Principles of Operating Systems

Sacha Krakowiak

translated by
David Beeson

The MIT Press
Cambridge, Massachusetts, London, England

Original edition © 1987 by Bordas, Paris, published under the title *Principes des systèmes d'exploitation des ordinateurs.*

This book was typeset in Oxford by Cotswold Press Ltd. and printed and bound in the United States of America.

Library of Congress Cataloging-in-Publication Data

Krakowiak, Sacha.
Principles of operating systems.

Translation of: Principes des systèmes d'exploitation des ordinateurs.
Bibliography: p.
Includes index.
1. Operating systems (Computers) I. Title.
QA76.76.063K7313 1988 005.4'3 87-5676
ISBN 0-262-11122-5

Principles of Operating Systems

Preface

The operating system in a computer, or a computer installation, is a set of programs carrying out two main functions:

- managing an installation's resources, ensuring, if necessary, that they are shared between several users and controlling their distribution over the many tasks submitted to an installation, and
- providing a series of services by presenting users with an interface that is more adapted to their requirements than that of the physical machine; this interface is that of a 'virtual machine' made up of a set of functions to manage and communicate information and to implement the application software.

Recent developments in computer science have been marked by the emergence of microcomputers and telematics services, alongside distributed computer systems and individual workstations, and the integration of computers both into complex industrial processes and into equipment for the mass market. Although these developments have led to changes in the form and function of computer systems, their design is still based on a set of principles that have been gradually elaborated over the years, supported by theoretical research and validated by their application to the construction of actual systems.

This book has two aims:

- to describe the main principles underlying computer operating systems: management of parallelism and synchronization; naming, storage, and protection of information; and resource allocation; and
- to illustrate the application of these principles to the actual design of systems, taking account of recent developments in computer technology and its fields of application.

To meet the latter objective, several sections (3.3, 5.3, 10.1, and 10.2) will deal with specific examples, for which simplified systems based on existing implementations have been drawn on. A faithful and complete description of a real system would have overloaded this book with details that would not have contributed to an understanding of the principles underlying system design. Nonetheless it is hoped that the discussion of simplified examples will assist in the detailed study of real systems by providing guidelines to their structure and highlighting their essential aspects. It is strongly recommended that such a study be carried out in order to supplement the lessons given in this book, particularly by teachers who draw on it for course material. Bibliographic references to guide a study of this kind are given at the end of chapter 1.

The field covered is essentially that of systems using a common memory, so-called centralized systems, whose design principles are now fairly stable. A

letter dated 21 May 1987 from M. Vitry, Foreign Rights, Bordas, states, "We have just put in production the second French edition which contains minor corrections and the additional chapter 11 ... It's due to be published early in October 1987."

This book contains material corresponding to preparation for a master's degree or the equivalent. It is principally directed at computer scientists who want to improve their knowledge in the operating system field.

It shall be assumed that readers are familiar with

- the general principles of computer architecture and
- the principles underlying the design of algorithms and programs; a programming language, including its practical use; and the general functions and organization of a compiler.

It is always difficult to select a language in which to present algorithms when discussing operating systems. As well as the difficulty of giving an overall view of complex programs, there is that of dealing with problems of hardware. Since this book is concerned with illustrating the methods used, rather than the details of their implementation, a Pascal-based notation that is quite common with any local extensions felt to be necessary has been chosen.

The end of each chapter gives an annotated list of bibliographic references and a set of exercises. Some of these exercises represent a direct application of concepts discussed; others require a certain amount of design work and could form the basis for a project; others are intended to encourage students to consider questions that have not been discussed and include references to guide further study in these areas.

Acknowledgements

The general approach in this book was heavily influenced by the work published by the Crocus[1] group: *Systèmes d'exploitation des ordinateurs* (*Computer Operating Systems*), published by Dunod in 1975. The task is easier now, as the essential concepts that at the time were still being elaborated or were changing rapidly are now more stable. But without the clarification and survey work carried out by that group, this book would never have been written; my cordial recognition is therefore due to my colleagues in the Crocus group.

Although in this case it was more indirect, another influence on this book was that exerted by the work since 1976 of the Cornafion group. The presentation given here of the questions of synchronization and object management, in particular, owes a great deal to the Rennes sub-group: Françoise André, Daniel Herman, and Jean-Pierre Verjus.

