Mathai Joseph,
V. R. Prasad and
N. Natarajan

A Multiprocessor Operating System

PRENTICE-HALL
INTERNATIONAL
SERIES IN
COMPUTER
SCIENCE

C. A. R. HOARE SERIES EDITOR

A MULTIPROCESSOR OPERATING SYSTEM

Mathai Joseph
V. R. Prasad and
N. Natarajan
NCSDCT, Tata Institute of Fundamental Research, India

Prentice/Hall  International

Englewood Cliffs, NJ London New Delhi Rio de Janeiro
Singapore Sydney Tokyo Toronto Wellington

Library of Congress Cataloging in Publication Data
Joseph, M.
A multiprocessor operating system.
Bibliography: p.
Includes index.
1. Operating systems (Computers) 2. Multiprocessors.
I. Prasad, V. R., 1950– . II. Natarajan, N., 1950–
III. Title.
QA76.6.J68 1984 001.64 83 -11204
ISBN 0–13–605170–7

British Library Cataloguing in Publication Data
Joseph, M.
A multiprocessor operating system.
1. Multiprocessors
I. Title II. Prasad, V. R. III. Natarajan, N.
001.64 QA76.5
ISBN 0–13–605170–7

0-13-605170 7

Prentice-Hall International, Inc., *London*
Prentice-Hall of Australia Pty, Ltd., *Sydney*
Prentice-Hall Canada, Inc., *Toronto*
Prentice-Hall of India Private Ltd., *New Delhi*
Prentice-Hall of Japan, Inc., *Tokyo*
Prentice-Hall of South-East Asia Pte., Ltd., *Singapore*
Prentice-Hall Inc., *Englewood Cliffs, New Jersey*
Prentice-Hall do Brasil Ltda., *Rio de Janeiro*
Whitehall Books Ltd., *Wellington, New Zealand*

Typeset in the UK by Alphabyte Ltd., Cheltenham
Printed in the United States of America

10 9 8 7 6 5 4 3 2 1

FOREWORD

There are three reasons why I welcome the publication of this book in the Prentice-Hall International Series in Computer Science.

Firstly, it makes an exemplary contribution to the objectives of the series: it treats the design and implementation of major computing systems programs as a topic of national and professional study, of which a complete understanding can be obtained by reading well-crafted and elegant code.

Secondly, it brings wider recognition to the valuable and practically oriented research in Computer Science conducted at the Tata Institute of Fundamental Research in Bombay.

Thirdly, it appeals to my own long-standing personal interests, both in operating systems and in the programming of multiple processor networks. If all those engaged in systems programming were to study books like this, we should avoid perpetuating the long series of technical failures which have plagued us since the 1960s.

As I was personally responsible for one of the earliest of these failures, publication of a book describing a successful solution gives me special satisfaction.

C. A. R. Hoare

PREFACE

It is customary for a course on operating systems, as in many other fields, to teach principles and techniques. Such courses often end with exercises requiring students to construct parts of operating systems that illustrate the use of these techniques. But practical reasons force most courses to stop short of taking students through 'real world' operating systems and only a few particularly energetic participants pursue their interests through the inhospitable documentation of commercial operating systems. This reinforces the widespread belief that principles and theory are for courses, not for practical use, and that there is a welter of problems that are neither described in courses nor solvable by the techniques that are taught.

The purpose of this book is to present the design of what we hope will be seen as a 'real' operating system, in a manner that makes it possible to understand the problems that must be solved in the practice of constructing operating systems. The designs of major components of the operating system are described by developing programs and, finally, these programs are integrated to form the whole system. Thus, much of the book consists of program text written in an extended form of Pascal and, with some additional system-dependent code, these programs can actually be put together to form a working operating system. The material is presented in this form to illustrate the importance of studying well-structured programs in learning how to construct new programs in this area (and similar ones).

The first three chapters are purely introductory. Chapter 1 has a general introduction, Chapter 2 describes the CCNPascal language which is used for all the programs in the book, and Chapter 3 gives a summary of

multiprocessor architectures. The design of the operating system starts in Chapter 4, where we take some simple requirements and develop the outlines for the components that must be constructed. The first such component is a basic memory allocator, whose design is described in Chapter 5, and this is then used for building a main memory allocator (Chapter 6) and a disk space allocator (Chapter 7). The next two chapters describe a complete file system; a relatively simple 'user' view of the file system appears in Chapter 8 and the details of the file system structure are presented in Chapter 9. The handling of input and output using physical devices (as opposed to virtual file 'devices') is described in Chapter 10. These components are brought together in Chapter 11, where they are used for the management of jobs; this chapter deals with scheduling and memory management and completes the design of the visible part of the operating system. Chapter 12 describes the kernel which underlies the operating system and provides abstract views of the physical hardware, in addition to supporting the implementation of CCNPascal programs. Chapter 13 reviews the techniques used in the book and the design of the operating system.

The book has not been designed for a particular teaching course, though we have envisaged its use in many different kinds of courses. Understanding the material in the book does require some background: familiarity with simple operating systems, and with the use of programming abstractions such as the *monitor*, the *class* and the *process*. With this in hand, the book could be used in several ways:

1. As *adjunct* material to a course, where parts of the book are used as case-studies,
2. As the basis of a one-semester second course on operating systems, using Chapters 1–8, Chapter 10 and the first part of Chapter 11, or
3. As a two-semester course on operating systems.

There are several sets of exercises, some to modify programs given in a chapter and others to write programs to meet different constraints. A few of the exercises are designated as group exercises, as they could profitably be attempted by small groups of students working together.

But the material in the book is not delimited by what can easily be taught. Chapter 9, on the structure of the file system, the second part of Chapter 11, which deals with fairly complex job management, and much of the description of the kernel in Chapter 12, are all undoubtedly difficult. Rather than simplify the design to eliminate this material, we have chosen to present it because of its use to another reader we have in mind: the professional programmer in industry. Moreover, having seen the tenacity

and endurance with which some students plough through documentation on commercial operating systems, we feel that these parts of the book may even be of use to students.

Despite its size, there is much that is *not* described in the book. Consistency and crash-recovery in the file system, swapping policies and swapper programs, operator control of the system parameters, the use of virtual memory, and several other aspects that have bearing on the performance and usability of an operating system have only been outlined in text. Such areas are important in their own right, but a line must inevitably be drawn to separate what is presented from what is not. We have chosen to omit descriptions that would add considerably to the size of the book, and those where detail would appear to outweigh structure. This book will have served its purpose well if it encourages readers to write elegantly structured programs for these aspects of operating systems.

Historical Background

The operating system described in this book is based on a very similar one that was actually built for a multiprocessor system (a list of publications on this project is given in the References section). Since its development had some interesting aspects, a short account of the background is given here.

In the first few weeks of 1975, we had the opportunity of participating in a workshop organized at the National Centre for Software Development and Computing Techniques (NCSDCT) with the assistance of the United Nations Development Programme. Among other things at this workshop, two important new developments were described: P. Brinch Hansen gave a series of lectures on his language, Concurrent Pascal, and W. A. Wulf discussed the design of C.mmp, the multiprocessor then under construction at Carnegie-Mellon University, and Hydra, its operating system. At that time, some of us had already been talking of the possibility of building a multiprocessor system using Indian-made TDC 316 computers and the workshop served as a stimulus in crystallizing these ideas into the more definite proposal that was submitted a few months later to the Electronics Commission of the Government of India. This proposal was approved and, by the end of the year, the first TDC 316 machine had been delivered. In the meantime, we had been examining how Concurrent Pascal could be altered to meet the requirements we had in mind, and working on simple schemes for interconnecting several TDC 316 computers to form what we called the Close-Coupled Network (CCN).

By the end of 1976, we had an experimental compiler, modelled on the Pascal compiler and written in Pascal on the DEC System 10, for our

version of Concurrent Pascal; by then, the changes we had made in the original language made it prudent to seek a new name and 'CCNPascal' suggested itself as a suitable alternative. Also, three TDC 316 computers had been linked together in equally experimental fashion. Having the well-known properties of experimental designs, neither the compiler nor the system worked with adequate reliability for sustained use and much of 1977 was spent in bringing them to a state where other kinds of experimentation, such as implementation of parts of the operating system, could begin. Various ambitious designs for the operating system were tried out and it was two years later that a final, and altogether simpler, operating system was completed and the first 'user' programs were executed.

In retrospect, it was risky to have attempted to use a new and relatively untested computer for a system of this nature, especially as the small project team (about 4–5 members at any time) was also involved in the design of the new features of the CCNPascal language, in the development of its first compiler, and in the use of this language for programming the operating system and its kernel, each of which went through several versions. Nevertheless, though we never quite solved the problems of hardware unreliability with our prototype system, much of the program development work went on without unmanageable difficulties. The lesson, that use of a good programming language and systematic design techniques are of irreplaceable value, is not one we are likely to forget.

Acknowledgements

The CCN project was financed by the Electronics Commission of the Government of India and we owe a great deal for this support. The National Centre for Software Development and Computing Techniques, to which all the authors are affiliated, provided support right through the project and, subsequently, during the writing of this book: our thanks go to its director, R. Narasimhan, and to many of our colleagues (past and present) who helped in the project. The Electronics Corporation of India Ltd. (ECIL) went far beyond the call of commercial duty to construct the special hardware needed to interconnect their TDC 316 computers and to help us to maintain the hardware.

At various times, and in various measures, several people at NCSDCT worked on the design and implementation of the CCN software: R. Viswanathan and K.V.S. Prasad were involved with the early designs for the Kernel, Satish Thatte designed the first version of the file system, Sandhya Desai wrote the system loader which linked together the modules of the operating system, and K.T. Narayana did a great deal of work on the first version of the CCNPascal compiler. The final form of the Kernel and

the operating system, and their implementations, owe much to the work of Mukul Sinha, who also contributed to the system design of the hardware, and of I.V. Ramakrishnan. At ECIL, A. K. Kaul and P.V.S. Nayak worked on the design of the special hardware. During the course of the development, M.V. Wilkes was an annual visitor who commented on the design; another annual visitor, W.A. Wulf, spent considerable time and effort in studying our design and in providing details on the progress of the C.mmp project.

The suggestion that we write a book about the operating system came from C.A.R. Hoare, who is not only the editor of this series of books and the originator of many of the techniques we have used, but a long standing campaigner for the publication of programs; the fact that we accepted the offer is largely due to his persuasion that the effort was worth making. To him, to H. Hirschberg and R. Decent of Prentice-Hall International, and to R.M. McKeag and R. Gimson, who patiently and painstakingly reviewed the manuscript, considerable thanks are due.

Several other readers have helped us with comments on the manuscript: H.N. Mahabala, V.K. Joglekar, R. Chandrasekhar, and K. Lodaya, to name a few. To the others, and to the many people upon whose work this book has depended, we offer our special thanks.

Text for this book was typed, edited and formatted on the NCSDCT DECSystem10 using standard text editors and the NCSDCT text composition system DIP.

Mathai Joseph
V.R. Prasad
N.Natarajan

NCSDCT
November, 1982

CONTENTS

Foreword xi

Preface xiii

1 INTRODUCTION 1

Writing an operating system 2
Design problems 4
The language 6
The architecture 8
The operating system 9
The implementation 10
Summary 10

2 THE CCNPASCAL LANGUAGE 12

Sequential Programming 12
Generic types 14
Subprograms 15
Repetitive statements 16
Error returns 18
Data abstraction: the class 20
Multiview records 25
Concurrent Programming 30
Process 30
Monitor 31

Queues 33
Resource scheduling 35
Pure class 37
Signals and queues
Standard subprograms 43
Discussion 44
Compilation of programs 44
Summary 44
Exercises 45

3 MULTIPROCESSOR ARCHITECTURE 48

Concurrency in the operating system 48
System Architectures 49
Multiprocessors 50
Processor–memory interconnection 51
Input/output structure 56
Interprocessor communication 58
Programming Issues 58
Mutual exclusion 58
Resource allocation 59
A Model Architecture 60
Discussion 61
Exercises 63

4 OVERVIEW OF THE OPERATING SYSTEM 64

Introduction 64
Requirements 65
Job Management 66
The user view 66
The internal view 68
File Management 72
The user view 72
The internal view 76
Input/Output On Devices 80
The user view 80
The internal view 82
Memory Management 83
The user view 83
The internal view 84
The Kernel 86
Discussion 89
Exercises 89

5 A BASIC MEMORY ALLOCATOR 91

Introduction 91
Representing memory in an operating system 92
Outline of a memory allocator 94
Allocating segments from a large chunk 96
Returning memory—fragmentation problems 98
Choosing from a set of segments 98
Compaction of segments 101
Merging segments with MainChunk 101
Operation of the whole allocator 102
Making the allocator generic 106
Discussion 107
Summary 110
Exercises 110
The Basic Memory Allocator Program 112

6 THE MAIN MEMORY ALLOCATOR 118

Introduction 118
Outline of the main memory allocator 119
Delaying requests 121
Discussion 124
Exercises 125
The Main Memory Allocator Program 126

7 THE DISK SPACE ALLOCATOR 129

Introduction 129
Allocation in a group 131
Outline of the disk allocator 133
Choosing groups for allocation 135
Reporting errors 139
Storing allocation data on the disk 140
Mounting and initializing a disk pack 143
Crash recovery 146
Discussion 147
Exercises 149
The Disk Space Allocator Program 150

8 THE FILE SYSTEM—PART I 161

Introduction 161
Files 162
Random files 164
Internal structure of a file 165

Implementation Of File Operations 166
Sequential files 166
Random files 173
Directories 174
Sharing objects 175
Creating links 176
Managing objects and names 177
Internal Structure Of Directories 180
The structure of a directory 183
Implementation of the Indirectory 183
Active Files 185
Concurrency control of file transactions 188
Discussion 192
Summary 193
Exercises 194

9 THE FILE SYSTEM—PART II 196

Jobs And Directories 196
Associating directories with jobs 196
Setting the current directory 199
Checking space quota limits 201
The Master File Directory 202
Maintaining Directories 202
Error reporting 203
Directory lookup 205
Creating a new capability 207
Updating an existing capability 208
Renaming a capability 209
Handling links 210
Deletion of capabilities 212
Setting the current directory 213
Managing File Operations 216
Managing file descriptors 217
Maintaining file attributes 220
Implementing file operations 221
Implementing File System Procedures 227
Setting directories 227
Creating a subdirectory 228
Name management 229
Deletion of objects and links 230
Sharing 231
File System Administration 234
Registering a new user 234
Authentication 236
Pack and MFD creation 237

Discussion 238
The file system structure 238
Representation of files and directories 240
Crash recovery 241
Exercises 242
The Complete File System Program 243

10 INPUT/OUTPUT HANDLING 299

Introduction 299
Device Handling in the Kernel 300
Disk input/output 302
Assigned devices 304
Device independence 306
Device allocation 308
Discussion 309
Device spooling 309
Input spooling 310
Summary 310
Exercises 311
The Input/Output Program 313

11 JOB MANAGEMENT 324

Introduction 324
User job execution 324
Job representation 326
Levels of scheduling 328
Management Of Jobs And Programs 332
The job table 332
The program table 335
The CPU queues 340
The processor scheduler 343
Scheduling processes 344
The procedures Enter and Depart 348
Program table—detailed structure 351
Choosing segments to be swapped 361
Swapper processes 363
The Job Manager's cycle 364
Handling errors detected by the Kernel 366
Discussion 367
The operator process 367
Determination of Jobmix and processor slice 367
Call mechanisms and protection 369
User program structure 370
Operating system process structure 372
User level concurrency 373
Paging, segmentation and virtual memory 374

