SURVEY OF RECENT OPERATING SYSTEMS RESEARCH, DESIGNS AND IMPLEMENTATIONS

C. MOHAN

Department of Computer Sciences
University of Texas at Austin
Austin, Texas 78712

OCTOBER 1977

TR-75

1.0 INTRODUCTION

This paper surveys the recent theoretical and experimental advancements in the operating systems area. On the theoretical side much effort is being expended in rigorously defining the specifications and the logical structure (the hierarchical levels approach, the kernel approach with user defined subsystems, and so on) of an operating system, deriving proofs about the satisfaction of many properties (deadlock-free, synchronization constraints, protection requirements, and so on) and in proposing new theories, models and constructs. Researchers make extensive use of the results in the areas of graph theory, automata theory, operations research, computer architecture and software engineering.

On the experimental side new operating systems with novel features are being designed and implemented. With the increased use of computers for solving many complex tasks on dedicated and general purpose systems, operating systems are required to provide many new facilities. Existing systems are being used for various experimental investigations testing different hypotheses (about new memory management policies, scheduling disciplines, interprocess communication facilities, performance monitoring, process structures, and so on). Operating systems are being modified to enhance their capabilities (like providing virtual machines, support for new devices and multilevel storage hierarchies, parallel processing capabilities, and so on). The developments in the areas of firmware, data base management and distributed processing are also having a great impact on the operating systems area. As one reads through this paper the interplay of these various factors will become apparent. This paper is the result of an attempt to collect together much information and provide pointers to the relevant literature.

Most of this work was done while the author was a chemical engineering undergraduate student at the Indian Institute of Technology - Madras, India.

2.0 AXIOMATICS

The ever increasing software development and maintenance costs have resulted in great stress being placed on program verification [ACM 1975c] and development of appropriate techniques for producing reliable software [ACM 1975b, 1977], as even a casual reader of the software engineering literature would have noticed. Many new programming methodologies, and specification techniques (like data and control abstractions) and languages (like ALPHARD, CLU, Concurrent PASCAL, EUCLID, GYPSY, MESA and MODULA) have been proposed. These developments are already having a great impact on the design of operating systems [Feiertag 1977, Flon 1976, Neumann 1975, Parnas 1976, Popek 1975c, Robinson 1975a b, Saxena 1976a, Silberschatz 1976a c]. Saxena has presented the design, specifications and verification of a hierarchical operating system (supporting concurrent processes) using structured programming techniques and recently developed proof techniques. A specification language is used to define the abstract machines. Assertions are used to define the interconnections among levels [Robinson 1975b]. Recently the multilevel hierarchical approach to operating system construction has been thoroughly analysed in [Berry 1977] using the Contour Model. The abstract machines of the different levels are characterized in terms of nondeterministic information structure models and mappings amongst them.

At SRI Neumann and others have developed a methodology and the necessary tools (like the specification language SPECIAL, the Specification Analyzer, the Hierarchy Manager, the Development Data-Base Manager and so on) for the formal design specification of a Provably Secure Operating System (PSOS) and the proof of satisfaction of certain security properties. This methodology has also been applied to the design of several application subsystems for PSOS (including support for multilevel security classifications, for confined subsystems, for a secure relational data management system and for monitoring security). A very detailed description of the results of this project can be found in [Neumann 1977].

The axiomatic proof techniques initially applied to sequential programs [London 1977] are being extended to parallel programs as well [Andler 1977, Ariely 1975, Flon 1977a b, Hoare 1975, Keller 1976 1977, Lamsveerde 1976, Lipton 1975, Owicki 1976a b, Silberschatz 1977]. Since operating systems consist mostly of parallel processes, the above developments are bound to have a great impact on the design of reliable operating systems.

2.1 Models Of Parallel Computation

Research has been directed at discovering better ways to represent and analyse parallel computations. As a result many models (most of them based on directed graphs) have been proposed. To provide a good theoretical basis Zave has presented a formal definition of parallel processes [Zave 1976]. Newton has developed a graph model, called the Set of Interacting Procedures model, for representing interacting computations [Newton 1975]. A generalized model for concurrent control structures, called G-net, has been proposed by Yeh [Yeh 1975]. He has used it for determining the correctness properties of interacting parallel processes.

Keller has presented an abstract model and a parallel program model, and an induction principle, which has been demonstrated to aid in proving correctness. The latter model differentiates between control and data states. Keller has given a formalization of deadlock also [Keller 1976]. Recently he has discussed the concepts of networks of operators on data types and has extended previous work, in the context of parallel programs, to new data types and indeterminate operators. He has presented the formal denotational semantics of such networks and has given techniques for verifying the correctness of their properties [Keller 1977]. Owicki has developed axioms and inference rules for establishing partial correctness of programs expressed in two parallel programming languages which she has defined (the General Parallel Language and the Restricted Parallel Language). Such partial correctness proofs have been shown to aid in establishing program termination, mutual exclusion and deadlock absence [Owicki 1975].

Owicki's results have been used by Flon in developing verification techniques for operating systems [Flon 1977a b]. Flon has also presented a methodology based on abstract data types for the specification and verification of operating system software. He has also used the Weakest Pre-condition concept [Dijkstra 1976] to prove total correctness of parallel programs. Griffiths has presented the high level language SAL for specifying synchronization problems among communicating concurrent processes in a synthesis/verification system [Griffiths 1975]. The concepts of L systems have also been applied to prove the correctness or incorrectness of the interactions among two or more parallel processes [Ellis 1975]. Ellis has indicated that this proof procedure could be mechanised.

3.0 KERNEL APPROACH

The trend, of late, has been more and more towards the design of kernel based operating systems, which, hopefully could be proved and certified to be correct [Bell 1975, Nehmer 1975, Popek 1974, Reed 1976, Schiller 1975, Walton 1975]. The kernel approach has many nice properties. It supports modularity and the separation of policies and mechanisms. Work is being done at the MITRE corporation in developing a security kernel for a PDP 11/45. This kernel design is being rigorously validated by using axiomatic proof techniques [Millen 1976].

The implemented kernel based operating systems are: Queen's University's [Kingston, Canada] QUIKDOS [MacEwen 1976], University of Toronto's SUE [Sevick 1972], Texas Instrument's DSOS [Frailey 1975], Carnegie Mellon's HYDRA [Wulf 1975] and the Cm* operating system [Fuller 1977], UCLA's PDP 11 virtual machine kernel [Popek 1975c] and University of Rochester's ALEPH [Ball 1976]. The Multics operating system is being modified to support the kernel approach [Montgomery 1976, Reed 1976].

4.0 SECURITY AND PROTECTION

Another important reason for the emergence of kernel based systems is the growing concern in providing computer security [Farber 1975, Fletcher 1975, Jones 1975a b, Neumann 1976, Popek 1975a b c, Schroeder 1975, Stahl 1974, Waguespack 1975, Walter 1974 1975, White 1975] and protection in operating systems [Ames 1975, Andrews 1974, Bratt 1975, Cohen 1976, Conn 1975, Denning 1975 1976a b, Ekanadham 1976a b, Ellis 1974 , Harrison 1975 1976, IRIA 1974, Lipner 1975, Redell 1974, Sevick 1975, Snyder 1977], in the wake of widespread usage of computers for the creation and manipulation of large data bases of vital and sensitive information which have to be guarded against unauthorized access [Kerr 1975]. Much theoretical and practical work has been and is being done in the protection area.

4.1 Theory

Cohen has formalized the notions of problem, mechanism and solution, and has shown how protection, synchronization, and parallel and sequential control mechanisms can be modeled. Then he has shown how, using his formalisms, many problems could be provably solved [Cohen 1976a]. He has also developed a formalism, called Strong Dependancy, for analysing information transmission, in computational systems, to enforce controlled information sharing [Cohen 1976b]. Dorothy Denning has studied the problem of information transmission among security classes. A lattice model, useful for formulating and enforcing security constraints, has been developed by her. This model is helpful in constructing automatic program certification mechanisms also [Denning 1975 1976a b]. Jones has given precise definitions of security policies and protection mechanisms, and the concept of 'soundness', which is used to verify whether a mechanism enforces a particular policy [Jones 1975b].

Methods for proving that an operating system enforces 'confinement' have been formulated [Lipner 1975]. Harrison and co-workers have developed a model of protection mechanisms and have shown as to how , for a system based on Lampson's Access Matrix, the problem of determining whether a subject can acquire a given right to an object (termed the 'safety' problem) is decidable under certain conditions and undecidable otherwise [Harrison 1976]. A linear time algorithm for deciding the Safety of a particular protection mechanism, the 'take and grant' system, has been presented in [Lipton 1977]. Recently Snyder has discussed a formal approach (graph theory based) to the analysis and synthesis of protection system designs using the 'take and grant' model [Snyder 1977]. Ellis has given a mathematical definition of the degree of protection of a system [Ellis 1974].

For a survey on the problem of data security in the context of Data Base Management Systems, please see the section 'Data Security' in [Mohan 1977b].

Walter, using Multics as an example, has presented a technique called Structured Specification, in which successively more detailed models of security have been developed [Walter 1975a]. Conn has described techniques of modeling program control flow and has given the RISOS (Research In Secure Operating Systems) project as an application [Conn 1975]. Bell and La Padula have given a narrative description of the ESD/MITRE computer security model and its interpretation in the context of Multics [Bell 1976]. At SRI the Bell and La Padula, and Walter models have been generalized, resulting in two new models for multilevel security. The aim has been to prove properties of system specifications for multilevel security. An illustration of the proof of a system design has been presented in [Feiertag 1977].

4.2 Designs And Implementations

Earlier Andrews had developed a complete system (operating system independent), including data structures, primitive operations and a monitor, for enforcing security, based on the capability concept [Andrews 1974]. Popek has presented the design and implementation of the UCLA PDP 11 virtual machine system, which is an operating system kernel developed to provide high reliability protection and security [Popek 1975c]. Gat has worked on the concept of 'memoryless execution' to overcome the security problem. He has given the design of such a system and discussed its implementation considerations on a segmented microprogrammable computer [Gat 1976].

The HYDRA kernel [Cohen 1976c, Lamb 1976, Reid 1975, Wulf 1975] with an extended capability mechanism provides the most advanced protection features in systems implemented to-date. This system, with its abstracted notion of a resource, provides object level access and control protection and solves such protection problems like Mutual Suspicion, Confinement, Revocation, Conservation, and limiting Propagation of Capabilities. An excellent account of HYDRA's protection features can be found in [Cohen 1975]. Efforts are also being made to identify minimum mechanisms that are to be present in a security kernel for Multics so that protection enforcement could be verified [Adelman 1975 1976a b, Davis 1976, Schroeder 1975 1977]. The security kernel of an operating system to be used by the U.S. Air Force has been designed at the Case Western Reserve University. This system is to be used in a time-sharing environment with users of different security clearences [Walter 1975b]. The kernel for the Carnegie-Mellon University's multi-microprocessor (LSI 11's) system Cm*, is also being implemented with features similar to HYDRA [Fuller 1977]. MITRE has been designing a security kernel for the PDP 11/45 [Harper 1976].

4.3 Techniques

Newly implemented systems have begun to adopt the capability based protection mechanism as a means of attaining high security. The great utility of this mechanism has been surveyed in an excellent set of papers in the Computing Surveys special issue on Reliable Software [ACM 1976]. In the traditional capability schemes the capabilities of different objects could be combined together, in an uncontrolled fashion, to form new capabilities. This

facility could prevent the preservation of the least privilege norm and information flow requirements. To solve this problem the capability mechanism has been enhanced. Based on this enhancement a Generalized capability Vector Machine has been designed [Saal 1977]. The concept of conditional capabilities has been introduced [Ekanadham 1976a]. Unlike in the traditional capability scheme, in this approach the mere possession of a capability does not automatically enable a user to to do the operation associated with it. A state ('context') matching condition is also to be satistied. As far as the system is concerned the semantics of the conditions (which are user specified) are arbitrary. This scheme has been shown to support, in an elegant manner, type extensions and revocation of capabilities.

Informal methods for detecting protection vulnerabilities of existing operating systems are being developed at the Information Sciences Institute of the University of Southern California. The aim of this Protection Analysis project is to develop techniques and tools for detecting such anamolies statically by analysing program listings of the operating systems [USC/ISI 1975]. The 'types' of protection errors and 'error patterns' that commonly occur have been identified as a result of work in this project [Bisbey 1975, Carlstedt 1975 1976, Hollingworth 1976]. Hollingworth has described a search strategy for detecting 'residuals', that is, data or access capabilities left after the completion of a process and not intended for use outside the context of that process. Carlstedt, after giving a sample of 'validation errors' (validation is concerned with prevention or inhibition of certain operations - including operations at levels higher than the machine level - unless certain conditions are met) in existing systems, has outlined a scheme for finding them. Bisbey has described a procedure for finding errors characterized by the inconsistency of a data value between pairs of references [Bisbey 1975]. Some of the results of the RISOS project have been summarized in [Abbott 1976]. A set of security flaws have been identified and classified. Some enhancements to three commercially available operating systems have been suggested.

Another way to assure security is through cryptographic techniques [Matyas 1974, Neat 1975, Stahl 1974]. Matyas has presented the CRYPT procedure as a computer oriented cryptanalytic solution for multiple substitution enciphering systems. Stahl has presented a cipher based on a generalization of homophonic substitution. Neat has developed the Expanded Character Set (ECS) cipher as a new multi-substitution cipher that provides each user with the capability of controlling privacy protection for his programs and data. He has described the algorithms for ECS encipherment, decipherment, and description.

5.0 SYNCHRONIZATION AND DEADLOCK

Problems in these areas have been receiving very great attention during the recent past [Agerwala 1975, Andler 1977, Belpaire 1974 1975, Byrn 1975, Campbell 1976, Chang 1975, Cohen 1976, Griffiths 1975, Habermann 1975, Henderson 1976, Hicks 1976, Hoare 1975, Knott 1974, Lamport 1976a b, Lauer 1975a b, Lipton 1976a b, Lomet 1977, Reed 1977, Rivest 1976, Shrivastava 1975]. Much of the work has been necessitated by the advent of multiprogramming and distributed processing. A very good survey of the

concurrent programming research can be found in [Bryant 1977].

5.1 Models

Petri Nets have been found to be highly useful in modeling synchronization problems (a tutorial introduction to Petri Nets can be found in [Peterson 1977]). Their properties and capabilities have been thoroughly analysed. But the basic Petri Net alone has not been sufficient to represent certain problems like the 'reader/writer' problem. Chen and Coffman have suggested an extension ('token exhibitor arcs', which will inhibit the firing of a transition when an 'inhibitor input place' contains one or more tokens) to Petri Nets for modeling this problem [Chen 1975a]. Agerwala has proposed an extended Petri Net model for modeling concurrency and has given a wide variety of situations in computer systems that could be modeled. He has analysed the capabilities and limitations of ordinary Petri Nets and has established a hierarchy of Petri-like Nets, for the systematic analysis and comparison of a large number of existing models for parallel computations [Agerwala 1975]. Very recently the Coloured Petri Net model (permiting coloured 'tokens') has been introduced. Its modeling power has been analysed thoroughly [Zewos 1977]. Chen has described some two dimensional graph models for representing, with great ease, synchronization problems, for descriptive purposes [Chen 1975b].

Ariely has developed a model which unlike most other models of parallel systems (Petri Nets, etc.) considers time explicitely. He has also modeled multiple accessing of objects, dynamic process creation and deletion, process multiplexing and interrupts [Ariely 1975]. Cohen has proposed a graph model, called sequential marked graphs, for depicting flow of control through parallel processes [Cohen 1976]. Another model of computation is Hewitt's Actor system, in which all the basic steps of computation are carried out by sending messages amongst the different Actors [Hewitt 1977a b].

Belpaire has defined a framework for discussing problems of synchronization in both centralized and distributed systems [Belpaire 1975]. He has distinguished the different classes of synchronization problems and has defined formal rules to characterize them. He has given in detail a formal treatment of the deadlock problem for the class of problems of exclusions between critical sections [Belpaire 1974]. Miller has studied the theory of deadlock to determine how centralized resource management algorithms may best be adopted for distributed control and has proposed a resource management scheme along with the requisite synchronization methodologies [Miller 1974]. Chang has studied the cost of different deadlock avoidance algorithms and resource utilization under the different schemes [Chang 1975].

5.2 Synchronization Primitives

Efforts have been directed at analyzing and comparing existing and newly proposed synchronization primitives. Very recently Andler has given an excellent overview (with an extensive annotated bibliography) of the various primitives and the proof rules for many of them [Andler 1977]. Zamorski has

dealt with the selection criteria for choosing synchronization techniques to be used in an operating system [Zamorski 1975]. In order to help in making the choice, the behavioral characteristics of systems incorporating variations of the PV primitives have been given [Henderson 1976]. Byrn after developing theoretical results for Petri Nets has shown certain results which suggest that 'power' per se is not a particularly good criterion to use in comparing coordination primitives, unless there is agreement on what set of stylistic constraints are to be in force. He has introduced a new primitive called Valve which provides for limited interactions between processes without disclosing to one process what the other is doing [Byrn 1975]. Lamport has defined and given an algorithm for the implementation of a powerful synchronizing primitive which when used results in the continuous operation of a multiprocess system despite the failure of individual processes [Lamport 1976b]. Another coordination scheme based on a somewhat similar approach has also been reported [Rivest 1976]. Petterson presented a tutorial introduction to the then existing synchronization primitives and then presented a new form of primitive for expressing conditional critical regions [Petterson 1975].

Recently 'path expressions' have been proposed as a synchronization primitive [Campbell 1976]. Habermann, after defining and explaining the use of 'path expressions', has shown their advantage over the P and V operations on semaphores [Habermann 1975]. Extending this work, very recently Lauer and co-researchers have introduced the 'basic path notation' for describing concurrent programs. They have given rules for transforming such a program into its corresponding transition net in order to prove the 'adequacy' (protection of shared resources, absence of deadlock and observation of capacity bounds) of the program [Lauer 1977].

6.0 DISTRIBUTED SYSTEMS

A tremendous amount of interest is being shown towards distributed processing on multi-mini processor systems and distributed computer networks (packet-switched networks in particular). Multi-mini processor networks like C.mmp, Cm*, PRIME, DCS, SPIDER, MESH and PLURIBUS, and packet switched networks like ARPANET, ALOHANET, CYCLADES, DATAPAC, INFOSWITCH, ETHERNET and EPSS, have become highly popular. Many inter-network connections have been successfully implemented. All these have necessitated the development of operating systems and other software for making efficient and convenient sharing of the resources offered by them [Bedard 1977, Eckhouse 1976, Kimbleton 1976a b, Mills 1976a b, Retz 1975, Rowe 1975]. Many distributed operating system designs and implementations have been reported. Many schemes for interprocess communication have been proposed and some have been implemented [ACM 1975a, Akkoyunlu 1976, Greif 1975, Hicks 1976, Thomas 1976].

6.1 Designs

Bowie, using a graph model, has analysed OS/360 to detect the control flow and the interconnections between the modules of the operating system, so that parts of the system could be distributed over many processors for use in a multiprocessing system [Bowie 1974 1975]. Huber has discussed tradeoffs in

the design of interprocess communication (IPC) facilities for the network environment operating system, the Stony Brook System, which has no interrupts [Huber 1975a b]. Schantz has given the design of a hierarchical multiprogrammed operating system for a network environment. The structure of the system is insensitive to hardware and software peculiarities of any particular network configuration. He has also introduced a high level IPC facility called data ports [Schantz 1974]. Bernstein and Ekanadham have given the considerations that help in deciding the features to be provided in a network IPC facility [Bernstein 1975].

Reames has given the design of the DLOS operating system for the Distributed Loop Computer Network. DLOS makes the presence of the network transparent to the user and provides for IPC by process name, global program control, dynamically alterable, multi-layered process control structures, and distributed resource management and file system [Reames 1976]. Nutt has compared the effectiveness of dedicated uniprogrammed, dedicated multiprogrammed and distributed multiprogrammed operating systems for a hypothetical multiple-control-unit single-instruction-stream multiple-data-stream (MSIMD) computer [Nutt 1975a], for given job loads [Nutt 1975b]. Kimbleton and Mandell after identifying the objectives of a network operating system and determining the constraints in achieving these objectives, have presented an approach for network operating system development [Kimbleton 1976a b].

6.2 Implementations

Efforts have also been made to modify the existing uniprocessor operating systems for use on computer networks and multiprocessors [Burchfield 1975, Chesson 1975a b, Hawley 1975, Thomas 1974 1975].

A network version of UNIX has been implemented at the University of Illinois at Urbana for use on the ARPANET. It is a standard UNIX augmented with the Network Interface Program (NIP). It provides the user with a simple and clean interface and permits him to read from and write to foreign Hosts. Experimentations are in progress to increase the utility of Network UNIX for distributed data techniques and for connecting multiple UNIX systems together for distributed processing [Chesson 1975b]. A multiprocessor version of UNIX called MUNIX has been implemented at the Naval Postgraduate School [Hawley 1975]. An adaptive scheduling algorithm has been developed for this operating system [Joy 1975]. A partitioned-segmented memory manager has been implemented in UNIX [Emery 1976].

