Chapter I1
Introduction to MPX-PC

Revised: Jan. 10, 2011

I1.1 INTRODUCTION

This manual provides detailed instructions for a series of experimental projects to accompany the text *A Practical Approach to Operating Systems*. Each project module results in the development of a component of a simple MultiProgramming eXecutive, or MPX.

This version of MPX is designated "MPX-PC." It is designed to run on today's most common computer type, a personal computer or "PC" using the Intel/IBM architecture and running a current version of Microsoft Windows such as XP, Vista, or Windows 7.

There have been many approaches to the development of a project environment for studying operating systems. Several of these provide the complete source code of a full production-quality operating system for detailed study and possible modification. Such a package may run to hundreds of pages of program listing, and it is a daunting challenge to understand even a portion of it completely. The most important concepts may become lost in a host of details which are necessary in a realistic system, but which greatly obscure the essential structures. The project environment we provide here is designed to be straightforward and simple. MPX is not intended to be a complete and practical operating system. Instead, it is an experimental platform designed to allow the study of limited but realistic issues in the design of selected OS components. The philosophy of this approach is that you will learn best if you are challenged to implement your own operating system, even a simple one, rather than studying or enhancing an existing OS.

The projects defined in this manual are based on a model developed over many years on various computers at West Virginia University. It is a *building-block approach*. Although each project appears to be a standalone project, each will form an important component of the final complete MPX. Using the framework of an interactive command handler, which is the first module, each subsequent module can be independently tested as it is developed. Modules that your instructor does not assign for implementation can either be omitted entirely, or replaced with example implementations which we provide in the instructor's package. The final step in the approach is the integration of the modules into a fully-developed MultiProgramming eXecutive.

The popular "KISS" (Keep It Simple, Stupid) principle is applied for each project that you will implement. This is important, because the environment of a PC can be complex and very unforgiving (i.e., the system will "hang up" with little else for you to do but reboot the system).

By describing to you only the aspects of the hardware that are important for the project at hand, we have simplified the hardware environment so that you may focus on applying the particular OS principles being studied when you implement a particular project.

I1.2 YOUR PROJECT ENVIRONMENT

The essential components of an environment suitable for the development of MPX-PC are:

- A computer using a processor with the Intel 80x86 architecture and the IBM-PC memory and interrupt structure, a VGA or better display, and at least one USB port,
- A Microsoft MS-DOS or Windows family operating system,
- An ANSI C compiler. Borland Turbo C/C++, although old, is recommended for its extensions that provide direct access to hardware resources.

A printer is needed for some project requirements, although generally it may be shared. During final integration, it is desirable to have *both* a printer and a second PC connected via USB ports. These devices should be connected directly, not through a network.

The monitor is generally used in simple (monochrome) text mode, and no special graphics mode is assumed. Those with color monitors supporting EGA, VGA, etc., may choose to make use of these features, but this will limit the portability of your software to less capable environments.

MPX may be developed on any Microsoft Windows environment, but it generally operates in the "command line" environment. The services MPX uses from Windows are primarily those that comprise the MS-DOS environment on which the Windows command line is based.

The MPX-PC project is designed for development using the C language. C has achieved a unique status as a language that is widely available, fairly well standardized, and suitable for systems programming. The characteristics required for a good systems programming language (SIL) are not necessarily the same features which would be judged most important for higher-level programming. In particular, it may be necessary to give up some degree of safety features in order to achieve direct, efficient control of addressing, data representations, and specific hardware resources where required.

The project requires a C compiler supporting the standard C language (ANSI C). At the same time, it requires direct control of hardware resources, which is not provided by ANSI C. For this reason the Turbo C/C+ compiler (Version 2.0 or 3.0) by Borland is recommended. The Borland compilers are not the only ones that could be used, but we have found them to have some distinct advantages: They provide mechanisms for direct access to hardware resources including registers, and for convenient construction of interrupt handlers. You may use other compilers, but you may then need to make use of assembly language for low-level hardware access.

-

I1.3 OVERVIEW OF THE PROJECT

The sequence of project modules in MPX is designed to enable the construction of a basic operating system that is capable of managing a full range of resources and providing a full range of operating system services. Emphasis is placed on the development of a process management mechanism that supports scheduling and resource sharing for multiple interactive processes. Modules for device management, memory management, and file management are among the optional extensions for the project. In order to provide a means of managing and experimenting with this system as it evolves, MPX is provided from the start with an interactive, command-driven user interface.

Ideally, a complete MPX would provide all needed services and resource management to the user interface and other application software (processes). MPX in turn would interact directly with the hardware resources, and no additional support software would be required. This situation is illustrated in Figure I1-1. In this figure, MPX is the sole arbiter between the hardware and the application software, which consists of the command handler (COMHAN) and the ordinary processes.

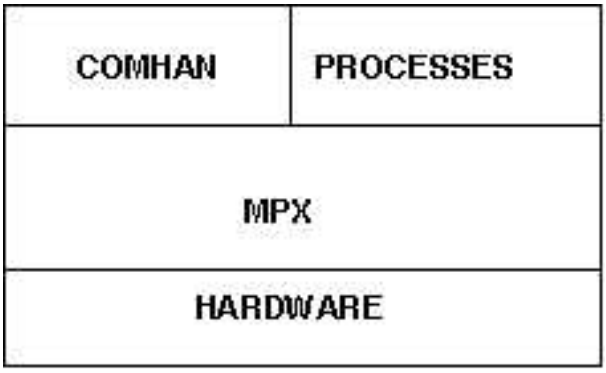

Figure I1-1: An Ideal MPX Environment

In reality, MPX must be viewed as an evolving operating system, and even when finished it will be incomplete. Additional support software is required to provide services that are not (yet) provided by MPX itself. The primary source for this extra support is the host operating system, Microsoft Windows. MPX will not attempt to replace Windows completely. Instead, it will provide a limited set of services, while calling on Windows (MS-DOS) for the additional services it needs.

In order to provide a uniform interface to the host system, we provide an additional software layer in the form of a library of support procedures. Your evolving MPX will generally not invoke Windows services directly. Instead, it will make use of support routines which in turn will obtain services from Windows. The purpose of this library is to provide MPX with a uniform support environment as the project evolves, and to minimize the differences between MPX-PC and other MPX versions that may operate in other environments.

A more accurate picture of the system structure during MPX development is given in Figure I1-2.

-

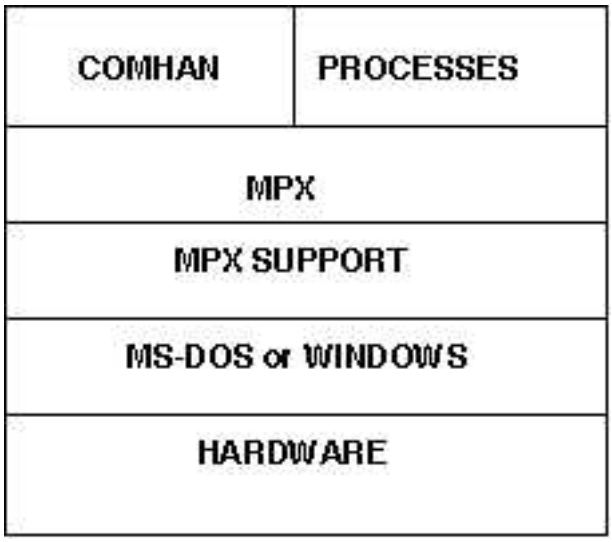

Figure I1-2: A Realistic MPX Environment

Note that MPX, MPX-support, and MS-DOS or Windows may be collectively viewed as the "system software." MS-DOS is part of your existing environment, and we provide the MPX support library. Your task is to develop selected components of MPX itself. The role of MPX will gradually expand during the project at the expense of the other components. For example, in the initial stages, COMHAN and the test processes will interact with the keyboard and display using MPX support routines and MS-DOS to control the terminal hardware. MPX will not be involved. At a later stage you may add a terminal driver to MPX that will take over the role first performed by MS-DOS.

COMHAN (the command handler), and the processes, form the application software that uses MPX. You will develop COMHAN during the first module. For much of the project, COMHAN will be treated as an integral component of MPX itself. During the final integration phase, however, COMHAN will become an independent process that must compete with all other processes for resource use and MPX services.

Test processes are provided for all of the modules that require them. In some cases, you may be able to write your own test processes if you choose.

The evolution of MPX will take place in three principal phases:

Phase one consists of five required modules. The first of these will construct a command handler and a small set of basic commands. The next three modules will lead to the development of a process management facility that can create and manipulate processes and PCBs, load program files, and schedule process execution in a round-robin fashion. The final required module implements one or more device drivers.

Phase two provides a series of optional modules. Your instructor may choose assignments from these options as time permits. Some of the options include a clock interrupt handler; a

-

timesharing scheduler; a message-passing mechanism; additional device drivers; a memory manager; and a simple file system.

Phase three is the final integration of your components, together with other components we may provide, into a functional multiprogramming executive. During this phase the command handler will become a true process, and an extended collection of processes must be scheduled concurrently while performing a variety of input and output tasks.

I1.4 THE MPX SUPPORT SOFTWARE

Support software for student use is provided on the website that accompanies this manual. A more comprehensive set of support software is included in the instructor's package. It would be almost impossible for you to implement *all* components of a complete MPX-PC in a one semester course. To simplify the task, a number of support procedures are provided which work with MS-DOS to provide basic support services. These include:

- Directory searching
- Date and time access
- Memory management
- Program loading
- Selected device drivers

Details of these support procedures are presented as needed along with the module definitions. All procedures are contained in the support module MPX_SUPT.C, which is provided.

The description of each support procedure includes a text summary, a prototype, description of the parameters, and a list of possible error codes returned. In a few cases, examples of usage are also given. NOTE: The list of error codes is an upper bound; in fact not every implementation of the support software will return all of these codes or detect with certainty all of the errors mentioned. A set of common definitions is also provided, including prototypes for the support procedures. These are contained in the header file MPX_SUPT.H. This file should normally be "included" by all of your source files.

Several project modules require the loading of executable program files by MPX processes. Test program files have been provided for this purpose. All of these files have the extension `.MPX`.

I1.5 ORGANIZATION OF THE PROJECT MANUAL

This project manual is divided into four parts. Part I, including chapters designated I (for Introduction) provides basic background information on the project concepts and the hardware and support software that you will use. Part 2, including chapters designated R (for required), presents the actual required project modules. Part 3 includes optional project modules in chapters designated O (for optional). Finally, Part 4 presents the Final module in a single chapter designated F.

-

The presentation of each project module, beginning in Chapter R1, follows a standard format. Each module description includes the following parts:

Introduction - a brief overview of the complete module.

Key Concepts - a discussion of important ideas that will be studied during the construction of this module.

Detailed Description - the project specifications are presented here. Included are the functional requirements, interface requirements including valid parameter values, constraints, error detection.

Support Software - the support software required in a particular module is described. Included in the description are the purpose and requirements for using the software.

Testing and Demonstration - the requirements for testing the software for this module are presented. In some cases, test programs are provided to you that may be used to validate the operation of projects.

Documentation Requirements - All modules have specific documentation requirements. These are based around the ongoing development of a *User's Manual* describing MPX usage, and a *Programmer's Manual* describing MPX structure.

Optional Features - This section may suggest some optional extensions to the basic module requirements that may be suitable for advanced classes.

Hints and Suggestions - This section offers hints and suggestions on how to proceed most effectively with each project. Here you will find such information as suggestions for data structures, how to handle certain error conditions, information on specific hardware features pertinent to the project, specific features or characteristics of Turbo C, and how to go about debugging the project.

I1.6 DOCUMENTATION

General Requirements

To receive complete credit, all of the software you develop for MPX must be properly documented. Your instructor will determine what portion of your total grade will be based on documentation. The documentation requirements we recommend for MPX contain three principal components: Source program listings with suitable comments; a *User's Manual;* and a *Programmer's Manual*. Each of these components should evolve in a natural way as the various MPX modules are developed.

-

Program Comments

Every source file which you develop for MPX should be documented with three principal types of comments:

- A descriptive header for the entire file
- A descriptive header for each function, procedure, or other significant element within the file
- Suitable comments within the body of each element

File headers may have a variety of attractive forms to aid visibility, such as boxing with asterisks, etc. The important thing is to follow a *consistent* format. We recommend that all file headers contain at least the following information:

- File name
- Author(s) name(s)
- Version information, including date last changed
- Identification of the procedures and other components the file contains

In addition, it is very desirable to include a "change log" or history information, summarizing the significant changes that have been made as the file was developed. In any case, it is *most important* that the date in the file header *always* correctly reflects the date of the most recent change. Time information could optionally be included here as well. If you are not careful about this, sooner or later, an old version will crop up to haunt you. You must control changes to your source code carefully, and the file date and change log are ways of helping you do this. The header for each procedure or function should include at least the following:

- Procedure Name
- Purpose
- List of parameters and returned value
- Procedures called, and global data accessed
- Summary of algorithm and any other significant notes

Dates of change and history information for individual procedures are optional but recommended. If different elements in a file had different authors, this should be noted as well.

In a similar manner, your data structures, symbolic names and variables should be carefully documented within the program.

Sufficient internal comments should appear within each procedure to make it easy to understand your program statements. A good plan might be to have at least one comment for every four or five source lines. These comments should not be whimsical or restate the obvious, but they should really add to the understanding of the code.

Comments should always aid in understanding the program *as it is now*. The central goal is to explain your program thoroughly to a competent programmer who never saw it before (and can't find you to ask questions). While it may occasionally be useful to "comment out" old statements rather than removing them totally, old comments must be brought up to date. Comments indicating which lines were changed by someone are also sometimes valuable, but these must not take the place of comments that aid understanding.

User's Manual

The MPX project will require a *User's Manual* as a principal part of its documentation. The purpose of this manual is to explain to users of MPX how to use all of its capabilities as well as possible. For our purposes the user can be viewed as a person at a terminal accessing MPX through its interactive command handler. The emphasis in the User's Manual is on "how to do it," not "how does it work."

Your *User's Manual* should contain at least the following sections:

- Table of contents
- Overview of COMHAN and MPX
- Summary of commands
- Detailed description of each command, including:
- Syntax
- Use
- Example
- Possible Errors
- Summary of Error Messages
- Index

Write the *User's Manual* so that *you* would be happy with it!

Clearly the *User's Manual* is meant to evolve as the various MPX modules are developed. This manual should be developed so that it can be easily modified.

Programmer's Manual

A *Programmer's Manual* is also required for the project. This manual is designed to provide all necessary information for programmers who may need to maintain or enhance your software. The Programmer's Manual describes the program structure and how it operates. It should include the following information:

- Table of Contents
- Overview of MPX and COMHAN identifying their principal structural elements
- Summary of contents of each file
- Summary of each function, including parameter specifications
- Identification of all data structures and variables
- Cross References: Where is each function called from? Where is each variable defined and used?
- Index

This manual is intended for technical staff who must maintain the MPX software. Ask yourself the question, "would I be happy if given the responsibility to maintain this program using this manual?"

Like the *User's Manual*, the *Programmer's Manual* will evolve over the course of the project, and should be developed in such a way that it can be easily updated.

-