auxiliary storage and user B's is brought in for execution. All necessary control information is saved between invocations. In the above case, the system would have to wait while user B's program was brought in for execution. Thus, two or more partitions can be used for swapping to reduce the I/O wait. The use of several partitions permits other user programs to be on their way in or on their way out while one user's program is executing. This method reduces wait time but increases the amount of system housekeeping and overhead. A variation to the single partition approach is the onion-skin method used with the CTSS system at M.I.T. With this method, only enough of user A's program is written out to accommodate user B. In a sense, user A's program is peeled back for user B's program. If user C requires still more space than B, then A is peeled back even more. In a paging system, main storage is divided into fixed-size blocks called pages. Pages are allocated to users as needed and a single user's program need not occupy consecutive pages, as implied in Figure 3. Thus a translation is required between a user's virtual storage, which is contiguous, and real storage, which is not. A technique called dynamic address translation is employed that uses a table look up, implemented in hardware, to perform the translation. First, the address field is segmented to permit a hierarchical set of look up tables (Figure 6). Then, each effective computer address goes through an address translation process (Figure 7) before operands are fetched from storage. The process is usually speeded up with a small associative memory (Figure 8). When a user program references a page that is not in main storage, a hardware interrupt is generated. The interrupt is fielded by the supervisor program which brings the needed page in for execution. Meanwhile, another user can use the processor. Look up tables (Figure 7) are maintained such that when a page is brought into

<table>
<thead>
<tr>
<th>Segment</th>
<th>Page</th>
<th>Byte</th>
</tr>
</thead>
</table>

Figure 6—Segmentation
main storage, an entry is made to correspond to its relative location in the user's virtual storage.

The methods vary, obviously, in complexity. An eventual choice on which technique to employ depends solely on the sophistication of the operating system, the access, performance, and utilization required, and the underlying hardware.

**Scheduling**

In modern operating systems, the supervisor program assumes the highest priority and essentially processes and does the housekeeping for interrupts generated by problem programs and external and I/O devices. In this sense, the supervisor (or the system) is interrupt driven. It is generally hoped that the processing done by the supervisor is kept to a minimum. When the supervisor has completed all of its tasks, it must decide to whom the processor should be allocated. In a single job system, the running program simply retains control of the processor. In a multi-job batch environment, where the system is performance oriented but not response oriented, the processor is usually given to the highest priority job that demands it. This philosophy is generally termed multiprogramming as discussed previously.

In a time-sharing environment, performance is measured in terms of terminal response, and processor scheduling is oriented towards that end. Thus, a user is given a slice of processor time on a periodic basis—frequently enough to give him the operational advantage of having a machine to himself. The scheduling philosophy is influenced by the user environment (i.e., compute-bound jobs, small jobs, response-oriented jobs) and the method of storage management. Three strategies have been used frequently enough to warrant consideration. The most straightforward method is *round robin*. Jobs are ordered in a list on a first-in-first-out basis. Whenever a job reaches the end of a time slice or it can no longer use the processor for some reason, it is placed on the end of the list and the next job in line is given a slice of processor time. A strict round robin strategy favors “compute” jobs and “terminal response” jobs equally and tends to be best suited to a core resident storage management system. With an *exponential scheduling* strategy, several first-in-first-out lists are maintained, each with a given priority. As a job enters the system, it is assigned to a list on the basis of its storage requirements—with lower storage
requirements being assigned a higher priority since they facilitate storage management. The scheduling lists are satisfied on a priority basis, no list is serviced unless higher priority lists have been completed. Terminal (or response) oriented jobs are kept in the highest priority list—thus assuring rapid terminal response. If a job is computing at the end of its time slice, then it is placed at the end of the next lowest priority list. However, lower priority lists are given longer time slices, of the order \(2t, 4t, 8t, \ldots\), so that once in execution, a compute-bound job stays in execution longer. Exponential scheduling has “human factors” appeal in that a terminal-oriented user, who gets frequent time slices, is very aware of his program behavior whereas the program behavior of a compute-bound user is generally transparent to him. One of the biggest problems in processor scheduling is the difficulty in developing an algorithm to satisfy all users. The schedule table strategy is an attempt to do that. Each user is given a profile in a schedule table. When a job enters the system, it is assigned default values. As the job develops a history, however, the table values are modified according to the dynamic nature of the program. The scheduler is programmed to use the schedule table in allocating the processor while satisfying both user and installation objectives. The schedule table approach is particularly useful in a paging environment where certain programs require an excess of pages for execution. Once the required pages have been brought into main storage, then the job can be given an appropriate slice of processor time.

Scheduling strategies differ to the extent that a different one probably exists for each installation that is developing one. As such, scheduling algorithms continue to be the object of mathematical description and analysis by simulation.

THE LITERATURE

There are a wealth of good papers on operating systems in the computer literature. In fact, the volume is so great that a literature survey would invariably do injustice to a great many competent authors. In spite of this initial disadvantage, a sample of interesting papers will be mentioned.

Dynamic storage allocation, storage hierarchy, and large capacity storage have been studied in detail by Randell and Kuehner\(^{14}\), Freeman\(^{15}\), Lauer\(^{16}\), and Fikes, Lauer, and Vareha\(^{17}\).

Performance, program behavior, and the analysis of system characteristics have been reported by Belady\(^{18}\), Fine, Jackson, and McIsaac\(^{19}\), Wallace and Mason\(^{20}\), Coffman and Varian\(^{21}\), Randell\(^{22}\), Dennis\(^{23}\), Den-

The IBM collection entitled, TSS/360 Compendium, contains the following papers and reports:


From the collection of the Computer History Museum (www.computerhistory.org)
McKeehan, J. B., "An Analysis of the TSS/360 Command System II."

Johnson, O. W., and J. R. Martinson, "Virtual Memory in Time Sharing System/360."


SUMMARY

Seven properties were introduced for the description, classification, and comparison of operating systems: access, utilization, performance, scheduling, storage management, sharing, and configuration management. On the basis of these properties, the following types of operating system were identified: multiprogramming, hypervisor multiprogramming, time sharing, virtual systems, and tri-level operating systems. Lastly, two major areas of resource management were discussed: storage management and scheduling. Generally, storage management techniques can be classified as to whether they apply to multiprogramming or time-sharing—although the dividing line is not well-defined. Multi-programming techniques presented were: fixed partition, region allocation, and roll in/roll out. Time-sharing techniques included: core resident, swapping, and paging. Scheduling methods are similarly related to either multiprogramming or time sharing. After a brief discussion, the following time-sharing scheduling philosophies were introduced: round robin, exponential, and the schedule table.

Although formal methods have not been applied to any great extent to operating systems, the interest level is high and many related papers exist in the literature. Operating system technology continues as one of the more challenging areas in the field of computer science.

REFERENCES

1 H Katzan
Advanced programming: Programming and operating systems
Van Nostrand Reinhold Company 1970

2 A J Critchlow
Generalized multiprogramming and multiprogramming systems
Proceedings of the Fall Joint Computer Conference 1963

3 S Rosen
IBM operating system/360 concepts and facilities

4 T M Dunn J H Morrissey
Remote computing—An experimental system. Part I: External specifications
Proceedings of the Spring Joint Computer Conference 1964

5 A D Falkoff K E Iverson
APL/360 user’s manual
IBM Thomas J. Watson Research Center Yorktown Heights
New York 1968

6 J G Kemeny T J Kurtz
Basic
Dartmouth College Computation Center Hanover New Hampshire 1965

7 F J Corrado V A Vysotsky
Introduction and overview of the MULTICS system
Proceedings of the Fall Joint Computer Conference 1965

8 W T Comfort
A computing system design for user service
Proceedings of the Fall Joint Computer Conference 1965

9 C T Gibson
Time-sharing in the IBM system/360: Model 67
Proceedings of the Spring Joint Computer Conference 1966

10 A S Lett W L Konigsford
TSS/360: A time-shared operating system
Proceedings of the Fall Joint Computer Conference 1968

11 N Weizer G Oppenheimer
Virtual memory management in a paging environment
Proceedings of the Spring Joint Computer Conference 1969

12 An introduction to CP-67/CMS
IBM Cambridge Scientific Center Report 320–2032
Cambridge Massachusetts 1969

13 F J Corrado et al
The compatible time-sharing system
The MIT Press Cambridge Massachusetts 1963

14 B Randell C J Kuehner
Dynamic storage allocation systems
Communications of the ACM May 1968

15 D N Freeman
A storage-hierarchy system for batch processing
Proceedings of the Spring Joint Computer Conference 1968

16 H C Laufer
Bulk core in a 360/67 time-sharing system
Proceedings of the Fall Joint Computer Conference 1967

17 R E Fikes H C Laufer A L Vareha
Steps toward a general-purpose time-sharing system using large capacity core storage and TSS/360
Proceedings of the 1968 ACM National Conference

18 L A Belady
A study of replacement algorithms for a virtual storage computer
IBM Systems Journal Volume 4 No 2 1966

19 G H Fine C W Jackson P V McIsaac
Dynamic program behavior under paging
Proceedings of the 1966 ACM National Conference

20 W L Wallace D L Mason
Degree of multiprogramming in page-on-demand systems
Communications of the ACM June 1968

21 E G Coffman L C Varian
Further experimental data on the behavior of programs in a paging environment
Communications of the ACM July 1968

22 B Randell
A note on storage fragmentation and program segmentation
Communications of the ACM July 1969

23 J B Dennis
Segmentation and the design of multiprogrammed computer systems
Journal of the ACM Volume 12 No 4 1965

24 P J Denning
The working set model for program behavior
Communications of the ACM May 1968

From the collection of the Computer History Museum (www.computerhistory.org)
25 P J DENNING
A statistical model for console behavior in multiuser computers
Communications of the ACM September 1968

26 P J DENNING
Thrashing: Its causes and prevention
Proceedings of the Fall Joint Computer Conference 1968

27 S STIMLER
Some criteria for time-sharing system performance
Communications of the ACM January 1969

28 S MADNICK
Multi-processor software lockout
Proceedings of the 1968 ACM National Conference

29 A N HABERMANN
Prevention of system deadlocks
Communications of the ACM July 1969

30 G ESTRIN L KLEINROCK
Measures, models, and measurements for time-shared computer utilities
Proceedings of the 1967 ACM National Conference

31 F D SHULMAN
Hardware measurement device for IBM system/360 time-sharing evaluation
Proceedings of the 1967 ACM National Conference

32 L A BELADY R A NELSON G S SHEDLER
An anomaly in space-time characteristics of certain programs running in a paging machine
Communications of the ACM June 1969

33 P WEGNER
Machine organization for multiprogramming
Proceedings of the 1967 ACM National Conference

34 P WEGNER
Programming languages, information structures, and machine organization
McGraw-Hill Book Company 1968

35 R W O'NEILL
Experience using a time-sharing multiprogramming system with dynamic address relocation hardware
Proceedings of the Spring Joint Computer Conference 1967

36 B W ARDEN R A GALLER R C O'BRIEN
F N WESTERVET
Program and addressing structure in a time-sharing environment
Journal of the ACM Volume 13 No 1 1966

37 G F BADGER E A JOHNSON R W PHILIPS
The Pitt time-sharing system for the IBM systems 360
Proceedings of the Fall Joint Computer Conference 1968

38 J I SCHWARTZ E G COFFMAN C WEISSMAN
A general purpose time-sharing system
Proceedings of the Spring Joint Computer Conference 1964

39 M J MENDELSO A W ENGLAND
The SDS SIGMA 7: A real-time, time-sharing computer
Proceedings of the Fall Joint Computer Conference 1966

40 H A KINSLOW
The time-sharing monitor system
Proceedings of the Fall Joint Computer Conference 1964

41 W M MCKEEMAN
Language directed computer design
Proceedings of the Fall Joint Computer Conference 1967

42 A J PERLIS
The synthesis of algorithmic systems
Proceedings of the 1966 ACM National Conference