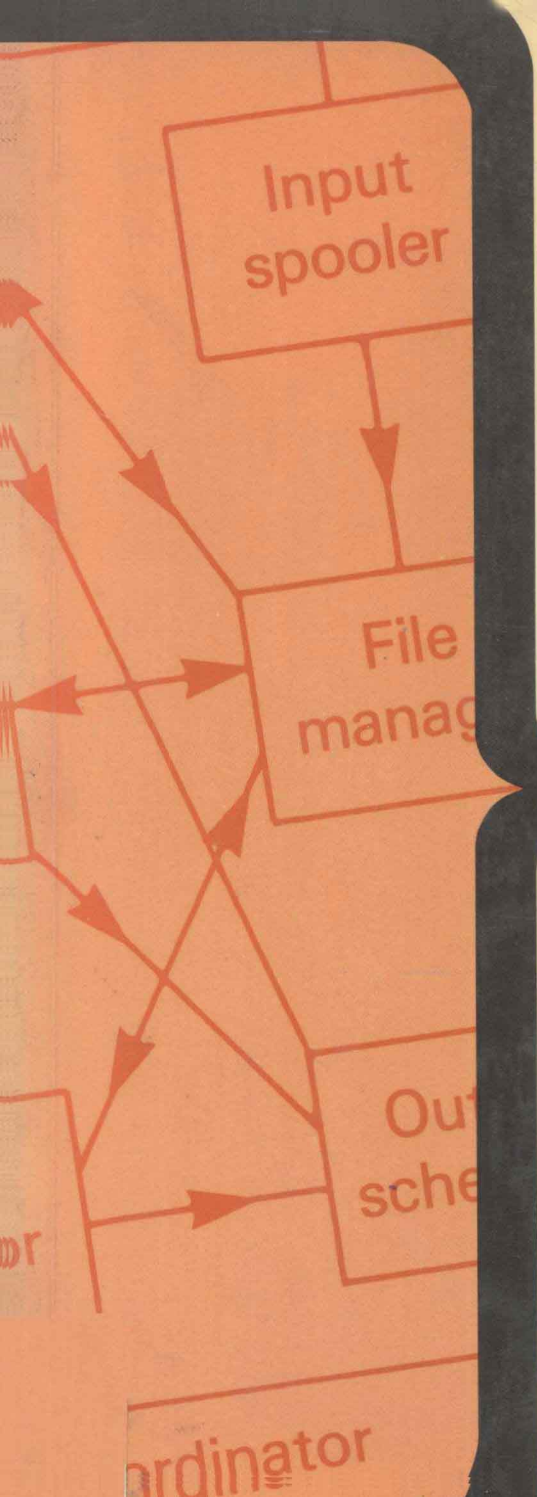

A Practical Course on Operating Systems

Colin J. Theaker
and
Graham R. Brookes

A Practical Course on Operating Systems

Colin J. Theaker

Senior Lecturer in Computer Science,
University of Manchester

Graham R. Brookes

Senior Lecturer in Computer Science,
University of Sheffield

First published 1983 by
THE MACMILLAN PRESS LTD
London and Basingstoke
Companies and representatives
throughout the world

Printed in Great Britain by
Unwin Brothers Limited
The Gresham Press, Old Woking, Surrey

ISBN 0 333 34678 5
0 333 34679 3 pbk

Preface

An operating system is one of the most important pieces of software to go into any modern computer system, irrespective of its size, yet because systems seek to be transparent to the computer user, much mystique surrounds their design. As computer systems increase in complexity, particularly at the microprocessor level, more complexity appears in the system software. The primary objective of this book is to dispel the mystique associated with operating system design, and to provide a greater practical understanding of the design and implementation of operating systems.

The contents of this book are derived from a number of courses of lectures given to undergraduate students of computer science. In its entirety it provides suitable material for a full course on operating systems for students who have a basic grounding in computer science, or who have some practical experience of computing. However, less detailed courses may be derived by choosing a sub-section of the chapters. For example, the first four chapters provide a simple overview of the structure of an operating system, and can therefore be used as material for a lower level course, which seeks to provide a general discussion of the nature and role of an operating system.

The first four chapters also provide the background for the more detailed considerations that follow. This begins with an examination of scheduling principles and the algorithms associated with scheduling. The treatment of memory management traces much of the evolutionary path of operating systems, from the need for base-limit registers to the design of paged segmented machines. Subsequent chapters examine the problems of resource management, including the protection of resources and the avoidance of deadlocks. The problems of concurrency are then examined, and a number of techniques for achieving cooperation and synchronisation of processes are described. Finally, the interface most familiar to the user, namely the job control language is considered.