This book emerged from courses taught since 1973 in various departments of the Université Scientifique et Médicale and the Institut National Polytechnique, both in Grenoble, France. Many people, both teachers and students, have contributed over the years to the development of these courses. The way in which concepts are presented here, as well as the preparation of example applications and exercises, draws heavily on long-standing work in collaboration with Jacques Briat, Joëlle Coutaz, Guy Mazaré, Jacques Mossière, and Xavier Rousset de Pina. I naturally remain responsible for any errors there may be (which I invite readers to point out to me, for correction in any subsequent edition).

The original edition of this book was published with the support of the French Ministry of Education, Industry and Research, as part of the programme to assist the publication of scientific and technical books in French.

(1) Crocus: a collective name for J. Bellino, C. Bétourné, J. Briat, B. Canet, E. Cleeman, J.-C. Derniame, J. Ferrié, C. Kaiser, S. Krakowiak, J. Mossière and J.-P. Verjus.

Contents

Chapter 1
Introduction

Programs executed on a computer are generally classified, according to their functions, under two headings: *system software* and *application software*. The operating system is an important component of system software. The aim of this introduction is to explain precisely what is meant by these terms, to define the main functions of operating systems with a few examples as illustrations, and finally to describe their common characteristics.

1.1 Functions of an operating system

A computer system is a combination of hardware and software intended to carry out tasks involving the automatic processing of information. Such a system is linked to the outside world by input-output mechanisms, allowing it to communicate with human users or to interact with the physical devices it is intended to control or to supervise.

The function of a computer system is to provide services suitable for the resolution of common problems:

- Information management: storage, naming, retrieval, communication
- Preparation and debugging of programs
- Execution of programs

It is convenient to regard the services provided by a computer system as defining, for the user, a new machine that is often called *abstract* or *virtual* as opposed to the physical machine formed by the components of the hardware. The description and operating instructions for these services form the **interface** of the computer system. We can therefore say that this interface defines a language, which is that of the abstract machine, and which allows users to communicate with the system. All the information necessary for the proper use of the system by a user (or by a connected physical device) is contained in the interface.

The level of abstraction of the means of expression provided in this way tends to increase with technical advances, i.e. the objects and operations considered as elementary on the abstract machine are implemented by more and more complex sets of objects and operations on the physical machine. Furthermore, different users of a single computer system may require different abstract machines. To simplify the design of the system, it is convenient to decompose the task to be carried out: in very broad terms, functions common to many applications are implemented by a set of programs called

system software; application software, on the other hand, implements a specific application by calling on the services provided by system software. Figure 1.1 shows this form of organization by means of a hierarchical schema where each *layer* uses the resources provided by the layer immediately below and presents the layer above with an interface describing the resources it offers; the interface of the highest layer is that of the whole computer system. The concepts associated with the decomposition of systems are discussed in detail in chapter 3.

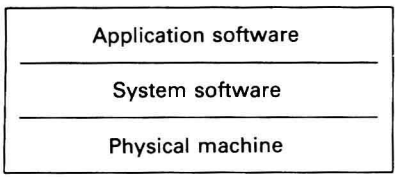

Fig. 1.1. Structure of a computer system

This structure should not be interpreted too strictly; in particular, the boundaries between layers are often imprecise and moving: a program initially developed as an application program may be built into system software if it turns out to be a commonly used tool for a class of users; in the same way, system software functions initially implemented by a program may be implemented by a microprogram or by an electronic circuit if savings or improved performance justify this step. Within system software itself, it is common to distinguish two levels (fig. 1.2).

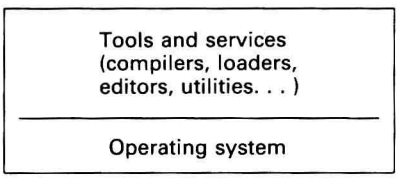

Fig. 1.2. Components of system software

Here again, the boundary cannot be strictly defined. The main functions of the lower layer of the system software, the **operating system**, can be classified under two main headings.

1. Definition and implementation of a virtual machine

a. Information management: structuring, storage, naming (virtual memory, files); transfer (input-output).
b. Execution control: execution of programs in sequence, in parallel, program composition, etc.

c. Miscellaneous services: assistance to debugging, fault handling, time measurement, etc.

2. *Resource management and sharing*

a. Management of physical resources: allocation of main memory, of secondary memory, of input/output devices
b. Information sharing and exchange between users
c. Mutual protection of users
d. Miscellaneous services: resource usage accounting, usage statistics, performance measurement, etc.

Decomposition into layers as a structuring tool occurs again within operating systems.

1.2 Examples of operating systems

The four examples below illustrate the variety of functions that an operating system must carry out. These examples are characteristic of several major classes of systems, but are not intended to be an exhaustive list. For each example, we indicate the functions to be carried out and the major characteristics required. Common characteristics of operating systems are summarized in section 1.4.1.

1.2.1 Personal computers

A personal computer, in its simplest configuration, is made up of a central processing unit, main memory and a terminal (screen and keyboard, possibly a mouse). This configuration is generally supplemented by secondary memory (floppy disk) and a printer (fig. 1.3).

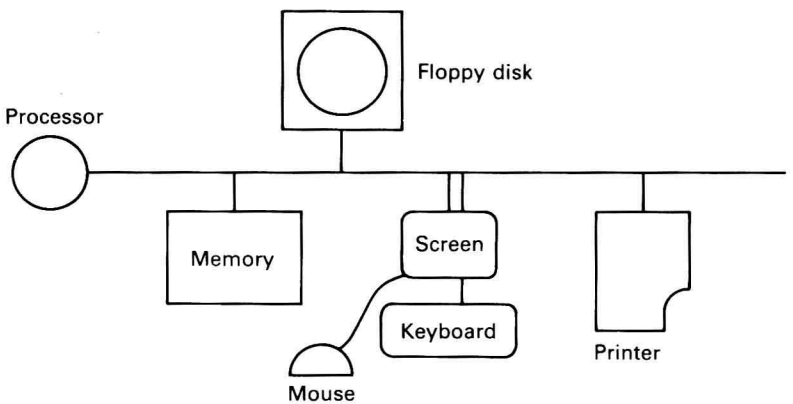

Fig. 1.3. Personal computer

Users of such systems essentially require two types of service:

- Creating and naming files, or structured sets of information; storage of these files in secondary memory; transfer of information between files and input/output devices (screen, keyboard, printer).
- Execution of programs which may be part of the system or be introduced in the form of files; data are entered through the keyboard or provided in a file; the results are displayed on the screen, listed on a printer, or copied to a file.

The interface presented by the system to a user is a **command language**. A command is a construction of the form

$$< \textit{action} > <\textit{parameters} >$$

input via the keyboard, or using a mouse, and immediately interpreted by the system.

Here are two examples of typical sequences of actions:

1. Debugging programs

- Write a program at the keyboard (using a text editor)
- Execute of the program providing data through the keyboard
- Modify of the program if results are not satisfactory, and reexecute it
- Store the final version of the program

2. Operation

- Request the execution of a pre-existing program, or a program written as above, using a set of data contained in a file or input on demand to the keyboard. Results are displayed on the screen, listed on a printer or copied to a file to be reused.

In such a system, there is no resource sharing, since the machine is being used by a single user who has total control over it. Resource allocation occurs for memory management and file loading. The main functions of the operating system are file management, implementation of input/output and interpretation of the command language.

For this type of system, the most important qualities are:

- Reliability
- Efficiency (as the performance of the hardware is often low, it is important to make the best possible use of it)
- Simplicity in use
- Ease of extension by adding new utility programs, or adapting to new peripherals

The latter two aspects highlight the importance of a careful design of the

interfaces provided by the system, both for the command language and the file management system.

The organization of a simple operating system for a personal computer is discussed in chapter 3.

1.2.2 Control of industrial processes

In a chemical products factory, a reactor is used to synthesize product C from two products A and B (fig. 1.4).

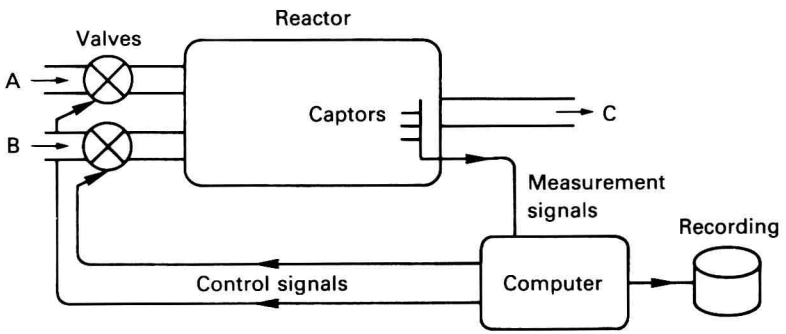

Fig. 1.4. Control of a chemical reactor

This manufacturing process is controlled by a computer, which carries out three functions.

