



Input
spooler

File
manag

Out
sche

ordinator

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



© Colin J. Theaker and Graham R. Brookes 1983

All rights reserved. No part of this publication may be reproduced or transmitted, in any form or by any means, without permission.

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

Preface

PART 1 DESIGN OF AN OPERATING SYSTEM.

1	Basic Operating System Concepts	2
1.1	General Features	3
1.2	Performance Considerations	4
1.3	Input/Output Limited Jobs	6
1.4	CPU Limited Jobs	7
1.5	Summary	8
1.6	References	9
1.7	Problems	9
2	Performance of Input/Output Systems	10
2.1	Simple Principles of Multiprogramming	10
2.2	The Use of Interrupts	11
2.3	The Concept of Buffering	12
2.4	Implementation of a Simple Buffering System	13
2.5	Summary	16
2.6	Problems	16
3	Spooling	18
3.1	Offline Spooling	18
3.2	Online Spooling	20
3.3	Design of a Simple Spooling System	23
3.3.1	Input System	25
3.3.2	Job Scheduler	25
3.3.3	Job Processor	26
3.3.4	Output Scheduler	27
3.3.5	Output System	27
3.3.6	Disc Manager	28
3.3.7	The Coordinator	28
3.4	Problems	29
4	Time-Sharing Systems	31
4.1	Characteristics of the Time-sharing System	31
4.2	Design of a Time-sharing System	32
4.3	Problems	34

PART 2 OPERATING SYSTEM TECHNIQUES

5	Buffering Techniques	36
5.1	More Sophisticated Buffering Techniques	36
5.2	Double Buffering	37
5.3	Cyclic (Circular) Buffering	38
5.3.1	Requirements	39
5.3.2	Implementation	40
5.4	Problems	41
6	Scheduling - Principles	42
6.1	Preemptive and Non-preemptive Scheduling	42
6.2	Time-slicing	44
6.3	Choice of a Time Slice Period	46
6.4	Estimation of Response Time	48
6.5	Problems	50
7	Scheduling - Algorithms	52
7.1	Objectives of Scheduling	52
7.1.1	Response/turnaround time	53
7.1.2	Meeting user-specified deadlines	53
7.1.3	CPU utilisation	53
7.1.4	Utilisation of other resources	54
7.2	Deterministic Scheduling for Independent Tasks on a Single Processor	54
7.3	Simple Queuing System	55
7.3.1	Arrival distribution	56
7.3.2	Service distribution	56
7.3.3	Queue length	56
7.3.4	Waiting time	56
7.3.5	Little's result	56
7.3.6	Utilisation factor	57
7.4	Single-processor Non-deterministic Scheduling	57
7.4.1	First-come-first-served scheduling	57
7.4.2	Non-preemptive shortest processing time scheduling	58
7.4.3	Shortest remaining processing time scheduling	59
7.4.4	Comparison of response time	60
7.5	Single-processor Time-sharing Systems	61
7.5.1	Derivation of response in a time-sharing system with no time slicing	61
7.5.2	Derivation of response in a time-sharing system using a Round Robin algorithm for time slicing	63
7.5.3	The overheads of preemption	64
7.6	References	65
7.7	Problems	65
8	Memory Management - Basic Principles	67
8.1	Swapping Strategies	68
8.1.1	Simple swapping system	68

8.1.2	A more complex swapping system	70
8.1.3	Further developments of the swapping system	71
8.2	Memory Protection	72
8.3	Virtual Addressing	73
8.4	References	75
8.5	Problems	75
9	Memory Management - Segmentation	77
9.1	Fragmentation	77
9.2	Program Locality	78
9.3	Sharing of Code and Data	79
9.4	Multiple Base-limit Register Machines	80
9.5	Address Translation on a Multiple Base-limit Register Machine	82
9.6	Shared Segments	85
9.6.1	All direct	85
9.6.2	One direct, all others indirect	86
9.6.3	All indirect	88
9.7	Common Segments	88
9.8	References	91
9.9	Problems	91
10	Memory Management - Paging Systems	92
10.1	Paging	92
10.2	Address Translation in a Paged Machine	93
10.2.1	Address translation using page address registers	93
10.2.2	Address translation using current page registers	95
10.3	Paged Segmented Machines	97
10.4	Store Management in a Paged Segmented Machine	98
10.4.1	Segment table base register	99
10.4.2	Process segment table	99
10.4.3	System segment table	101
10.4.4	Page table	101
10.4.5	Loading of current page registers	102
10.5	Action on a Virtual Store Interrupt	103
10.5.1	Virtual store interrupt procedure	103
10.5.2	Find empty page procedure	104
10.5.3	Store rejection algorithm	105
10.5.4	Disc Interrupt procedure	107
10.6	References	107
10.7	Problems	108
11	Memory Management - Algorithms and Performance	109
11.1	Performance	109
11.2	Locality	113
11.3	Page Replacement Algorithms	115
11.3.1	Belady optimal replacement algorithm	115
11.3.2	Least recently used algorithm	116
11.3.3	First in first out algorithm	117
11.3.4	Not recently used algorithm	118

11.4	Stack Algorithms	118
11.5	Multiprogramming	119
11.5.1	Reducing individual swap times - prepagging	120
11.5.2	Improving swap rates by multiprogramming	120
11.6	Thrashing	121
11.6.1	Thrashing prevention by load control	122
11.6.2	Thrashing prevention by controlling interference	123
11.7	Storage Allocation Techniques	124
11.7.1	First fit algorithm	125
11.7.2	Next fit algorithm	125
11.7.3	Best fit algorithm	125
11.7.4	Worst fit algorithm	126
11.7.5	Buddy system	126
11.8	References	126
11.9	Problems	127
12	File Management	128
12.1	Requirements of a File System	128
12.2	Directories and Access Rights	129
12.2.1	Single-level directory structure	130
12.2.2	Hierarchical directory structure	130
12.3	Dumping and Archiving	131
12.4	Secure File Updating	132
12.5	References	134
12.6	Problems	134
13	Resource Management - Deadlocks	135
13.1	Allocation of Resources	135
13.2	Deadlocks	136
13.2.1	Deadlock prevention	137
13.2.2	Deadlock detection	138
13.2.3	Deadlock avoidance	139
13.2.4	The bankers algorithm	140
13.3	References	142
13.4	Problems	143
14	Resource Management - Protection	144
14.1	Introduction to Protection Systems	144
14.2	A General Model of Protection Systems	145
14.2.1	Defining and representing domains	146
14.3	Interdomain Transitions	148
14.3.1	Domain changes only on process changes	149
14.3.2	Restricted domain change systems	149
14.3.3	Arbitrary domain changes	151
14.4	An Example of a Capability System	153
14.4.1	Types and access amplification	154
14.4.2	Creation of new types	156
14.4.3	An example	156
14.5	References	157
14.6	Problems	157

15	Process Synchronisation – Basic Principles	158
15.1	Introduction	158
15.1.1	Process synchronisation	159
15.1.2	Process competition	159
15.2	Flags	161
15.3	Semaphores	163
15.3.1	Mutual exclusion by semaphores	164
15.3.2	Process communication using semaphores	164
15.4	Implementation of Semaphores	165
15.5	Semaphore Implementation for a Multiprocessor System	167
15.5.1	Hardware implementation	167
15.5.2	Software implementation	168
15.6	Semaphore Implementation for Multiprocessing Systems	169
15.7	An Example in the use of Semaphores – the Readers and Writers Problem	169
15.7.1	Readers have priority	170
15.7.2	Writers have priority	171
15.8	References	172
15.9	Problems	173
16	Process Synchronisation – Advanced Techniques	174
16.1	Introduction	174
16.2	Message Systems	175
16.3	Message-passing Operations	176
16.4	Practical Considerations	177
16.4.1	Form of the message	177
16.4.2	Addressing of processes	178
16.4.3	Type of message-queuing discipline	178
16.4.4	Message validation / protection	179
16.4.5	Message deadlocks	179
16.5	Solution of the Readers and Writers Problem using Messages	180
16.5.1	Design of the controlling process	180
16.6	The Language Based Approach – Monitors	183
16.6.1	Encapsulation	184
16.6.2	Mutual exclusion	185
16.6.3	Communication	185
16.7	Modula-2	186
16.7.1	Modules	186
16.7.2	Processes – coroutines	187
16.7.3	An example module	188
16.7.4	Devices and interrupts	189
16.8	References	190
16.9	Problems	190
17	Job Control Systems	191
17.1	Functions of the Job Control System	191
17.2	Types of Job Control Language	193
17.3	Requirements of Job Control Languages	194

17.4	Additional Features	195
17.5	Standardisation	196
17.6	Procedural Approach to Job Control	196
17.7	References	197
17.8	Problems	197
<i>Index</i>		199

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

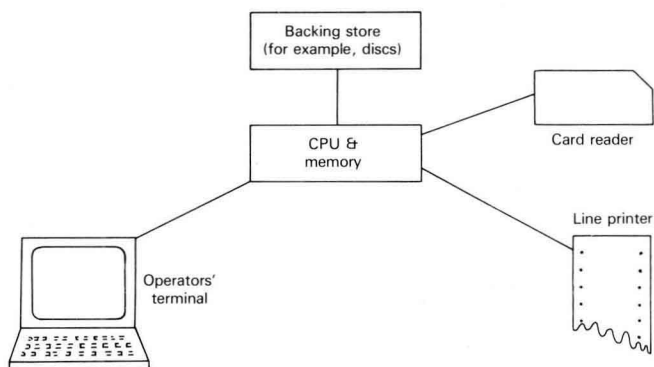


Figure 1.1 A simple computer system

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 μ s/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	$(10000 \times 1) \mu\text{s} = 10 \text{ ms}$
(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:

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

Using the characteristics above yields an efficiency given by:

$$\text{Efficiency} = \frac{10}{410} \times 100 \approx 2.4 \text{ per cent}$$

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.

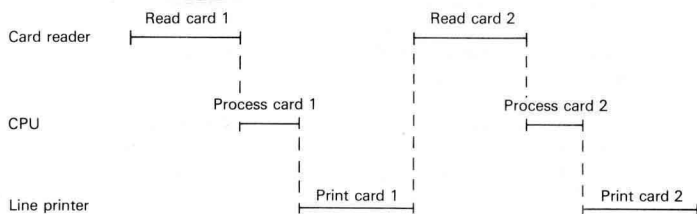


Figure 1.2 Process sequence for a simple system

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: