CHAPTER 1

INTRODUCTION

R. Bayer, R.M. Graham, J.H. Saltzer, G. Seegmüller

This book contains the lecture notes of an Advanced Course on Operating Systems held at the Technical University Munich in 1977 and 1978. The material of the course was discussed and organized during a preparatory seminar attended by all lecturers in early 1977.

An attempt was made to agree upon a uniform approach to the field of Operating Systems. The course differs from the usual approaches in its emphasis and selection of topics. We presume that the reader has had the experience of a more traditional operating systems course and that he has worked with some real operating systems also. The set of topics of this course is not the traditional set. It is strongly influenced by two considerations. The first observation is the beginning of a dramatic change in tradeoffs in view of decreasing hardware costs. The second one has to do with recently emerging new results in computer science which reflect a better understanding of several areas closely related to operating systems. So we are not going to present much on programs, processes, scheduling, resource control blocks, building of file systems and performance modelling. Rather an attempt will be made at a more intensive treatment of areas like protection, correctness, reliability, networks and decentralization.

What is an operating system? Although there are many terms used for the versions of existing operating systems, and no universally accepted definition, there is certainly agreement that operating systems are essential parts of at least the following three conceptual kinds of computing systems.

Programming systems consisting of

editors, compilers, debuggers,
the operating system,
the hardware.

Data base systems consisting of

data base managers,
the operating system,
the hardware.

Application systems consisting of

application programs,
the operating system,
the hardware.

There is also agreement on those aspects that are at the heart of operating systems. In fact, the terms nucleus or kernel are often used for the most essential functions of an operating system. Much of the research and development in operating systems has focused on resource management and the user's interface to this management. Our view of operating systems and the focus of this course is resource management in a very wide sense and the attendant user interface. We shall concentrate on the semantics of this interface, on internal system structure and, to some extent, on hardware architecture.

It is interesting and instructive to look briefly at the history of modern computer systems. In the beginning, computers were small, simple, and free standing. Each individual could use the machine on a one-to-one basis. Generally, there has been an evolution from this state to the current large, complex, multiprogramming, multiprocessor, central systems with virtual memory and many ancillary devices and subsystems. The major trends have been: from one user to many users of the same system; from isolated users to cooperating users; from sequential batch to multiprogramming, to time sharing; and, in both hardware and software, an increase in the degree of concurrency. Most importantly, we see a trend toward increased concern with the management of non-physical resources.

The first computer users always had the entire computer all to themselves for some interval of time. A user always had all the resources. Any resource management facilities provided by an

operating (or programming) system were entirely for the user's convenience. As the user community grew it was necessary to insure efficient, equitable distribution of the system's physical resources among all the contenders. It has become clear that any kind of sharing, even sharing between the operating system and a single user, requires resource management for the shared resources.

Even in a sequential batch system, a user had to be prevented from monopolizing the computer. Thus, system management of the central processor was required, at least to the extent of limiting the execution time of user programs. Memory was another resource that was managed quite early. The operating system itself required some primary memory. The programs and data of other users in the batch had to be protected from destruction by the user program currently executing. This was especially true as soon as direct access secondary memory was available in sufficient quantity to make permanent data storage feasible. Hence, system management of I/0 devices and secondary memory were required.

As the hardware became more complex, the management of these physical resources became more comprehensive and complex. Multiprogramming and time sharing had a substantial impact on resource management. Management of the processor evolved from simply enforcing the maximum execution time for a user's program to multiplexing the central processor(s) among a number of different user programs. Primary memory management evolved from a simple division between the system and a single user to virtual memories, which facilitate simultaneous sharing of primary memory among many users and the treatment of secondary memory as a direct extension of primary memory.