Throughout the book there are exercises for the student at the end of each chapter, together with references in which the student can find more detailed information on specific topics.

The authors wish to acknowledge the assistance and encouragement of colleagues, in particular to Dr G.R. Frank, whose lectures on operating systems at the University of Manchester provided much inspiration for this book, and more generally to Professor D. Morris and Professor F.H. Sumner. They thank Professor D. Howarth for helpful suggestions in earlier drafts. Finally, they thank Margaret Huzar for her patience in typing the manuscript of this book throughout its many stages.

Contents

Part 1 Design of an Operating System

1 Basic Operating System Concepts

We shall begin our consideration of operating systems by asking the following questions:

(1) What is an operating system?
(2) Why are operating systems needed?
(3) Do situations exist when they are not needed?
(4) How would an operating system be designed?

The first part of this book aims to provide answers to these questions. During the course of this, we shall present a simple overview of the design of an operating system. The subject is developed in more detail in the second part of the book, where particular problem areas are discussed together with algorithms used within operating systems. Whenever possible, the book contains illustrations of a theoretical operating system written in the style of the language Pascal.

Take the first question – 'what is an operating system?'. In simple terms it is just a program, but its size and complexity will depend on a number of factors, most notably the characteristics of the computer system, the facilities it has to provide and the nature of the applications it has to support. For example, the operating system for a single user microprocessor can be relatively simple in comparison with that for a large multi-user mainframe computer.

Regardless of size, the operating system is usually the first program loaded into the computer when the machine is started. Once loaded, some portions of it remain permanently in memory while the computer is running jobs. Other portions of the operating system are swapped in and out of memory when facilities are required by the users.

To answer the second – 'Why are operating systems needed' – it is worth stating the basic objectives that an operating system is seeking to attain:

(1) To provide a higher level interface so that the hardware of the computer becomes more readily usable.

(2) To provide the most cost effective use of the hardware of the computer.

Operating systems attempt to satisfy both of these objectives although, in practice, these requirements are not mutually exclusive and so a compromise in design has to be made. In consequence, there are many types of operating system. In this book we shall seek to illustrate the principles behind a range of operating systems rather than provide a full consideration of any specific system.

As with most complex pieces of software, it is possible to regard the structure of an operating system as a layered object, analogous to, say, an onion. At the centre is a nucleus of basic facilities, to which additional layers are added as required to provide more sophisticated facilities. Some modern operating systems, such as VME/B for the ICL 2900 (Keedy, 1976; Huxtable and Pinkerton, 1977) and UNIX (Ritchie and Thompson, 1974), exhibit this neatly layered structure and some machines even provide hardware support for such a layered organisation.

Rather than starting with such a system and decomposing it to identify the important components of an operating system, the approach we intend to take in this book is to concentrate initially on the nucleus of the system. Starting at the most primitive level, the hardware, we shall consider the design of a very simple operating system, examining its limitations and hence identifying what improvements and enhancements would be required to provide a powerful and sophisticated system. We thus intend to build up the design of an operating system in an evolutionary manner. In many respects, this reflects the historical development that has led to the present structure of modern operating systems.

This book is divided into two parts. In the first, we examine the needs of a very simple system and develop its design so that, by the end of part 1, the framework of an operating system has been established. In the second part, we identify the deficiencies of this system and examine techniques and algorithms that can be used to resolve these problems.

1.1 GENERAL FEATURES

Initially it is necessary to identify the sort of services that an operating system might provide to help the user run a program. These are somewhat analogous to the existence of assemblers and compilers that allow a user to write a program in languages other than binary code.

(1) Convenient input/output operations.
Users do not want to know the details of how a particular peripheral has to be driven in order to read or print a character. Clearly, a higher-level interface than this must be provided for the user.

(2) Fault monitoring.
No matter how proficient the programmer, it is impossible for anyone to write faultless programs all the time. It is therefore essential for the system to cater for errors arising in a program. When errors are detected, the operating system intervenes to print suitable monitoring information to help the user find the source of the fault. Various levels of monitoring information may be printed, some by the operating system, some by the compiler or run-time support system of the programming language.

(3) Multi-access.
Allowing several people to use the computer simultaneously is more convenient for the users, even though some users might suffer a longer response time at their terminal than they would if they had sole use of the computer.

