SOFTWARE TOOLS FOR OS/2

Creating Dynamic Link Libraries

Michael J. Young

SOFTWARE TOOLS FOR OS/2

Creating Dynamic Link Libraries

Michaél J. Young

Addison-Wesley Publishing Company, Inc.
Reading, Massachusetts Menlo Park, California New York
Don Mills, Ontario Wokingham, England Amsterdam Bonn Sydney
Singapore Tokyo Madrid San Juan

Library of Congress Cataloging-in-Publication Data

Young, Michael J.
Software tools for OS/2 : creating dynamic link libraries / Michael J. Young.
p. cm.
Bibliography : p.
Includes index.
ISBN 0-201-51787-6
1. OS/2 (Computer operating system) I. Title.
QA76.76.063Y676 1989
005.4'469--dc20 89-34203
CIP

Production Editor: Amorette Pedersen
Cover Design by: Doliber Skeffington Design
Set in 11-point New Century Schoolbook by Benchmark Productions

ABCDEFGHIJ-AL-89
First Printing, July, 1989

INTRODUCTION

When OS/2 was introduced in 1987, it was accompanied by a myriad of new concepts and terms. One of the most prominent of these was the expression **dynamic-link library**.

Briefly, a dynamic-link library is a collection of subroutines stored in a disk file, which may be read into memory and called by application programs. The process of loading and accessing a dynamic-link library is known as **dynamic linking**. Traditionally, a subroutine is not stored in a separate disk file, but rather is incorporated directly into the executable file of the program that calls it.

Initially, the concept of dynamic linking may seem esoteric, and the distinction between a dynamic-link function and a normal subroutine may appear academic. However, after reading this book and after working with the operating system, you will realize that dynamic linking is one of the most important features of OS/2, and that dynamic-link libraries have far reaching and practical significance for both software developers and system users. Here are two primary reasons for the unique importance of dynamic linking.

First, the vast collection of services that the operating system provides for application programs (known collectively as the *Application Program Interface*) is implemented as a set of dynamic-link libraries. The dynamic-linking mechanism allows programs written in high-level languages to call these services using the standard function calling protocol. Thus, through dynamic linking, the facilities of the operating system are made readily accessible to programs, and form an integral part of OS/2 applica-

tions. It is important to understand dynamic-linking so that you can make optimal use of these services when writing application programs.

A second reason for the significance of dynamic linking is that you can package your own collections of subroutines as dynamic-link libraries. Whether you are developing a set of functions for use within your own applications, or you are preparing a function module to be sold commercially, a dynamic-link library provides a convenient and efficient vehicle for implementing your module.

The MS-DOS programming world has proven the importance of using packages of prewritten routines for developing application programs. Such function libraries allow the application programmer to focus on the main program logic, and eliminate the need to develop routines for tangential tasks such as managing windows, performing screen I/O, and handling indexed files. In fact, creating and distributing function libraries currently forms the basis for an entire software industry. The added complexity of OS/2 and the Presentation Manager will intensify the need for using prewritten function libraries when developing OS/2 applications.

Function libraries for OS/2 could be packaged in the same manner as those for MS-DOS (typically, either as source code or as binary code within standard object modules). Implementing subroutine packages as dynamic-link libraries, however, offers many advantages. For example, since dynamic-link libraries are stored in separate disk files, executable program files remain small and fast loading. Also, once a dynamic-link library has been read into memory, the code it contains can be shared by several simultaneous application programs; consequently, computer memory is conserved. Furthermore, a program can call a given dynamic-link library regardless of the language in which it is written; thus, dynamic-link libraries form truly generic software tools.

Finally, because functions included in dynamic-link libraries are called in the *same manner* as the basic services of the operating system, dynamic-link libraries can be used to form seamless operating system extensions. In fact, major operating system extensions, such as the Presentation Manager, are implemented as collections of dynamic-link libraries. You can also replace certain OS/2 services with your own

dynamic-link library routines. Specifically, you can replace many of the basic functions for managing the screen, keyboard, and mouse.

As you read this book, you will learn other advantages of dynamic linking, and should come to appreciate the flexibility and elegance of this mechanism.

This book will help you understand and appreciate many of the features of dynamic linking. More importantly, however, it is a practical handbook, written to show you how to *create* dynamic-link libraries. To date, the documentation on writing dynamic-link libraries is sketchy and spread out over many sources. This book gathers this diverse information into a single source, and offers many tips for avoiding problems and optimizing your use of the dynamic-linking mechanism. The book also provides many example listings; most of the discussions begin with concrete programming examples, and subsequently add theoretical and general information to deepen your understanding of the basic techniques.

Note finally that dynamic-link libraries developed according to the techniques given in this book can be used both by basic protected-mode programs and by Presentation Manager applications. Thus, whether you are developing a text-mode OS/2 program or a Presentation Manager application, you will find the programming methods presented in this book relevant to your work.

An Overview of the Book

The treatment of dynamic-link libraries in this book can be divided into three primary areas of emphasis: Chapters 1, 2, and 8 describe how to *use* dynamic-link libraries; Chapter 3 explains how dynamic-link libraries *work*, and the remaining chapters show how to *create* dynamic-link libraries.

Chapter 1 summarizes the techniques for writing a basic OS/2 protected-mode program, and Chapter 2 outlines the methods for developing an elementary Presentation Manager application. These two chapters serve to explain (or review) basic OS/2 programming techniques before the book embarks on the more advanced techniques required to develop dynamic-link libraries. These chapters also show how to call dynamic-link functions from the two primary types of OS/2 programs.

Chapter 3 explains how dynamic-link libraries work, and lays the theoretical groundwork for understanding the techniques presented in the remainder of the book.

Chapter 4 presents the techniques for writing a simple dynamic-link library, and provides a general overview of the entire dynamic-link library development process. Dynamic-linking, however, is a highly flexible mechanism that offers many options and variables. The techniques presented in Chapter 4 use only the simplest of these options; each of the remaining chapters in the book explores one or more of the advanced options.

Chapter 5 describes how to define both shared and non-shared data segments, so that a dynamic-link library can either share data among all programs that call the library, or provide data that is private to each program.

Chapter 6 shows how to create dynamic-link library initialization and termination routines. These routines are especially valuable for dynamic-link libraries that manage resources shared by several programs.

Chapter 7 describes the special versions of the C runtime library that support multiple-thread applications and dynamic-link libraries. It explains the steps that you must take to allow a program or dynamic-link library to use one of these libraries.

Chapter 8 shows how a program can explicitly load a selected dynamic-link library at runtime (normally all referenced dynamic-link libraries are automatically read into memory when the program is loaded).

Chapter 9 summarizes the steps for providing a real-mode version of your dynamic-link library, so that this library can be used by programs designed to run under either real or protected mode, which are known as **dual-mode** programs.

Almost all of the example listings given in chapters 1 through 9 are written in the C language. Chapter 10, however, describes the methods for writing a dynamic-link library in assembly language. This chapter also shows how to use assembly language to write dynamic-link functions that execute with **I/O privilege** (permission to use certain restricted machine instructions).

The Glossary at the end of the book defines many of the technical terms you may encounter while reading this book or other literature on OS/2.

Finally, the Bibliography cites a number of useful books on OS/2 programming, C and assembly language, and the architecture of the 80286 processor.

There are several special uses for dynamic-link libraries that are not covered in this book. For example, these libraries can be used to store OS/2 **resources** (a form of read-only data read stored within an executable file). Also, dynamic-link functions can be used to replace certain OS/2 services (see the documentation on the KbdRegister, MouRegister, and VioRegister OS/2 functions). The basic information presented in this book, however, should make it easy to employ dynamic-link libraries for these and other special uses you may encounter.

How to Use the Book

If you feel the need to review fundamental OS/2 and Presentation Manager programming techniques, you should begin by reading chapters 1 and 2. In all cases, you should read Chapters 3 and 4; these two chapters constitute the heart of the book. Chapter 3 describes the basic mechanisms underlying dynamic linking, and Chapter 4 presents the basic techniques for creating a dynamic-link library.

Once you have finished Chapters 3 and 4, you can read the remaining chapters in any order. Each of these chapters discusses one or more specific features of dynamic-link libraries, and you can select from among them according to your particular needs.

Finally, you should use the comprehensive glossary provided at the end of this book. The literature on OS/2 programming and the C language abounds with technical terms, newly coined expressions, and words used with special meanings. The book employs many of these terms without stopping to define them (or perhaps they are defined only the first time they appear). Accordingly, be sure to use the glossary if a term is unfamiliar, or if you are uncertain of the meaning of a word within a specific context.

Tools and Requirements

Developing OS/2 programs and dynamic-link libraries requires a large number of software tools. The examples in this book were written using

the Microsoft OS/2 Software Development Kit, which was a large (and expensive) collection of software tools supplied as a series of shipments, beginning with pre-release software and culminating with the final retail products. The required tools are now available as the following separate retail products: the Microsoft C compiler version 5.1 (essential), the Microsoft Macro Assembler version 5.1 (most of the examples in the book can be prepared *without* the assembler), the Microsoft Programmer's Toolkit version 1.1 (containing documentation on the OS/2 API functions, and a variety of optional utilities), and IBM OS/2 version 1.1 (the operating system itself, obviously required).

The programming examples in the book are based upon this collection of software tools. Accordingly, it will be easier to use the book if you have these specific tools (or *later* versions of these tools). However, you may be able to develop OS/2 programs and dynamic-link libraries using tools supplied by other vendors; you may also be able to use a high-level language other than C. In either of these cases, you can employ the basic concepts from this book but will need to translate the specific implementation details according to the tools you are using.

Note that the term "programmer's reference," which you will see many times in this book, is a general description referring to either of the following two specific reference books cited in the bibliography: the Microsoft *Programmer's Reference* (supplied with the Programmer's Toolkit) or the IBM *OS/2 Technical Reference*. These reference books fully document each of the operating system functions. The most important OS/2 functions used in this book are described in accompanying figures; however, for additional details on these functions, as well as descriptions of other functions, see either of these two reference books.

Also helpful for writing OS/2 programs are third-party function libraries and programmer's tools. See the Software Offer at the end of the book for a description of several such products I have developed. These products supply complete source code, and are thus valuable as learning resources as well as for facilitating program development.

Finally, this book assumes a working knowledge of C and assembly language, and of OS/2 programming basics. If you need further background knowledge in any of these areas, see the Bibliography for the titles of useful books.

TABLE OF CONTENTS

CHAPTER 1

MAKING PROTECTED-MODE PROGRAMS

This chapter explains the basic steps for writing protected-mode programs under OS/2. The applications described here are designed to be run from the OS/2 full-screen command prompt. Although many of these programs can also be run within a window of the Presentation Manager, they cannot use the full set of Presentation Manager features. Chapter 2 describes the basic methods for developing programs specifically for the Presentation Manager—these programs have free access to the facilities of this environment including its extensive set of graphics functions.

Both chapters serve several purposes. First, they provide a brief introduction to (or review of) basic OS/2 programming concepts that are important for understanding the more advanced techniques involved in developing dynamic-link libraries. Next they reveal the general context in which dynamic-link libraries are used. Dynamic-link libraries are not freestanding entities; they are software tools called by applications, and must closely integrate their activities with those of the calling process. Finally, the chapters show how to use dynamic-link libraries. There are few OS/2 applications that do not call dynamic-link library functions—the example programs in these chapters use the dynamic-linking mechanism to access the services of the operating system. (Subsequent chapters will explain how dynamic-link libraries work, how to create your

own dynamic-link libraries to provide custom extensions to the operating system services, and how to package your own collections of routines.)

In this chapter you will discover that you can easily use dynamic-link libraries without knowing how they work or how they are created. From the application programmer's vantage, using a dynamic-link library function is almost identical to calling a subroutine contained in a conventionally linked library; such as the C runtime library. The simplicity of using dynamic-link libraries adds to their importance as software tools; it also makes it possible to postpone the discussion of their inner workings until you have explored several examples of their use.

First you will see how to develop a simple protected-mode application—a variation on the archetypal "hello, world" C program; then add multitasking features to this example; and finally, how to coordinate the activities of the separate program tasks using interprocess communication. Consequently you will learn the fundamental steps to prepare any protected mode application and how to use several of the advanced features of OS/2. The chapter will emphasize the differences between developing protected-mode programs for OS/2 and developing real-mode programs for MS-DOS.

Developing a Simple Protected-Mode Program

Figure 1-1 lists a C source file destined to produce a simple protected-mode OS/2 program. This program repeatedly prints a message on the screen and terminates it when the user presses a key. The message consists of 11 lines form a box containing the string "Hello from main." The program pauses for 1/2 second between messages.

Like most C programs, this example contains calls to external functions (functions not defined within the source file), namely, **DosSleep**, **KbdCharIn**, **DosExit**, and **VioWrtTTY**. As you can see from the source code, these functions perform most of the work of the program. Although the program is written in the same manner as a typical MS-DOS application, there are two important features of these external function calls that are unique to OS/2.

Figure 1-1

```
/*
     Figure 1-1

     You can prepare this program using the following commands:

          cl /c /W2 /G2 /Zp FIG1_1.C
          link /NOI /NOD FIG1_1.OBJ,, NUL, SLIBCE.LIB OS2.LIB, FIG1_6.DEF
*/
#define INCL_DOS
#define INCL_SUB
#include <OS2.H>

void PrintMessage (void);              /* Prints a message on the screen.   */

void main (void)
     {
     KBDKEYINFO Key;                        /* Holds 'KbdCharIn' information*/

     for (;;)
          {
          PrintMessage ();                  /* Print a message on the screen*/

          DosSleep                          /* Create a delay.              */
               (500L);                      /* 1/2 second.                  */
```