- *Regulation*. For the process to take place correctly, operating parameters (temperature, pressure, concentration, etc.) must be maintained within fixed limits. This means regulating the input rate of products. Operating parameters are measured using sensors; the computer collects these measurements and acts in consequence on the input valves, as specified in a regulation program.
- *Recording*. The various results obtained by measurements are recorded periodically, their values displayed on a control panel and copied into a file (or 'log') with a view to additional processing later (operating statistics).
- *Security*. If certain measured parameters exceed a predefined critical value (as would be the case in an accident), emergency shutdown of the reactor must take place.

Let us now examine the constraints introduced by this mode of operations.

1. Measurements are taken periodically; let T be the sampling period and let t be the total time necessary for the computer to process a set of measure-

ments (collection, recording, determination and execution of valve commands).

The system can only operate if

$$t \leqslant T$$

2. The security function must be given priority over all others; in other words, it must be possible at any moment to detect that a critical value has been exceeded and procedures for handling such an occurrence must interrupt operations that are under way.

The main functions of the operating system are the following.

- To act on external devices (reading of sensors, control of valves)
- To schedule events in physical time (periodic triggering of the processing cycle)
- To react to external events (emergency shutdown)
- To manage information (storage and maintenance of the log file)

The specification of a physical limit to the duration of a computer process, the existence of scheduling periods, the concept of priority handling and the connection to equipment for the control or measurement of an external device are characteristic of computer processes known as **real time** applications.

As well as process control, examples of the use of these systems are: the management of telephone switchboards; automatic pilots for aircraft, missiles or satellites; medical monitoring, robot control; etc.

For these systems, the major requirement is reliability. As the consequences of a failure may be catastrophic, the system must be able to guarantee the safety of the device it controls at any time: it must therefore be able to guarantee a minimal service even in the case of a hardware breakdown, an accident or a human error.

An example of a system kernel suitable for programming real time systems is described in chapter 5.

1.2.3 Transaction systems

Transaction systems are characterized by the following properties.

1. The system manages a set of information, or database, which may be very large (billions of bytes).

2. A certain number of predefined operations, or transactions, which are often interactive, can be applied to this information.

3. The system has a large number of access points (terminals) and many transactions can take place simultaneously.

Systems used to handle rail or air seat reservation, to manage bank accounts or to retrieve documents, are all examples of transaction systems.

Information contained in the system database is subject to integrity constraints expressing its internal consistency. These constraints obviously depend on the application: e.g. the number of seats reserved in a train may not exceed the number of seats available, the same seat cannot be assigned to more than one person, etc. Execution of a transaction must maintain the consistency of the information.

The qualities required of a transaction system are availability and reliability; for certain systems, fault-tolerance is also essential. Important characteristics of transaction systems are that they cater to many parallel activities, and, in many cases, components are widely spread geographically, which leads to specific problems (see section 1.3.3).

1.2.4 Time-sharing systems

The function of a time-sharing system is to provide services to a group of users; each of them may use:

- Services equivalent to those available on a personal computer
- Services associated with the existence of a community of users: sharing of information, communication between users

Moreover, thanks to the sharing of costs between a large number of users, it is possible to offer each of them common services that would not be available individually, such as access to special peripherals (graphics output, etc.) or to complex software requiring large amounts of memory.

The problems involved in designing time-sharing systems therefore combine those of personal computers and those of transaction systems. They can be classified as follows:

- Definition of the virtual machine offered to each user
- Sharing and allocation of common physical resources: processors, memory, input output devices
- Management of shared information (files) and of communication

The qualities required of a time-sharing system also cover those of a personal machine and of a transaction system: availability, reliability, security; efficient use of the hardware; quality of the interface and of the services provided to users; ease of extension and adaptation.

1.3 Historical development of operating systems

A rapid description of the way operating systems have evolved will give a