Much work is being done at BBN in extending TENEX for use on the ARPANET. A distributed executive-like system called RSEXEC [Thomas 1974] has been implemented for TENEX Host computers as an experiment to investigate the feasibility of the multi-Host TENEX (a distributed operating system) concept. It removes the logical distinction between local and remote resources and in many contexts permits references to files and devices to be made in a site independent manner. It supports a distributed, and optionally a 'multi-image', file system at the command level. It also supports inter-site user interaction functions and facilities for monitoring system load on all ARPANET TENEX's, remotely from one or more TENEX Hosts. RSEXEC has been made

available to TIP's also, providing them a 'logical front end'.

A new mechanism has been introduced in TENEX which permits one process to control the execution environment of other processes by providing them with a virtual machine that enlarges, restricts or completely redefines the 'standard' virtual machine provided by TENEX. The controlling process does this by intercepting the system calls made by the controlled processes [Thomas 1975]. This trap mechanism has been found to be highly useful in implementing RSEXEC and in establishing the National Software Works (NSW) environment. (NSW is an attempt to improve the production of large programs on the ARPANET by linking together the great variety of programming 'tools' now available on the network into a coherent system for software development employing a standard interface for users and a large secondary memory for storing and manipulating user files with a File Manager which will always be online monitoring the structure and content of users' files). Improvements to the security features of TENEX are also being carried out [Burchfield 1975, Thomas 1974].

At the University of Illinois at Urbana DEC's DOS operating system has been modified to implement 'MULTIBATCH', which provides an interprocess communication facility that permits a software system to be implemented and checked out using only one PDP 11 processor while production versions could be distributed among several processors [Chesson 1975a]. As the result of a similar effort Winspur has presented the design and a prototype implementation of the Pooled Computer System operating system which permits an application program to be run under a single or multicomputer version of the operating system without re-coding [Winspur 1975].

At the University of Rochester an operating system called ALEPH has been designed and implemented on Rochester's Intelligent Gateway (RIG), which connects users to many computers (IBM 360/65, PDP-10, and the multi-microprogrammable processors system , the 'Computer Engineering Research Facility' (CERF)) and computer networks (ARPANET and ETHERNET). ALEPH provides full or half duplex character transmission between terminals and any of the Gateway accessible computers, file transfer operations between any two systems or peripherals and process to process communication among systems. Interprocess communication is through messages. Each process could have many ports through which it could receive messages [Ball 1976]. Carnegie-Mellon's Cm* operating system provides sophisticated interprocess communication facilities and a virtual common memory that spans all the LSI-11 microprocessors in the system [Fuller 1977].

7.0 SCHEDULING

A good introduction to scheduling techniques in operating systems can be found in [Bunt 1976]. After introducing different techniques Bunt has compared the schedulers of OS/360, Multics and UNIX. A study of the applicability of recent operating system structuring concepts for the design of a scheduler, intended to serve more than a single category of job loads, has been carried out [Itzkowitz 1976]. An interesting approach to the scheduling problem has been proposed recently. The concept of a family of scheduling strategies in which a few parameters can be varied to achieve

different performance levels has been introduced [Mitrani 1977]. With the advances made in the firmware area attempts are being made to incorporate software features in firmware [Brown 1976, Richards 1975]. The microprogrammed implementation of a scheduler is described in [Chattergy 1976]. Recently a structured approach to the design of the processor multiplexing mechanism has been undertaken. The aim has been to disentangle virtual memory management from processor management, with a view to build a secure kernel for Multics [Reed 1976].

Much research work on deadline and multiprocessor scheduling and modeling has been reported.

7.1 Deadline Scheduling

Some of the problems and characteristics of deadline scheduling have been discussed in [Mok 1976]. Zwass has proposed a new data processing organization in which the priority jobs will have deadlines attached to them. If these jobs are accepted by the system then their processing should be completed within that deadline. In this type of system the priority based pricing will vary dynamically according to the backlog of priority jobs. These objectives are met through a component of the operating system called the pricing/scheduling system [Zwass 1975]. Coffman has developed a decision procedure for developing preemptive scheduling strategies in a system with known demand and prespecified waiting time requirements [Coffman 1975]. Chandy has studied the bounds on completion times and the applicability of optimal deterministic schedules to probabilistic models of execution times in a multi-tasking system [Chandy 1975].

7.2 Multiprocessor Scheduling

Many studies have been made with different preemptive and non-preemptive algorithms to determine their time complexities. Villanueva in investigating the execution time properties of multiprocessor scheduling has studied the stablity properties of schedules produced by non-preemptive algorithms and those produced by four important non-enumerative algorithms that produce an optimal schedule length provided certain conditions are met. He has also shown that in real life, where processor switching and preemption costs must be taken into account, one can no longer say that the preemptive scheduling discipline is strictly more powerful than the non-preemptive scheduling discipline. He has given certain guidelines for deciding, in a given situation, whether or not preemptive scheduling is better [Villanueva 1975]. Kafura has given the worst-case bounds on a variety of basic scheduling strategies for a model of a multiprocessor system containing an arbitrary number of identical, independent processors but with varying amounts of private memory. He has considered preemptive scheduling and strategies with a 'look-ahead' feature also [Kafura 1974]. Soh has developed some heuristic algorithms for preemptive scheduling of N periodic jobs, characterized by their frequency of occurrence and execution times, in a multiprocessing system, both in the presence and absence of dependencies among the jobs [Soh 1974].

Gonzalez and Sahni have presented an O(n) time algorithm for obtaining an optimal finish time preemptive schedule for N independent tasks and M processors of varying speed. This algorithm is better than that of [Horvath 1975] because of the difference in the worst case number of preemptions which is only 2(M-1) in [Gonzalez 1976] whereas in the other it is N(M-1). Ibarra and Kim have studied the worst case bound for the ratio of actual finish time to the optimal finish time and the run time of several heuristic non-preemptive algorithms for scheduling N independent tasks on M non-identical processors [Ibarra 1975]. Krause, Shen and Schwetman have studied the worst case performance of many non-preemptive scheduling algorithms for a system of many identical processors with arbitrary memory size. They have also given a new heuristic algorithm which uses a 'look-ahead' strategy [Krause 1975]. Goyal has studied the scheduling of partially ordered tasks on M non-identical processors with the additional constraint that a task can be scheduled only on the processor that the task specifies. He has shown that the problem of determining the optimal schedule for this 'M-processor-bound' system is NP-complete even in restricted cases [Goyal 1976]. Millen has presented, based on the theory of parallel program schemata, three alternatives for the design of a scheduler for a multi-mini processor system [Millen 1975].

7.3 Modeling

Ramamoorthy has proposed and tested the effectiveness, through simulation, of certain heuristics for scheduling parallel processable tasks on a uniprocessor [Ramamoorthy 1976]. Recently Jensen has constructed a combined general model (it consists of a program model, a hardware model, a contention model and a scheduling model) of a complete multiprocessing system. This model has been simulated for testing the effectiveness of different scheduling disciplines [Jensen 1976]. Reeves has developed and validated a multiple subsystem simulator of processor scheduling for the performance of a time-sharing system [Reeves 1975]. Agrawala has developed four models of memory scheduling and has examined their characteristics using simulation [Agrawala 1975]. Nehmer has algorithmically defined and illustrated the use of primitives to be used in the structured construction of kernels for dispatching processes [Nehmer 1975].

7.4 User Experiences

Gonzalez has completely analysed the operation and performance of the PDP-10 TENEX operating system scheduler. He has carried out several experiments to test hypotheses about performance improvements of the scheduler and has presented two theoretical models for the Roll-in and Roll-out of blocked jobs [Gonzalez 1975]. Due to its use in a system which had less primary memory than required by TENEX, its scheduler caused poor responses at the University of Utah. So a new scheduler, an unconventional one which considered more than one critical resource, was implemented at Utah. This change has resulted in shorter response times [Ellison 1975].

8.0 STORAGE MANAGEMENT

With the advent of virtual memory systems, development of new memory management techniques has assumed great importance [Janson 1977]. Many algorithms for the allocation of primary memory space and for page replacement have been developed. Measures have been developed for comparing the performance characteristics of these algorithms. In order to better understand the nature of the problems involved many program behavior models have been proposed [Chow 1977, Graham 1976].

8.1 Paging Problems

Burris has developed two algorithms, the Locality Matrix Model (LMM) and the Dynamic Cluster Model (DCM), for dynamically clustering the pages of a problem program, based on past program behavior, in a demand paged virtual memory environment. When these algorithms are used, during a page fault, not only the demanded page but also any associated clustered pages are brought into primary memory. Simulation results have indicated the superiority of this approach over most of the currently implemented algorithms [Burris 1976]. Jain has obtained an a priori model of program behavior, based upon localities, using information about program connectivity and then using this structural information has obtained feasible algorithms for the pagination, replacement and memory allocation problems in virtual memory computer systems [Jain 1975]. Gupta has analytically studied system performance under the paging algorithms Working-Set (WS), Page-Fault-Frequency (PFF) and LRU. Several models of program reference behavior have been developed and analysed including a new model, the Multi-Partition Locality Model [Gupta 1975]. Courtois and Vantilborgh have developed a model of program behavior in which distinction is made between short and long run equilibrium states in nearly completely decomposable, that is highly modular, systems [Courtois 1976]. After reviewing the other important models they have argued the superiority of their model.

Peter Denning has shown the near optimality of load controls, based on his Working Set memory management, using recent advances in program behavior and system behavior models [Denning 1976a b c d, Graham 1976, Kahn 1976]. Sadeh has developed a mathematical model and using the properties of imbedded Markov Chains has analysed the performance of the PFF algorithm [Sadeh 1975]. Recently Hamilton has proposed a Markov model for studying multi-level paging hierarchies. He has derived certain equations for analytically determining the load at which the hierarchies become overloaded. The model has been validated by real life measurements [Hamilton 1976]. Methods for computing the fault rates (for multi-level hierarchies under the Independent Reference Model of program behavior) for the replacement policies A0 and LRU have been presented in [Mortenson 1976]. A multiprocess design of a paging system has been presented in [Huber 1976].

Ferrari has advocated that a program's behavior could be tailored so that it adheres to the model underlying the storage management technique in use [Ferrari 1975 1976]. He has given an algorithm for performing such restructuring for the WS model. Other researchers have also been devoting attention to the study of this problem [Archard 1977].

In contrast to previous studies recently the 'write' (modifying the contents of a page) behavior of programs has been analysed to find out the effect of page write costs (the cost of copying onto auxiliary storage, a modified page, at page replacement time) on the page fault rate [Yu 1976].

8.2 Primary Memory

Weinstock in studying the dynamic allocation of memory has developed a model for identifying the properties required of a dynamic allocation method and has used it for describing the techniques in current use. He has developed a new method called Quick Fit, which has been shown to be the fastest in terms of speed of allocation. He has also given a means of choosing a method for a balance of good storage utilization and speed [Weinstock 1976]. Ting has shown that all allocation algorithms require O(M log M) words of memory to preclude the possibility of overflow, where M is the number of busy words at any one time. He has defined a cost for evaluating the performance of memory compaction algorithms, and has defined new algorithms and discussed their implementation [Ting 1975].

Clifton has compared the effectiveness of a number of methods for dynamically allocating storage so that memory utilization, without doing compaction, remains high. A simulation model and results of simulations have been given for the Paged and Partitioned allocation methods, and Randell's and two variants of Randell's algorithm [Clifton 1975]. Nair has outlined a simulation methodology employed for analysing these algorithms [Nair 1976a b].

8.3 Comparison Measures

Sneeringer has developed comparison models of several memory management techniques for time-sharing systems and a cost-per-user model to illustrate the saturation points associated with the different systems and to provide a means of choosing the most economical system which will support a given number of users [Sneeringer 1975]. Kubo has defined a system wide measure called 'Run-Cost' which reflects cost/performance of actual computer systems, for the evaluation of virtual memory systems to determine the optimal page size [Kubo 1975]. Kain has given methods to partially order the performance of paging algorithms [Kain 1975]. Rao has compared the performance of different cache buffer organizations like Direct Mapping, Set Associative, Fully Associative, and Sector buffer, in a two level demand paged memory system under replacement policies like FIFO, Random Replacement and the LRU scheme [Rao 1976]. Yu has proposed a total page transportation cost, which takes into account the cost of page writes also [Yu 1976].

Recently Saltzer has questioned the practical utility of the theoretical results that have come out of modeling of paging algorithms [Saltzer 1976]. Peter Denning has attempted to answer him by pointing out some of the good effects of attempts to model memory management [Denning 1976f].

The TENEX pager module has been thoroughly analysed at the Case Western Reserve University [Radelja 1976]. Some deficiencies were found in the TENEX WS algorithm, in a shared page environment. Some modifications have been implemented. Yang has shown that a MIMD pipelined processor is more susceptible to page faults and hence the performance of the system initially depends on the size of the primary memory [Yang 1975].

9.0 MONITORS

Brinch Hansen's Monitor concept has become highly popular and has been receiving much attention from researchers. Some recent operating systems have the monitor concept embedded in them: SOLO [Brinch Hansen 1976] and SUE/11 [Greenblatt 1976] operating systems designed and implemented at California Institute of Technology and University of Toronto, respectively.

Research results on proving of monitors [Howard 1976a, Kieburtz 1977a, Saxena 1976b] and on their implementation [Bochman 1976, Brinch Hansen 1976 1977, Howard 1976b, Karp 1976, Lister 1976a b 1977, Schmid 1976], have been reported recently. Howard has axiomatized the 'wait' and 'signal' operations on monitors and has proposed the use of 'history' variables for proving monitors using Hoare's methodology. He has illustrated his concepts with many examples [Howard 1976a]. A methodology for building correctly functioning asynchronous systems has been described in [Silberschatz 1976a]. These results have been extended in [Akkoyunlu 1977] for the case where synchronization primitives could be embedded in monitors.

Brinch Hansen's Concurrent PASCAL implements monitors [Brinch Hansen 1975]. Lister has discussed the problems in the use and the construction of monitors in practice and has described the implementation of monitors in BCPL [Lister 1976a b 1977]. Schmid has given methods for identifying invariants in systems using conditional critical regions. These invariants aid in the efficient implementation of the latter. A monitor is obtained by collecting all conditional critical regions that are associated with the same shared variable into one program module. The advantages of this approach to monitor construction have been given [Schmid 1976].

Shrivastava has illustrated the usefulness of the monitor concept by using it in structuring scheduling algorithms and solving some fairly difficult scheduling problems [Shrivastava 1976]. Sufficient conditions for verifying the absence of deadlock in a system of monitors have been presented in [Saxena 1976a].

Recently the monitor concept has been extended (by the introduction of a new program component called 'Capability Manager') to provide dynamic access control [Kieburtz 1977b]. Owicki has generalized the Monitor concept by defining Shared Classes, which remove the restriction of permitting only mutually exclusive calls to the procedures of a Monitor [Owicki 1977].

10.0 COMMAND OR JOB CONTROL LANGUAGES

For a long time users were not very happy with the types of command languages that they had to put up with. The earlier designers had not given much thought to the human factors problems while designing the job control languages. But during the last two to three years an earnest attempt has been made to change the situation [Brunt 1976, Cox 1975, Gray 1975, Little 1975, Parsons 1975, Sayani 1976, Unger 1975]. Heafner has stated that in order to develop man-computer languages, taking into consideration the users' needs and habits, and the features of the computer service, and requiring optimal performance at the same time, one has to involve potential users in the language design process. he has described a protocol analysis method for achieving such objectives [Heafner 1975].

10.1 Designs

Tomlinson has designed an ALGOL 60 based high level language as a replacement for existing command languages. He has illustrated the effect of the primitives of this language on a hypothetical operating system and has compared the capabilities of this language with those of the existing ones. He has also discussed its implementation on top of OS/360 [Tomlinson 1976]. Abbot has given the functional specification for a computer based, interactive Command Language Processor, which operating on a dialogue grammar provides a very flexible user interface to a sophisticated text preparation and message processing service [Abbot 1974]. Research is also being carried out for the formalization and description of command languages, that would facilitate proving of command programs [Neuhold 1975 1976], just like proving ordinary programs written in programming languages [ACM 1975c].

Dakin has developed the structure of a common interface language for accessing the facilities of a variety of operating systems [Dakin 1975]. The progress achieved by a British Computer Society working group in its attempt to design a machine independent command language has been discussed in [BCS Group 5 1975, Newman 1976]. Newman has also discussed the design of a uniform command language for use on a network of dissimilar processors [Newman 1975].

10.2 Implementations

The MU5 [Frank 1976] and SOLO [Brinch Hansen 1976] operating systems present a very flexible user interface and permit high level languages, ALGOL and PASCAL respectively, to be used as command languages. An outline of three high level portable command languages, UNIQUE, GCL and ABLE, has been given in [Rayner 1975].

11.0 HIGH LEVEL LANGUAGES

Another trend is the increasing use of high level languages for implementing operating systems [Lipton 1976c, Schwartz 1975, Shaw 1976, Wetherell 1975]. Lipton has summarized the research work that he has performed on this topic.

11.1 Designs

Somogyi has given the design of a complete software system, for a minicomputer, to be implemented using Koster's Compiler Description Language [Somogyi 1975]. Vervoot has described the design of a time-sharing system using Concurrent PASCAL [Vervoot 1975]. Markstein has given the design of three operating systems: a uniprogrammed system, a multiprogrammed system and a time-sharing system, using PSETL [Markstein 1975]. PSETL is a version of Schwartz's set-theoretical language SETL, which has been designed to allow the description of algoritnms involving interrupts, parallelism, and to some extent, machine dependent features.

11.2 Implementations

A high level language called PLATON has been implemented on an RC 3500 minicomputer. PLATON contains facilities for hierarchical control of a dynamic system of processes by means of queued semaphores [Staunstrup 1975]. Doi has given the implementation of the Structured Operating System (SOS), which is to be used for teaching purposes [Doi 1975]. He has shown how ALGOL or FORTRAN could be used for realizing cooperating processes. An extension of PASCAL, called SIMONE, has been implemented at the Queen's University of Belfast for describing, developing, testing and simulating operating system algorithms [Kaubisch 1976]. This system, which facilitates quasi parallel programming, has been designed primarily for use in operating system courses. It is being extended. BCPL has been used in realizing an operating system for the MODULAR ONE computer [Szczep 1975]. The operating systems ALEPH [Ball 1976], Chi/OS [Lynch 1975], HYDRA [Wulf 1975] and Cm* Kernel [Fuller 1977], SOLO [Brinch Hansen 1976 1977], SUE/360 [Sevick 1972], SUE/11 [MacEwen 1976] and UNIX [Ritchie 1974] have been implemented using the high level languages BCPL, Chili, BLISS, Sequential and Concurrent PASCAL, SUE/360, SUE/11 and C, respectively. Recently UNIX has been implemented on a LSI-11 also using the C language [Lycklama 1977].

To permit dynamic resource management recently Concurrent PASCAL has been extended by the addition of a new program component called a 'manager' [Silberschatz 1976b].

12.0 CONCLUSION

While much work has been and is being done in the different fields of the operating systems area, an effort was made (by getting together the leading computer scientists in this area) to determine the new directions to be taken by research in this area. The results of this have been summarized in [Browne 1977]. Some more ideas can be found in [Denning 1977].

13.0 ACKNOWLEDGEMENT

I would like to convey my thanks to Jim Peterson for his help.

14.0 REFERENCES

1. Abbott, R. P., et. al. (1976) Security Analysis and Enhancements of Computer Operating Systems, Lawrence Livermore Laboratory TR NBSIR-76-1041, April.

2. ACM (1975a) Proc. of the ACM SIGCOMM/SIGOPS Interprocess Communication Workshop, March (Operating Systems Review, July).

3. ACM (1975b) Proc. of the International Conference on Reliable Software, April (Also in Special Issue on Reliable Software- IEEE Trans. on Software Eng., June).

4. ACM (1975c) Proc. of the International Symposium on Proving and Improving Programs, July.

5. ACM (1976) Special Issue - Reliable Software II - : Fault-Tolerant Software, Computing Surveys, December.

6. ACM (1977) Proc. of the Conference on Language Design for Reliable Software, SIGPLAN Notices, March.

7. Adelman, N. (1975) Effects of Producing a Multics Security Kernel, ESD-TR-76-130, Honeywell Inf. Sys. Inc., October.

8. Adelman, N., Gilson, J. R., Sestek, R. J., Ziller, R. J. (1976a) Security Kernel Evaluation for Multics and Secure Multics Design, Development and Certification, ESD-TR-76-298, August.

9. Adelman, N. (1976b) Engineering Investigations in Support of Multics Security Kernel Development, ESD-TR-77-17, October.

10. Agerwala, T. K. M. (1975) Towards a Theory for the Analysis and Synthesis of Systems Exhibiting Concurrency, Ph.D. Thesis, Johns Hopkins University.