Chapter R2
Process Management

Revised: Feb. 4, 2011

R2.1 INTRODUCTION

The process management module of MPX is designed to provide a basic framework for representing and manipulating processes. This is a required module which provides the first true services and data for MPX itself.

As discussed in the text, we represent processes by means of a **process control block (PCB)**. In this module you will define the necessary data structures to represent a collection of PCBs, together with a set of basic procedures to manipulate this collection. In addition, you will implement some new commands for the command handler to exercise these process management procedures (if you were careful with Module R1, adding commands should be easy!).

The process management module has three parts. The first part is the design of data structures to represent a PCB and to manage collections of related PCBs. In this part you will also design a structure for process queues. Part 2 requires the implementation of a set of procedures to allocate and initialize PCBs, to search for a specific PCB based on its content, and to manipulate the process queues. The final part calls for the addition of a series of command procedures to COMHAN. These commands will be used to invoke and exercise the control procedures of Part 2; in addition, commands will be needed to display the contents of a specified PCB or set of PCBs.

Although Module R2 defines a framework for process management, the "processes" you manipulate in the module will have no program to execute and will never actually be dispatched. However, this module will provide the basic mechanisms that will be used to implement true executable processes in future modules.

R2.2 KEY CONCEPTS

Process Management Data Structures

As discussed in the text, processes must be represented by descriptive data structures called *Process Control Blocks (PCBs)*. In addition, a great deal of process management involves the manipulation of *queues* which represent processes waiting for various types of service. This section discusses some general considerations for these structures.

Since a PCB contains a collection of information of varying type relating to a process, it logically takes the form of a *record* or *structure* in a high-level programming language such as C. There must be one PCB for each process. Operating systems that do not support the dynamic creation of processes will have a fixed number of processes and may manage these processes using a fixed array of PCBs. If processes can be created and destroyed, some type of dynamic allocation of PCBs will be necessary.

The textbook outlines the general types of information that must be maintained in a PCB. Each item is accessed by one or more process management operations. These operations will determine the most suitable representation for each item. For example, the process state may be represented explicitly by an integer, or explicitly by associating the process with a specific queue. The state could also be represented by a character-string name, but this would be inappropriate since it would greatly complicate the PCB operations that access the state information. More detailed considerations for the PCB content and its representation in MPX are contained in the next section.

The second key data structure for process management is the queue. Logically, a queue is a sequence of elements. The usual definition of a queue is a first-in, first-out sequence in which elements are added only at the tail (or end), and removed only from the head (or beginning). In operating system queues, however, it is often necessary to support other sequences, and to insert elements at arbitrary positions in the queue. Removal from arbitrary positions is less common, but sometimes necessary. In addition, the entire queue may need to be searched to determine if a specific element is present.

In many cases, the elements in the queue will be PCBs themselves (probably represented by identifiers or pointers). However, queues with other element types may sometimes be needed. For example, a request for a service such as input or output may be represented by a special request record, and queues of such records may be formed to wait for the appropriate I/O device. A queue is usually implemented either as an array of elements managed in a circular fashion, or as a linked list. The operating system requirements, including the unpredictability of queue size, suggest that a linked list implementation is preferable. In most cases, linked list elements will be dynamically allocated (although possibly from a private pool of some fixed size). If the queue elements are PCBs, one alternative is to assign a permanent pointer field in each PCB for certain classes of queues (*e.g*. process state queues), and to construct a queue by chaining the selected PCBs directly using these pointers.

R2.3 DETAILED DESCRIPTION

The Command Handler

The first requirement for this module is the definition of a set of data types and structures for managing processes. An essential goal is to define a PCB data type and create a PCB collection. The PCBs themselves will each have the form of a C structure, which should be predefined using the *typedef* facility. They may be either statically or dynamically allocated. In either case you will need a representation which allows you to perform the following operations: (i) find or allocate a free PCB; (ii) release a PCB that is no longer needed; (iii) determine the status (free or not) of a specific PCB; and (iv) systematically examine all PCBs which are currently in use for search or display purposes.

-

In addition to the PCBs themselves, a set of queues must be defined in this part. Processes in certain states will have their PCBs linked in appropriate queues.

We will not prescribe the exact structures to be used for allocation and general management of your PCBs. Your instructor may give you more specific instructions. In developing these structures you should consider the following issues:

1. PCBs in MPX must be allocated and deallocated, but you may choose between allocation from a fixed pool (in the form of an array), or direct dynamic allocation from the system heap. A fixed pool may be more efficient but places a greater burden of management on the process manager. Some operating systems are restricted by design to a fixed memory area; in this case the fixed pool is the only possible method.

2. If you choose a fixed pool, will you represent all PCBs as an array? How large will the set of PCBs be? (MPX requires at least 10.) What is the best way to systematically traverse the entire collection? How will you distinguish free PCBs from available PCBs?

3. If you choose heap allocation, how will the PCBs be linked? Will you actually free the storage for unused PCBs, or will you retain a pool for reuse? Will you enforce a maximum number of PCBs to control heap space? How will you deal with allocation failures, which are always possible?

4. Remember that your project must be compiled using the *large memory model,* and that all addresses should be *far pointers*, which should be allocated a 32-bit longword for storage. The compiler will automatically use the far pointer representation for any pointers you declare. If a pointer is converted to an integer, the type should be *unsigned long*. If you are allocating stack space for MPX data items, every address requires four bytes.

Regardless of the strategy for management of PCBs, there are some specific requirements for the PCB content. Each PCB structure in MPX-PC must include fields to represent at least the following information for a process:

Process Name: a character array that can contain at least 8 characters (plus a trailing null).

Process class: a code identifying the process as an application process or a system process. This may be an integer code or an enumeration type. If an integer is used, it should be specified as a named constant using the *#define* facility.

Priority: an integer of appropriate size representing priority values, which will be in the range -128 through +127.

State: A representation for the process state, in two forms. One part should be a code indicating the current state directly. The second part should be a provision for linking this PCB on a doubly-linked queue associated with the current state. The minimum set of states to be represented includes *running*, *ready*, *blocked*,

-

suspended-blocked, and *suspended-ready*. The state code may be an enumeration type or an integer, as discussed above. You may find it useful to separate the state into two parts: a code distinguishing the three states *running*, *ready,* and *blocked,* and a separate flag indicating whether the process is *suspended*. You may choose to use direct linking or separate queue elements to implement the state queues. If direct linking is used, then each PCB will need to contain a pair of pointers to other PCBs in its current queue. If separate elements are used, then no special fields are needed.

Stack area: A local stack area is required for each process in MPX-PC. This stack area must be "large enough" to handle the worst-case requirements of each process, since the 8086 architecture includes no provisions for handling stack overflow. We recommend a size of at least 1K (1024) bytes. The stack may be actually contained in the PCB, or it may be separately allocated, with an appropriate descriptor in the PCB itself. Note that the stack area serves also as the context save area. In addition to the stack itself, the PCB should contain a pointer to the current "top" of the stack. Remember that in the 8086 stacks grow from high addresses to low addresses. Thus the pointer should be initialized to the *highest* address in the stack area, *plus one*, and the current top address will always be the *lowest* address presently occupied.

Memory description: This information describes the memory to be used for the actual instruction code and static data for the process. It consists of an integer giving the memory size, and two addresses: a load address indicating the start of the memory area into which the process should be loaded, and an execution address indicating the address at which execution should begin.

Not all of these fields will be actually used in Module R2. However, they will all play a role in future modules. In addition, you may find it necessary to add additional fields to your PCBs as your project evolves.

Process queues logically consist of a linked sequence of PCBs. For this module you must define a *ready queue* and a *blocked queue*. These queues should contain all processes that are presently in the *ready* state and the *blocked* state, respectively. You may choose to define separate queues for the *suspended-ready* and *suspended-blocked* states, or you may maintain these processes on the same queue as their non-suspended counterparts.

Pointers to both the head and tail of each queue should be statically defined in your program. Each pair of pointers should form a single structure called a *queue descriptor*. If you have chosen to implement queues using direct linking, then each pointer in the descriptor will point directly to a PCB. Otherwise, you will need to define a queue element type, which includes pointers to forward and backward queue elements *and* a pointer to a PCB. You will also need a mechanism for statically or dynamically allocating queue elements. In this case the descriptor will contain pointers to queue elements.

In MPX, the ready queue(s) will always be maintained in priority order. Negative priority values indicate low priority, and positive values indicate high priority. The PCBs for the highest

-

priority processes should be closest to the head of the queue. By contrast, the waiting queue(s) do not consider priority; they are maintained in first-in, first-out (FIFO) order.

PCB Management Procedures

Once the structure of your PCBs is determined, you must implement a set of procedures to manipulate the PCB collection you have defined. The general description of these procedures is given in the text. Some further specific descriptions, are given here.

Note that each of these routines does a single job, but some of them must typically be used together. For example, `Allocate-PCB` would normally be followed by `Setup-PCB`, and operations which affect the PCB contents will often be accompanied by queueing changes using `Insert-PCB` and `Remove-PCB`.

We will not specify exact prototypes for these procedures, since some details will depend on your particular choice of PCB and queue representations and management strategies. However, the following descriptions can be used in a straightforward way to define the required prototypes, which should be documented in your programmer's manual.

Allocate-PCB

This procedure should find or allocate an available PCB. mark it as in use, and return a PCB pointer or identifier. This identifier will be used as an input parameter to identify this PCB in other procedures. No parameters are required for `Allocate-PCB`. A null pointer should be returned if no PCB can be allocated.

Free-PCB

This procedure should release an allocated PCB. The PCB identifier is the only required parameter. The procedure should ensure that the specified PCB identifier is valid, and return an error indication in case of any problem.

You should define suitable error codes to be returned by this and other procedures in abnormal situations. We recommend a code of zero to indicate success, and small negative integers to indicate specific errors. This practice is followed in the MPX support procedures, as described for Module R1.

Setup-PCB

This procedure initializes the content of a PCB, which is assumed to be newly allocated using `Allocate-PCB`. Input parameters should be defined to specify initial values for all PCB fields for which the desired initialization may vary; suitable default values must be used for all remaining fields.

In Module R2, the process state should be initialized to *ready (not suspended)*; In later modules a different initial state may be required. The stack should be initialized by setting the stack pointer to the highest value in the stack. We further recommend, to aid debugging, that all locations in the stack itself be initialized to zero.

-

Actual processes are not loaded in Module R2, so the memory area is not used. The memory size should be set to zero, and the related pointers to null.

The identifier of the PCB to be setup is also provided as an input parameter. This identifier should be tested for validity. Be sure also to check that the other input parameters are valid and reasonable, and that the given process name is unique. An appropriate error code should be returned if these requirements are not met.

Find-PCB

This procedure searches the PCBs for a process having a specified name. We assume process names are unique, so the first such process may be accepted. The search should cover all PCBs currently in use, regardless of the process state.

The character string name should be passed to this process as an input parameter. The return value is an identifier for the PCB found, or a null pointer if the search failed, just as for `Allocate-PCB`.

Insert-PCB

This procedure inserts a PCB into a specified queue. The queue is specified by an identifier that could, for example, be the address of its descriptor. Both the PCB identifier and queue identifier are passed as input parameters. An additional parameter is needed to indicate whether insertion is to be by priority order or by FIFO order; both methods must be supported. Either an enumeration type or an integer code may be used. As usual the validity of all parameters should be checked, and an appropriate code should be returned indicating the success or failure of the insertion.

Remove-PCB

This procedure removes a PCB from a specified queue. As in the previous procedure, the PCB identifier and queue identifier are provided as parameters. No additional parameters are required. Suitable checks should be made, and a success or error code should be returned.

PCB Management and Display Commands

In this final part of the process management module you will add commands to your command handler to manipulate PCBs and display their contents. These commands will make use of the data structures and procedures defined in the previous parts. Some of these commands will play an important role in later versions of MPX. Others are intended for temporary use in testing your basic process management. These commands will not be needed later.

This section describes the required command operations to be added. As in Module 1, each description includes *suggestions* for command and argument definitions. You may modify these suggestions to suit your own command format. However, all of the specified functionality should be included in your system.

-

The *first four* commands described below are temporary; they will not be used except in the testing and demonstration of Module R2. The remaining commands form a permanent part of your MPX system.

Create PCB

This command should allocate and setup a new PCB. The arguments to be specified include process name, process class (system or application), and priority. By default the process is initially in the ready (not suspended) state.

An appropriate error message should be displayed if the arguments are not valid or if no PCB can be allocated.

Delete PCB

This command should deallocate an existing PCB. Depending on your allocation strategy, the PCB may be marked as free or its storage may be actually released. The only argument is the process name. An error message should be given if the specified process does not exist.

Block

This command should place a specified process in a blocked state; its suspended status should not be changed. The process should be removed from the ready queue, if necessary, and inserted in the blocked queue. The only argument is the process name. An error message should be given if the specified process does not exist.

Unblock

This command should place a specified process in a ready state; its suspended status should not be changed. The process should be removed from the blocked queue, if necessary, and inserted in the ready queue. The only argument is the process name. An error message should be given if the specified process does not exist.

Suspend

This command should place a specified process in a suspended state. The state chosen will be either *suspended-ready* or *suspended-blocked*, depending on its previous state. If suspended processes are kept on separate queues from non-suspended ones, then this operation may require a queue transfer. The only argument is the process name. An error message should be given if the specified process does not exist.

Resume

This command should place a specified process in a non-suspended state. The state chosen will be either *ready* or *blocked*, depending on its previous state. If suspended processes are kept on separate queues from non-suspended ones, then this operation may require a queue transfer. The only argument is the process name. An error message should be given if the specified process does not exist.

-

Set Priority

This command is used to change the priority of a specified process. The command arguments are the process name and the desired priority value. The procedure should ensure that the process exists, and that the desired priority is within the acceptable range for the process class. If the process is in a ready state, then the change of priority may require a repositioning of the process in its queue. This can be accomplished by removing and reinserting the PCB in priority order.

Show PCB

This command is used to display all information contained in a single PCB for a process specified by name. The process name is the only argument. If the process exists, the command results in the display of process information on the screen. You should define an attractive display format for the process information. This display should present all of the significant information in your PCB structure. If it is possible that the display will not fit entirely on a single display screen, then you must provide a mechanism for paging as discussed for the *Help* command of Module 1.

Show All

This command is used to display information about all PCBs that are currently in use. The PCBs may be listed in any convenient order.

In this and the following commands that display information about multiple PCBs, the information you display may be limited or extensive. The required items for the *`Show All`* command are the process name and state (including its suspended status). Other information is optional. You must continue to ensure that all information fits on a single screen, or that multiple screens can be handled.

Show Ready

This command is used to display information about all processes that are currently in the *ready* state or the *suspended-ready* state. The display should show information for all non-suspended processes, followed by all suspended ones. Within each group, the processes should be presented in priority order, that is, the same order in which they appear in the queue. Besides the process name, this display should include at least the priority value and suspended status.

Show Blocked

This command is used to display information about all processes which are currently in the *blocked* state or the *suspended-blocked* state. The display should show information for all non-suspended processes, followed by all suspended ones. Within each group, the processes should be presented in the same order in which they appear in the queue.

Besides the process name, this display should include at least the suspended status.

-

R2.4 SUPPORT SOFTWARE

Module R2 requires only limited support procedures. These procedures are a subset of those described for Module R1. Of the ten procedures outlined for Module R1, only `sys_alloc_mem` and `sys_free_mem` may be used directly by Module R2 procedures. These procedures *should* be used exclusively if dynamic memory allocation is performed.

Since Module R2 is "added on" to Module R1, however, all of the other support procedures will continue to be used by the original commands of Module R1.

As before, all support procedures are found in the supplied library file `MPX_SUPT.C`, and their defining prototypes, along with other necessary definitions, are in the file `MPX_SUPT.H`. See the previous chapter for details.

R2.5 TESTING AND DEMONSTRATION

The process management facilities of Module R2 may be tested and demonstrated using the commands developed for this module. The *Show* commands may be used to verify that each operation performed by the other commands had the intended effect.

You should prepare a test plan which exercises all of the new commands and management procedures. Your plan should include the following elements:

- Create several PCBs including both system and application processes with a variety of priority values. Use the *Show All* and *Show PCB* commands to verify that all PCB contents are correct, and that PCBs can be located by process name.
- Use the *Show Ready* command to ensure that all processes are linked on the proper queues in the proper order.
- Change the state of various processes using the *Block*, *Unblock*, *Suspend* and *Resume* commands. Verify these changes using *Show All*, *Show Ready* and *Show Blocked*.
- Change the priority of some ready processes using *Set Priority*. Include priority values at the extreme ends of the valid ranges. Ensure that invalid priorities are rejected. Verify that these processes are properly ordered in the ready queue.
- Delete some PCBs using *Delete PCB*, and create some new ones. Verify these changes using *Show All*.
- Delete *all* remaining PCBs. Ensure that all *Show* commands produce an empty display.

R2.6 DOCUMENTATION

Add documentation for the *permanent* process management commands to the *User's Manual*. You should not include the temporary commands in the manual; instead they should be documented in a short report to be submitted separately.

The *Programmer's Manual* should be updated to include all of the data structures and procedures specified in Section R2.3, and any additional data structures and procedures you developed.

R2.7 OPTIONAL FEATURES

Additional commands and procedures may be defined to make other meaningful changes to the content of PCBs. An example would be a *Rename* command to change the name of a process. Further variations on the *Show* commands could conveniently display information for other process groups. Examples may include application processes, system processes, suspended processes, or non-suspended processes.

R2.8 HINTS AND SUGGESTIONS

Consider carefully the tradeoffs in deciding between pool-based or heap-based PCB allocation. Note that if the total size of each PCB is large (including the stack), it may not be possible to fit all PCBs in a single data segment.

When you are fine tuning your queue structure, consider all operations that need to be performed on queues. How easy is it to:

- Search a queue, either from front to back or from back to front?
- Perform insertions in priority order?
- Remove a PCB cleanly from a queue, including the release of any storage that was used for the queue element?
- Determine which queue, if any, a given PCB is in?
- Transfer a PCB from one queue to another, or to a different position in the same queue?

All of these are operations you will need to perform in this module as well as future modules.

As far as possible, you should think of your PCBs as a set of objects of an abstract data type, which should be manipulated only by the procedures you develop in this module. You may add further procedures to this set to avoid any direct manipulation of PCBs by higher-level software.

Although some of the commands for this module will not be used again in the same form, variations on these commands may appear again in later modules. For example, the operation of deleting a PCB will become one step in the complete termination of an active process. You may want to structure your code in a way that makes it easy to adapt and reuse such procedures in the future.

-