Exercises 375
Program Components For Job Management 377

12 THE KERNEL 399

Modes of operation 399
Entry to the Kernel 400
Operating system calls from a user 402
Kernel calls from the operating system 403
Execution Of Processes And Programs 404
Process representation 405
Executing processes 409
Program calls 411
Pre-emption of program execution 412
Context switching 414
Miscellaneous 415
Process Synchronization 416
High-level monitors 417
Queues 419
Low-level monitors 420
Routing calls to synchronization operations 423
Disk device 425
Input/Output Programming 424
Card reader 428
Line printer 428
Terminals 429
Routing calls to input/output operations 434
Miscellaneous 435
Discussion 435
Exercises 436
The Kernel Program 437

13 REVIEW AND CONCLUSIONS 456

Review of the operating system 456
The programming language 457
Operating system structure 460
Building and testing the operating system 460
Practical results 461
Summary 462

BIBLIOGRAPHY 465
Bibliography on the CCN project 467

Index 469

Program Component Index 477

1 INTRODUCTION

When good fortune, or a benevolent funding agency, presents us with a new computer system, one of the first tasks is to get hold of a manual for the operating system. A quick study of this manual leads us to a time-sharing terminal, to 'try things out'. A little later, those of us who have not retreated from the complexities of command formats, job control languages and inexplicable error messages, can be found going through the manual again to get a more complete idea of how the system can be used. And this goes on until each user has acquired the information needed to use the system for her or his problems. When a new problem arises, we return to the manual (or to a local wizard) to see if there is a simple solution; or, remembering that 'there's something about it in the manual', we look for the section that tells us how it can be done.

After some familiarity with the system, we begin to wonder *how* the operating system performs its functions. (The last course on operating systems taught us many useful principles but it did not prepare us for the way *this* operating system seems to work!). Discussions with the wizard prove to be tantalizingly incomplete, and there is no alternative but to see what documentation the 'system people' can produce. And there our problems begin. There are shelves full of manuals of internal documentation and each manual seems to expect us to know information present in other manuals. There are detailed flow charts, diagrams showing how bits are packed into table entries and, to confuse things further, amendment sheets that purport to alter what is in the manuals. Having been through the manufacturer's training course, the system programmers seem to

navigate successfully through all this documentation. But, like other experts, they answer every request for general information with the question 'What exactly do you want to know?'

And that would be a reasonable question, were it not for the fact that before we know *exactly* what to look for in the detailed documentation, we need to know the *general structure* of the system!

This story is familiar to many of us, and it has different endings. There are those who, not surprisingly, decide that the gap between theory and practice in operating systems is too large for the field to be of any interest to them. Then there are those who are convinced that the theory is perfectly adequate, *as theory*, but that practical operating systems have so much more excitement to offer (at the other extreme, there are those who gratefully return to the theory to wait until the practice becomes more respectable). And, finally, there are the resolute few who find structure in the mass of documentation, and go on to examine this in the light of the available theory.

There is no particular moral to this story, or to its endings. But this book has been written to show that it *is* possible to understand both the structure and the detail of a reasonably large operating system. To do this, we shall go through the exercise of constructing such an operating system for a multiprocessor system. The program components for this operating system will be systematically developed and, towards the end, these components will be brought together to form the operating system.

Writing an Operating System

Like other large programs, the design of an operating system presents problems of specification and correctness, of testing and performance evaluation, and of documentation. This book will be concerned with many such issues because they must be resolved if a correct, efficient and useable program is to be constructed. But since we shall be looking particularly at the question of designing an *operating system*, we are also confronted with issues of concurrency—i.e., many actions taking place simultaneously—and we cannot take recourse to having some other program control the computer system and its peripherals, or handle the errors that arise in execution. These two factors play a large role in determining the kinds of design we can consider for an operating system.

Designing a new operating system, like studying or modifying an existing one, is an exciting exercise for many reasons. The simplest reason, though perhaps the most misleading one, is that an operating system is a program