It is a principle of science that as complexity increases, the need for abstractions to deal with this complexity also increases. The evolution of operating systems is no exception. Early abstractions were files and processes. In each instance the abstraction takes the form of some non-physical resource and benefits both the user and the system. The abstraction of a file gives the user a unit of information that is extremely useful in organizing his data. Complex movement and manipulation of large amounts of data can be expressed very simply by the user in a device/location independent way.
At the same time, because of the abstract nature of files, system management of these resources translates easily into the management of physical secondary storage and I/O devices. In

addition, since the user does not specify details, the system has much greater latitude in physical memory management and more potential for efficient utilization of it.

In like manner, the abstraction of a process permits more efficient systems management of the central processor(s) as well as indirectly contributing to the ease of management of all other resources. The user also benefits from the process abstraction. With it he can establish sets of cooperating concurrent processes which not only take maximum advantage of the system's parallelism, but often result in clearer formulation of the problem to be solved. The notion of an abstract machine which is available to each user encompasses the essence of this direction of abstraction.

What is the current state of affairs? In a recent workshop the lecturers of this course concluded that the classic problems of physical resource management and concurrency management are well understood, at least to the extent that their implementation is routine and minor enough that operating systems that are satisfactory to the market place are being built. We have chosen to omit from this course any consideration of these problems. Acceptable solutions are widely known. In fact, all of the recent textbooks on operating systems contain extensive discussions of these problems and their solutions. Rather we tried to focus on problems that were less well understood in the past - that are on or near the frontier of the field and that showed significant progress within the last few years. For example, none of the textbooks has an adequate discussion of protection, yet this is one of the most important problems in the design of new operating systems.

Abstractions are based on models. We recognize that models are not only needed to cope with complexity, but ultimately they are needed to verify or validate the correctness and other desired properties of a specific system design. Models for the underlying hardware are the foundation upon which more abstract, general models are built, since they give us insight into the fundamental mechanisms for the final interpretation of a program that is required to produce actual results. In addition, through them we can glimpse a future kind of architecture with many parallel activities, highly distributed.

The object model is the basis for the abstract resource, an object. This very general model is applicable to both software and hardware. It has benefitted from more recent developments in

the study of programming languages. This benefit is not incidental. There, the need for careful specification of interfaces with total protection of their implementation has led to the introduction of abstract data types. Objects in operating systems correspond to data types as they appear in some more recent programming languages. The object model seems, in some sense, to capture fundamental properties that pervade all aspects of modern operating systems : protection, naming, binding, data, procedures, and physical devices. A model of this nature seems to be necessary in order to realistically consider the validation of important properties of an operating system, such as correctness and reliability.

There are a substantial number of major problems that affect the entire fiber of the more advanced operating systems. Most of these problems appear in the newer system organizations, such as, data base operating systems, distributed systems, and networks of computers. In these new settings the problems tend to be an order of magnitude more difficult. Naming and binding are fundamental. Resources cannot be managed without the use of names. The value of symbolic names was recognized long ago. Symbolic names need to be bound to specific objects. The complexity of this problem, when conjoined with protection and multiple computers networked together, is staggering. Protection, difficult enough in multiuser, timesharing systems, is far more complex when the access controls must extend throughout a network with a distributed data base. An important property of networks and distributed systems is that distinct components
are often under different administrative controls, thereby adding new problems
of coordination, protection, naming, and reliability.

The importance and need for correctness and reliability of operating systems has always been recognized. However, sensitive applications are currently being implemented within unreliable systems. Correctness and reliability issues are not unique to operating systems, but they are much more significant in this context. An undiscovered, minor bug in the system or a breach of the protection mechanism can result in great financial loss or even the loss of human lives.

What about the future? New hardware developments always influence the organization and function of new operating systems. Advances in communications technology have made networks of computers possible. New production and

miniaturization techniques make it possible to mass produce cheap processors. Distributed systems and highly parallel machines are inevitable. What are the advantages and disadvantages of such systems? What is the appropriate user interface? Current models are inadequate to deal with questions of correctness and reliability- nor are they of much help in guiding the designer to a simple and efficient implementation. Many of the readers will be deeply involved in these problems. In the lectures that follow, we hope that we will be able to help the reader prepare to cope with these problems.