CSUSM

Program 1 Report

CS 433 Operating Systems

Adriana Caetano
9-23-2020

CS 433 Operating Systems
Professor: Xiaoyu Zhang
Assignment 1
Date: 09/23/2020

Assignment 1 Overview

Implement data structures in operating systems:

- PCB: Create a data structure to represent each process in the system. It should have at least an ID and a state.
- PCB Table: Create a container to hold all PCB elements currently in the system.
- ReadyQueue: Create a queue of processes that are in the READY state. It should run the highest priority process first, and it should have at least the following member functions:
  - add(): add a PCB element into the ReadyQueue.
  - removeHighest(): remove and return the PCB with the highest priority from the queue.
  - size(): return the number of elements in the queue.
  - display(): display IDs and priorities of processes in the queue.

Summary

On this first assignment, the goal is to simulate how the CPU handles and executes processes based on its priority.
To test the program, 2 tests are executed. In the first test, a PCB table is created with objects that act like processes, each PCB element has its unique ID and a state of priority. The ReadyQueue receives the PCB elements that are READY and choose the element with the highest priority to run next. The second test evaluates the performance of the implementation by looping one million times to add and remove PCB elements to the ReadyQueue. The time to run all iterations should be measured and presented with a result report of how many times each process is added and removed from the queue.

Design Decisions

I used the starter code from the programming assignment on the server. I also referred to the programming code on ReadyQueue by Sean Payne from March 5th, 2009 that I found on the web. I used cplusplus.com and cppreference.com to search for useful algorithms and their implementation.

The ReadyQueue is the class that controls the flow of the processes that will execute next by the system. I could have used a linked list, an array, a binary tree, or a heap to implement the ReadyQueue. In this assignment, I chose a linked list for the ReadyQueue and tested the performance.

The ReadyQueue is a linked list of PCB processes collected from a PCBTable. The PCB struct creates the PCB process containing its ID, priority value, state, and variable counters to hold how many times it was added and removed from the ReadyQueue. The state value of the process is of type enumerator ProcState. The PCBTable class contains an array container that holds PCB objects and some methods to get the process, check its state, and display the table. The ReadyQueue class creates a linked list that points to the processes on the PCBTable. It has methods to create new nodes, add and remove them from the list, and display them, and two methods to perform the tests 1 and 2 required for this assignment. So, the main function only contains a ReadyQueue object that calls for the functions performTest1 and performTest2, and all the computation is hidden from the user.

Instructions

To run this program, you need to login at cs433.cs.csusm.edu server. All the files are included on the caeta001 folder. Type “make” to build the object, and then “./prog1” to run it.

Results

Running test 1 I was able to include the specified processes to the queue and remove them one by one. I still have some bugs on the code that does not insert the process in a decrescent order, so not always the head is the highest priority. On test 2 my program took about 3

seconds to iterate one million times randomly adding and removing processes to the list. In the end, I print the time spent on this last process and a summary with all the processes and how many times it was added and removed. The final values for added and removed seems low considering it iterated a million times, but I could not find why to fix it.

```
 Test 2 runtime = 0.0299773 seconds

PCB Table Summary:
PCB ID: 1 - Priority: 38 - State: READY - Added 44 times - Removed 43 times
PCB ID: 2 - Priority: 13 - State: READY - Added 5 times - Removed 4 times
PCB ID: 3 - Priority: 41 - State: READY - Added 7 times - Removed 6 times
PCB ID: 4 - Priority: 34 - State: READY - Added 9 times - Removed 8 times
PCB ID: 5 - Priority: 25 - State: READY - Added 15 times - Removed 14 times
PCB ID: 6 - Priority: 5 - State: READY - Added 2 times - Removed 1 times
PCB ID: 7 - Priority: 14 - State: READY - Added 9 times - Removed 8 times
PCB ID: 8 - Priority: 20 - State: READY - Added 190 times - Removed 189 times
PCB ID: 9 - Priority: 32 - State: READY - Added 4 times - Removed 3 times
PCB ID: 10 - Priority: 14 - State: READY - Added 3 times - Removed 2 times
PCB ID: 11 - Priority: 31 - State: READY - Added 7 times - Removed 6 times
PCB ID: 12 - Priority: 24 - State: READY - Added 59 times - Removed 58 times
PCB ID: 13 - Priority: 9 - State: READY - Added 2 times - Removed 1 times
PCB ID: 14 - Priority: 42 - State: READY - Added 3 times - Removed 2 times
PCB ID: 15 - Priority: 42 - State: READY - Added 23 times - Removed 22 times
PCB ID: 16 - Priority: 23 - State: READY - Added 2 times - Removed 1 times
PCB ID: 17 - Priority: 38 - State: READY - Added 14 times - Removed 13 times
PCB ID: 18 - Priority: 45 - State: RUNNING - Added 16082 times - Removed 16082 times
PCB ID: 19 - Priority: 39 - State: READY - Added 3 times - Removed 2 times
PCB ID: 20 - Priority: 14 - State: READY - Added 8 times - Removed 7 times
PCB ID: 21 - Priority: 3 - State: RUNNING - Added 15984 times - Removed 15984 times
PCB ID: 22 - Priority: 35 - State: READY - Added 29 times - Removed 28 times
PCB ID: 23 - Priority: 22 - State: READY - Added 4 times - Removed 3 times
PCB ID: 24 - Priority: 28 - State: READY - Added 13 times - Removed 12 times
PCB ID: 25 - Priority: 23 - State: READY - Added 6 times - Removed 5 times
PCB ID: 26 - Priority: 14 - State: READY - Added 6 times - Removed 5 times
PCB ID: 27 - Priority: 31 - State: READY - Added 4 times - Removed 3 times
PCB ID: 28 - Priority: 40 - State: READY - Added 331 times - Removed 330 times
PCB ID: 29 - Priority: 17 - State: READY - Added 7 times - Removed 6 times
PCB ID: 30 - Priority: 39 - State: READY - Added 31 times - Removed 30 times
```

Challenges

My personal biggest challenge was to remember how to write the code on C++. The syntax of language and the use of pointers was hard to remember and implement, causing many compile errors and segmentation faults. And this led to an unexpected increase in time to complete the program for this assignment, resulting on a lot more time than I predicted.

Additional Comments

The biggest lesson I take from this first assignment is to start earlier on the following programs, so I have time to ask for help during office hours. It would also be good to find a partner for the next assignments, so I would have someone to work with and to help debugging. It would be nice if the professor could intermediate introductions between students that have the same level of knowledge and commitment with this class that might be a good fit, promoting partnerships for the next assignments.

Future Improvements

I would like to try different containers to hold the ReadyQueue and test their performance to compare which one would be more efficient for this program. I would also like to improve the quality of my program writing more concise functions. Some functions still need review on the logic to create a proper list based on the priority.

Submitted Files

A list of the submitted files and a summary of its contents:

Makefile

It is a file containing a list of instructions that are used by make utility to compile, link and build an executable file from the source code. For this assignment I used the starter code, including the PCB.cpp file to the source code list.

prog1

Executable file for this program.

main.cpp

Is the source code file that contains the main function. First, it prints out the header of the program with some basic information about the assignment and a brief description of the program. Then a ReadyQueue objected is created calling the ReadyQueue constructor to perform test 1 and test 2. After the computation it exits with return zero.

PCB.h

The header file that contains:

1. Enumerator class ProcState:
   - Fixed process state values: NEW, READY, RUNNING, WAITING, TERMINATED
2. Struct PCB:
   - A user defined data type to represent each process
   - Member variables: unique ID, priority value, process state, and counter variables to hold how many times the process was added and removed from the ReadyQueue.
   - Void displayPCB(): a void return type function that prints the basic process information: ID and priority value.
   - Void displayFullPCB(): a void return type function that prints the process complete information. It calls displayPCB() to print the process basic information, and complements it printing the process state, and how many times it was added and removed from the ReadyQueue.
3. Class PCBTable.
   - Creates a table of PCB objects.
   - Default constructor: creates a default size container of 30 processes.
   - Parameterized Constructor: takes a size attribute to build a user defined size container of process objects.
   - Destructor: Deletes the container.
   - Void randomize(bool randomize): iterates through the PCBTable to assign an unique ID for each process incrementing their id value one by one. If attribute randomize is set to false, priority values are the same as id. If it is true, assign random priority values from 1 to 50 for all processes.
   - PCB* getProcess(unsigned int processID): return a process pointer to a process on the table given an attribute processID.
   - Bool checkState(unsigned int processID, ProcState state): return a bool value after checking of the attribute process ID has the attribute state.
   - Void displayTablePCB(): iterate through the table calling displayPCB() for each process to print the basic information of the processes on the PCBTable.

- Void displayFullTable(): iterate through the table calling displayFullPCB() for each process to print the full information of each process on the PCBTable.

PCB.cpp

The source code file that contains the implementation of the functions declared on the PCB.h file.

ReadyQueue.h

The header file that contains:

1. Struct Node:
   - Member variables: PCB pointer to a process, node pointer to the next node
   - Default constructor: create a new node with null pointers
   - Parameterized constructor: takes a pointer to a process as an attribute and assigns it to the PCB pointer, next is a nullptr.
2. Class ReadyQueue:
   - Member variables: pointers to head and tail of the list.
   - ReadyQueue (): Default constructor. Assign nullptr to head and tail of the list.
   - ~ReadyQueue (): Destructor that deletes list.
   - Bool isEmpty (): returns true if head is a nullptr. Returns false otherwise.
   - Node* createNode (PCB (&process)): receives an address attribute of a process and returns a node pointer to this process.
   - Void add (Node* node): adds the node pointer attribute to the list. It first checks if the process is in the list by checking its state. If not in the list, it changes the process state to Ready, increments its added counter, and inserts the process in the list accordingly to its priority values using the helper functions insertFront, insertAfter, or insertEnd. Highest priority value is added at the head of the list, and lowest priority level is added at the end of the list.
   - Void insertFront (Node* newNode): adds the node pointer attribute at the beginning of the list.
   - Void insertAfter (Node* prevNode, Node* newNode): insert the newNode pointer attribute after the prevNode pointer attribute in the list.

- Void insertEnd (Node* newNode): adds the node pointer attribute at th end of the list.
- Void include (unsigned int elements[], unsigned int elemSize, PCBTable* table): include specific processes into the list. The first argument is an array of processID that needs to be included, the second argument is this array size, and the third argument is a pointer to a table that holds the processes. It iterates through the array calling the functions createNode() and add() for each process.
- Void removeHighest(): removes the first element of the list. It increments the remove counter and changes the process state. If list is empty, writes a message warning the user.
- Int getSize (): returns the size of the table after counting how many nodes are in the list.
- Void display (): display ReadyQueue content with basic information of each process.
- Void displayFull (): display ReadyQueue content with complete information of each process.
- Void performTest1 (): using helper functions from PCBTable class and ReadyQueue class, it creates a table of 30 process with consecutive priority values. It adds and removes specific processes from the list displaying the list after each removal.
- Void performTest2 (): using helper functions from PCBTable class and ReadyQueue class, it creates an initial list of processes. Then, it evaluates the performance of this ReadyQueue by measuring the time to loop one million times randomly adding and removing random processes from the list. Finally, it displays the results of the test and a summary of the processes.

ReadyQueue.cpp

The source code file that contains the implementation of the functions declared on the ReadyQueue.h file.