(4) File systems.
The operating system maintains the directory and security of a user's files. Centralised control is necessary in order to allow several users to share the same hardware while maintaining a secure file system where the files of each user are protected from invalid access. An operating system might also provide utilities for accessing and manipulating the files (for example, editors).

1.2 PERFORMANCE CONSIDERATIONS

An important requirement of operating systems is to make the most cost-effective use of the computer hardware. This was particularly important in the early days of computing when the cost of even the most primitive of machines was quite substantial. Although technological advances have made modern microprocessor-based systems far more cost-effective, the problem of achieving good performance still remains with the larger mini and mainframe machines. In examining the problem of achieving good performance from a computer system, consider the system shown in figure 1.1.

To allow each user sole access to the computer hardware would mean in practice that the computer would be idle for long periods of time. For example, when the user was loading a program into the machine, it would be doing no other useful work. Even when a computer is running a job its efficiency may be very poor. Consider the example of running a simple assembler. The assembler might be organised to read a card from the card reader, generate the necessary instruction and plant it in memory, and print the line of assembly code on the

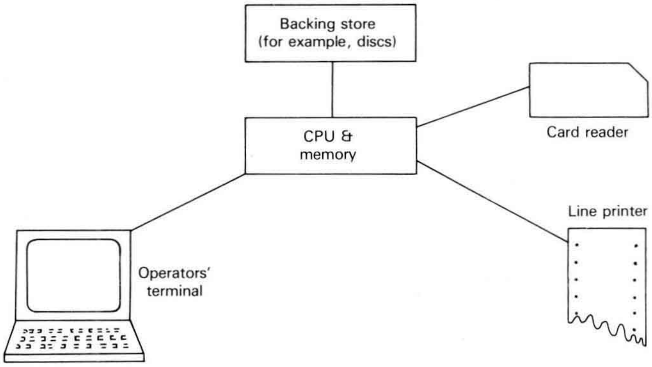

Figure 1.1 A simple computer system

lineprinter. Thus, in the sequence of processing each line there are three phases of operation:

INPUT PHASE	PROCESS PHASE	OUTPUT PHASE
(Read a card)	(Assemble the instructions)	(Print the line)

Assuming that the computer system has the characteristics of, say, a large minicomputer:

Card reader	300 cards/min
Lineprinter	300 lines/min
Central processing unit (CPU)	1 μs/instruction

Also, assuming that it takes about 10000 instructions to assemble each line, then the times for each of the three phases to process a card are:

(1) Input phase	200 ms
(2) Process phase	(10000 * 1) μs = 10 ms
(3) Output phase	200 ms

The CPU, which is only actively in use during the process phase, is busy for only 10 ms in every 410 ms.

The efficiency for CPU utilisation is defined by the following formula:

$$\text{Efficiency} = \frac{\text{Useful computing time}}{\text{Total time used}} * 100 \quad \text{per cent}$$

Using the characteristics above yields an efficiency given by:

$$\text{Efficiency} = \frac{10}{410} * 100 \simeq 2.4 \text{ per cent}$$

It is clearly very ineffective to use the CPU at such a low efficiency.

1.3 INPUT/OUTPUT LIMITED JOBS

We can readily see some possible improvements that might increase the utilisation of the computer system. It has been assumed in the calculations that the input phase (that is, the card reader) was started only when the process and output phases were complete. This would be achieved with the following control sequence for the card reader:

```
read card :  REPEAT
                start reader
                WHILE reader not finished DO nothing
             UNTIL reader not in error

             process card
```

This sequence, which results in the timing shown in figure 1.2, has the disadvantage that the card reader is idle during the time when the card is being processed and printed.

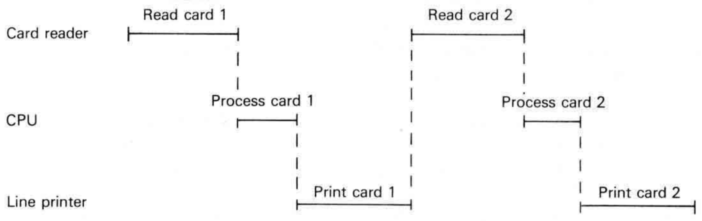

Figure 1.2 Process sequence for a simple system

The obvious way to speed this up is to copy the card image on to an intermediate area from where it can be processed and then to restart the reader immediately. The sequence that controls the card reader now becomes: