Module R2

CS450

Next Week

R1 is due next Friday

Bring manuals in a binder - make sure to have a cover page with group number, module, and date. You do not need to turn in your source code.

Project demonstrations during lecture

Demonstration options:

- Bring a laptop with your code and compiler (preferable), demonstrate in class
- Bring your source code, project files and executable on flash-drive, email, etc and we will test on my computer.

If your project requires an environment (compiler/OS) not supported(Windows XP and the version of Turbo C on my website), you must bring in a laptop with the appropriate environment.

- Talk to me **before next Friday** if you have any concerns about where your project will be tested.
- It is your responsibility to verify that your project works on whatever computer you are going to demonstrate on before coming to class.

At **EACH** team member needs to be present for the demonstration, or have a valid excuse that has been approved **before next Friday.**

If you need help, please come to my office hours, email me to set up an appointment, or email me your code and I will **try** to help remotely.

Introduction: R2

In Module R2, you will add process management functions to your system. You will implement a structures to hold information about processes and queues to hold lists of processes

You will also implement functions to create, delete, insert, remove, and modify processes.

You will add commands to your system to utilize and demonstrate this functionality.

Module R2 is basically all of the “back-end” code you will need to run processes in later modules.

R2 Overview

- Due Friday 10/8
- Structures
  - Process Control Block (PCB)
  - Queues
- Procedures
  - Allocate-PCB
  - Free-PCB
  - Setup-PCB
  - Find-PCB
  - Insert-PCB
  - Remove-PCB

- Commands
  - Create PCB*
  - Delete PCB*
  - Block*
  - Unblock*
  - Suspend
  - Resume
  - Set Priority
  - Show PCB
  - Show All
  - Show Ready
  - Show Blocked

*Temporary Command

Review: Two Possible Queue Implementations

There are two primary ways to implement queues.

As a linked list

Each element in the queue will contain a pointer to the previous and next element in the queue.

As a linked list of queue descriptors

A queue descriptor structure is created and contains

- Pointer to the queue element
- Pointer to the next queue descriptor in the linked list
- Pointer to the previous queue descriptor in the linked list

Process Control Block (PCB)

The MPX system will need to keep track of processes that are in the system, so you will need to create a structure to hold the information MPX will need

Information that needs to be maintained is:

Process Name

Process Class

Priority

State

Process Stack Information

Process Memory Information

(Possibly) Pointers to other PCBs- for use in the queues.

PCB: Attributes

Process Name

Character array of at least 8 characters + a null terminator (‘\0’)

Pay special attention that *each* PCB is allocated its own character array- be careful of this if you define your character array as a character pointer (char *array) rather than statically (char array[10])

Process Class

Code identifying the process as Application or System

Could use an int, char, etc. to represent this

Tip: Use #define to define constants for each class (ex: #define SYSTEM 1), and use the symbolic constant, rather than the numerical value, whenever you need to refer to the class- this makes your code easier to follow and debug

Priority

integer that will contain values of -128 through +127

-128 is lowest possible priority, +127 the highest possible priority

PCB: Attributes

State

There are two “states” that must be maintained in the PCB

- Is the process running, ready, or blocked?
- Is the process suspended or not suspended?

There are two options to how you could do this

- Maintain two separate state indicators
  - A state flag indicating whether the process is running, ready, or blocked
  - A suspended flag indicating whether the process is suspended or not
- Maintain a single state indicator
  - Possible states are running, ready, blocked, suspended ready, suspended blocked

You should store the states in the same manor as the process class- use an integer or character type and then define a symbolic constant for each value

PCB: Attributes

Stack Area

Each process requires a local stack area of at least 1024 bytes

You need to maintain two pointers- to the stack top and to the stack base

- Should be of type unsigned char*

Memory Descriptors

Memory Size

- Integer

Pointers to load address and execution address

- Should be of type unsigned char*

These won't be used until R3 and R4, but it's a good idea to implement them now

Queue Pointers

Depending on how you implement your queues, you may need to include as attributes pointers to other PCB's in the linked list. Most likely, you would need two pointers- one each to the previous and next PCB's in the queue.

If you implement your queues using queue descriptors, you will not need queue pointers in your PCB.

Queues

Two options

2 queues: ready and blocked

- Ready queue will hold all processes in ready state, regardless of whether a process is suspended or not suspended
- Blocked queue will hold all process in the blocked state, regardless of whether a process is suspended or not suspended
- Use this option if you have 3 PCB states and a separate suspended flag

4 queues: ready, suspended ready, blocked, suspended blocked

- Use this option if you have 5 PCB states- suspended and non suspended processes will be in separate queues

Queues Continued

You will need to create a structure for the queue

Suggested attributes:

Number of nodes in the queue

Pointer to the head of the queue

Pointer to the tail of the queue

- The head and tail pointers will either be to the PCBs themselves or to queue descriptors, depending on how you choose to implement your queues.

Ready queue(s) will be maintained in priority order

When you insert a PCB into the ready queue, insert it in order, with the highest positive priority at the front of the queue

Blocked queue(s) will be maintained in FIFO (first in, first out) order.

When inserting into the blocked queue, insert at the tail.

Allocate_PCB

Allocates the memory for a new PCB

Takes no parameters

Use the sys_alloc_mem support function to allocate memory for the PCB. You need to allocate the number of bytes equal to the size of the PCB struct you defined (for example, sizeof(PCB) would give you the number of bytes needed to store a PCB).

Returns a pointer to the new PCB, or NULL if an error occurs.

Free_PCB

Frees all memory associated with a PCB, including the PCB itself

Parameter is a PCB Pointer

Uses sys_free_mem to free all memory associated with the PCB, including the stack and the PCB itself

To free the stack, you need to free the memory associated with the base pointer

Can have either no return, or return an error code

Setup_PCB

Initializes a PCB's content (name, priority, class)

Parameters: name, priority, class

Should check that the parameters are valid

Name is unique

Priority between -128 and +127

Class is valid

Calls the Allocate_PCB function to allocate the memory for a new PCB

Sets the name, priority, and class based on the parameters

Sets state to ready, not suspended (this will change in later modules)

Sets remaining fields to defaults

Does not insert the PCB into a queue

Returns PCB Pointer if successful, NULL if not successful (including if one of the parameters are not valid)

Creating the Local PCB Stack

You can do this in either the Allocate_PCB function or the Setup_PCB function

You should allocate a stack size of at least 1024 bytes

Define the stack size as a symbolic constant

Set the stack base pointer equal to the address returned by sys_alloc_mem

Set the stack top pointer

Stack_top = stack_base + stack_size

Find_PCB

Searches all queues for PCB with given name

Parameters: process name

Should search all queues and all states to find the PCB with that name

Returns a pointer to the given PCB if found, NULL if not found

It is important that this function returns NULL if no PCB with the given name is found. This is an easy way to perform checks on the uniqueness of a process name!

This is a very important function- you will use it throughout your code.

Insert_PCB

Parameters:

Queue- which queue to insert into (symbolic constant)

- This parameter could be eliminated by having your function look at the state of the process to determine the queue

PCB* - pointer to PCB to be inserted

Inserts the given PCB into the specified queue

If inserting into a ready queue, insert the PCB in priority order

If inserting into a blocked queue, insert the PCB in FIFO order (simply put the new PCB at the tail of the queue)

Can either return nothing, or an error code

Remove_PCB

Removes the PCB from the queue it is currently in

Parameters: PCB pointer

This function should determine which queue the PCB is in based upon the PCB's state, it should not need to take the queue in as a parameter

Should not free the memory associated with the PCB

Can either return nothing, or an error code

Example: Lifetime of a process

1) MPX recognizes that a process needs to be created- calls *Setup_PCB* and passes the process name, class, and priority
2) *Setup_PCB* calls *Find_PCB* to verify that no process with the given name already exists and checks the other parameters
3) *Setup_PCB* calls *Allocate_PCB*, which returns a pointer to a new PCB
4) *Setup_PCB* allocates the PCB's local stack and initializes all PCB attributes, then returns a pointer to the PCB to the calling function
5) MPX calls *Insert_PCB* to insert the PCB to the appropriate queue.
6) The process runs for a while before MPX recognizes that the process has either terminated by itself or the user has requested to terminate it. MPX calls *Remove_PCB* to remove the PCB from whatever queue it is in.
7) MPX calls *Free_PCB* to free the memory associated with that PCB, including the PCB's local stack and the PCB itself

Commands

There are a number of permanent and temporary user commands that you must implement

You will probably have to write some sort of function in your command handler to “handle” that command- to ask for or interpret any necessary arguments, check their validity, and then call the appropriate R2 function to handle the request

You can use whatever syntax for these commands you want, but try to choose something intuitive. Whatever you choose, you must include all commands in the help summary as well as have help for each command individually.

Temporary Commands

Create PCB

Allocates and sets up a new PCB and inserts it into the ready queue

Needs to get the following from the user:

- Process name
- Class
- Priority

Must check the validity of the above and display an appropriate error message if one is invalid

- Process name must be unique
- Class must be valid (make sure to specify in the help and temporary command manual how to enter the class)
- Priority must be -128 to +127

Calls the setup_PCB function to setup the PCB, then call the insert_PCB function to insert it in the ready queue

Temporary Commands

Delete PCB

Removes an existing PCB from its queue and frees its memory

Get the PCB name from the user

- Check that the PCB exists

Calls the remove_PCB function to remove the PCB from its queue, and then calls free_PCB to free the PCB’s memory.

Display success or failure message

Temporary Commands

Block

Get PCB name from the user- check that the PCB with that name exists

Places the process (PCB) in the blocked state

Does not change its suspended status

Should remove the process from the ready queue (by calling remove_PCB) and place the process in the blocked queue (by calling insert_PCB)

Display appropriate error or success message

Unblock

Get process name from the user

Places the process (PCB) in the ready state

Does not change its suspended status

Should remove the process from the blocked queue and place the process in the ready queue, if necessary

Display appropriate error or success message

Note: in later modules, block and unblock will not be called by users, but the system will need to block and unblock processes themselves. Design your code in such a way that it will be easy for the system to block and unblock processes in code.

Permanent Commands

Suspend

Get PCB name from the user

Puts the PCB in the suspended state

Might require changing queues (from ready to suspended ready, for example) if 4 queues are used

Display appropriate error or success message.

Resume

Puts process in not suspended state

Might require changing queues if 4 queues are used

Display appropriate error or success message.

Permanent Commands

Set priority

Get PCB name and new priority from user

- Check that the PCB exists
- Check that the new priority is valid (-128 to 127)

If the PCB is in the ready state, you will need to change the position of the PCB in the queue based upon its new priority

Display appropriate error or success message.

Permanent Commands

Show PCB

Get PCB name from the user

Display all information about the PCB (except pointers) in “attractive” format

Display error if PCB doesn’t exist

Show all

Show information for all PCB’s in all queues, in any convenient order

At a minimum, must show process name, state, suspended status, and priority.

Limit information about a single PCB to a single line

Must fit on single screen, or paging must be used

Permanent Commands

Show ready

Show all process in the ready state

Show same info in same format as the show all command

Must show the processes in the order they exist in the queue

Show blocked

Same as Show ready, except show the processes in the blocked state

Requirements and Tips

Requirements:

Check the validity of ALL inputs, make sure your MPX system can handle anything the user may choose to type

Display non-cryptic error messages when errors occur (such as searching for a PCB that does not exist or creating a PCB with a name that is already used).

All memory from all PCB's in all queues and all PCB stacks must be reclaimed when the program is terminated.

Must add help entries for all new commands

Tips and Hints:

Create an error handler

- Function that prints out error messages
- Define a number of error codes using #define, and pass the error constant to the handler when an error occurs (ex. handle_error(PCB_DOESNT_EXIST)

Create R2-Init and R2-Clean functions

- R2-Init: initialize queues
- R2-Clean: clear queues, reclaim all memory allocated

Don't rewrite code- use the functions you already wrote!

Create separate source file for R2

CHECK FOR MEMORY LEAKS!

Documentation

Programmer & User Manuals

Add on to what you did already.

Temporary Commands

Include in your user and programmer manuals where appropriate.

Start your Temporary Manual now but simply taking those commands and putting them in here and later you can remove them from your other manuals.

Manuals due at the time of your demonstration