CS 421 - Operating Systems

OS Simulator - Assignment 2: PCBs, basic execution

Note: The entire OS Simulator project is heavily influenced by the "Project Manual to Accompany "A Practical Approach to Operating Systems"" by Malcolm G. Lane and James D. Mooney as well as the paper "Operating Systems I: An Operating System Simulation Developed as Homework" by Ronald Marsh.

Dr. Mooney was one of my professors at WVU. In the late 80's, he co-authored a book on operating systems. It came with a project manual and some accompanying software. The book and manual are now out of print. While some text from the projects will be taken directly from the manual, I have heavily modified the project to be more modern and open-ended.

Task 1: Create some way to represent the maximum amount of memory your system has. This can be an integer for now. You can either prompt the user for this information when your simulator first starts or have it stored in a file for easy modification later.

Task 2: Create a data structure to represent a Process Control Block. PCBs have the following information:

A. Process Identification Number (PID) - integer that uniquely identifies a process
B. CPU Usage Term - holds how many time cycles a process was in the CPU - starts at 0
C. I/O Request Term - holds how many time cycles a process was waiting for I/O - starts at 0
D. Waiting Term - holds how many time cycles a process was waiting for the CPU - starts at 0
E. Memory - how much memory is needed for the process

Task 3: Create Ready and Blocked queues that hold PCBs. Create a CPU that can only hold one PCB. Create an I/O event queue (more information on I/O events later)

Task 4: Create user commands for the PCBs

A. CreatePCB
    a. Allocates and sets up a new PCB and inserts it into the Ready Queue
    b. Need to get the following info from the user:
        i. Process ID
        ii. Amount of memory needed
    c. Must check the validity of the Process ID (must be unique) and the amount of memory (can't be bigger than the maximum memory in your system) Must give appropriate error messages.
B. DeletePCB
    a. Removes an existing PCB from its queue and frees its memory.
    b. Get the PCB name from the user
    c. Display success or failure message
C. Block
    a. Get PCB name from user
    b. Place the PCB in the Blocked queue
    c. Display success or failure message
D. Unblock
    a. Get PCB name from user
    b. Place PCB in the Ready queue
    c. Display success or failure message

E. Show PCB
    a. Get PCB name from the user
    b. Display all information about the PCB in an “attractive” format.
    c. Display error if PCB doesn’t exist.
F. Show all
    a. Show information for all PCBs in all queues, in order of PID.
    b. Limit information about single PCB to a single line
    c. Must fit on a single screen, or use a mechanism that allows the user to continue on once the screen is full (scrolling would be good here)
G. Show ready
    a. Show PCBs in the ready state
    b. Show same info in same format as the show all command
    c. Must show PCBs in the order they exist in the queue
H. Show blocked
    a. Same as show ready, except show the processes in the blocked state.
I. Generate Random PCBs
    a. Prompt the user for a number of PCBs to generate
    b. Create the PCBs by randomly assigning them PIDs and memory.
    c. Insert PCBs into the ready queue.
J. Execute
    a. Move the PCB at the front of the ready queue into the CPU. To simulate processing time, use the random number generator to select a value from 0 to 10,000.
    b. Add that number to the CPU Usage Term. Use that value to update the Waiting Term in other processes.
    c. For every ten time cycles, generate a random number between 0 and 10.
        i. If the number is 4, add an user input I/O event to the event queue with a timecycle stamp.
        ii. If the number is 9, add a hard drive I/O event to the event queue with a timecycle stamp.
    d. Generate a random number between 0 and 3 inclusive to determine what happens with the process.
        i. 0 - process terminates and is removed from the system
        ii. 1 - process returns to the ready queue to wait its turn
        iii. 2 - process requires I/O and goes into the blocked queue wanting an user I/O request.
        iv. 3 - process requires I/O and goes into the blocked queue wanting a hard drive I/O request.
    e. If the process terminates, print out the PID, CPU Usage, I/O Usage, and Waiting values.
    f. Go through the blocked queue and see if the event queue satisfies any of the waiting processes. If so, move them to the ready queue. An event cannot satisfy a process request unless it happened after the process entered the blocked queue. Update the I/O usage time with how long it took the process to get the appropriate I/O. While going through the queue, you can throw away events that cannot possibly satisfy the PCBs in the blocked queue. Time spent waiting for I/O should also be factored into the Waiting value.
    g. Repeat until there are no processes left.

Find a way to display a trace of your execution process nicely in a file. It should be clear and easy to understand.

Documentation and Code Organization

Use the same code standards that were discussed in the previous assignment. Add all of the new functionality to your User's Manual and Technical Support Manual.