CSEN 601 project report

Team 81:

Hamza Hatem (55-1937) T07
Marly Maged (55-2587) T12
Mikhail George (55-1895) T18
Moaaz Samir (55-2296) T18

Supervised by:

Dr. Eng. Catherine M. Elias

May 2024

1 Introduction

The code for this project is written in C. It is meant to simulate an operating system. This is done by implementing the round robin (RR) scheduling algorithm to schedule the processes in the system, creating an interpreter that reads the txt files and executes their code, implementing a memory and save the processes in it, implementing mutexes that ensure mutual exclusion over the critical resource.

2 Milestone 1 Recap

In the previous milestone, the goal was to create 4 threads each with multiple print statements to be able to test the effect of different predefined scheduling algorithms on their execution. At first, we attempted to force the system to use our chosen scheduling algorithm (FIFO, RR or OTHER) but failed. This happened due to that being in a multi-core environment, scheduling algorithms can not be forced - each CPU core ends up handling a thread. There was no need for scheduling or queuing. Later, the solution was to force the laptop, through code, to use a single core. The threads were forced to use the chosen scheduling algorithm. Another issue that arose was that the effect of scheduling was not obvious when all threads were doing equal work that took about the same time. To solve this issue, thread 3 was edited to do more work/take more time.

3 Methodology

There are three given programs. They are read from the txt files and written into the memory. The memory is represented as an array of 60 addresses where each address holds a name and a value in the form name:value. For example, the first line of code is represented as instruction1:LINE OF CODE. The name represents the instruction number and the value represents the actual instruction.

Due to the fact that the number of programs is known, manual allocation is used. The memory is divided equally across the 3 programs - 20 addresses each. The issue with manual allocation is that there will exist empty spaces that will remain empty.

Within the 20 addresses, the first 6 are allocated to the process control block (PCB) of the program. The PCB is a structure that holds: Process ID, Process State, Current Priority, Program Counter and Memory Boundaries (Lower and Upper Bounds of the process' space in the memory - allocated addresses). The next 3 addresses hold program variables. Then, the lines of code are read line by line and written into the memory (7 addresses for programs 1&2 each and 9 addresses for program 3).

Upon the arrival of process 1, the PCB is created. Then, program 1 is read from the txt file line by line and written into the memory. It is then added to the ready queue. When it is time to execute this process, based on the value of the PC in the PCB, the instruction is fetched and decoded. Based on the decoding, values may need to be extracted from the memory (for example: print x, you'd need to retrieve x).

Regarding scheduling, a custom non-preemptive round robin algorithm should be implemented and used. Round Robin (RR) is quantum-based where a process that is executing runs for this set amount of time (clock cycles). Quantum is taken as user input. The choice of which process to run is based on which is at the beginning of the queue. If a process runs and by the time it is done, another process has arrived, the arriving program is added to the queue first.

For example: Process 1 needs 7 clock cycles and the quantum is 2 clock cycles. Let's say it is done with the first run at clock cycle 2 and process 2 arrives, the arriving process (process 2) goes into the queue first.

Multiple exclusion (MUTEX) allows threads to share resources without accessing the same resources simultaneously. This is done through semWait and semSignal functions. If a thread wants a resource, it has to check if it is available (semWait). If it is, it gets to use the resource and let go of the resource when it is done (semSignal). If the resource was not available in the first place, the thread is blocked and queued till the resource is available again. To ensure that the thread requesting semSignal is the same one that is already using the resource, the ID of the thread is kept track of and checked.

4 Discussion

Since dealing with strings in C is different than in Java, ***string.h*** library is used. This bridges the gap between string manipulation in C and in Java. There are multiple functions provided by this library. To copy the value of a string into a variable, ***strcpy*** is used. To compare between two strings, ***strcmp*** is used. To duplicate the value of a string, ***strdup*** is used. To check the length of a string, ***strlen*** is used. To split the string using a delimiter and get a pointer to the first token, ***strtok*** is used.

Since C doesn't have queues, we had to implement our own queue structure for this project. They are needed for scheduling. When processes are first created, they are added to the ready queue. If the resource they are trying to access is unavailable, they are added to the blocked queue.

5 Code explanation

In this section, we will explain all the structures and functions used in the code.

Structure (word): This is the structure we made to define the shape of one cell in memory. It has 2 string variables in it (name and value) and the structure of each cell has the shape name : value.

Structure (Pcb): This is the structure that is used by the operating system to schedule the processes. This structure has 6 variables in it: id of the process, state of the process, priority of the process, program counter, the lower boundary of the address space of the project and the higher boundary of the address space of the process. Those variables are saved in memory that means if I want to access them, I can access them only through looping on the memory.

Structure (memory): This is the structure of the memory and it is an array of 60 elements simulating 60 memory addresses where each element is a (word) that has the shape that we talked about before in the word struct.

Structure (programVar): This is a structure we use to for the scheduling and it holds variables that I need to access while executing (execution time of process, quantum).

Structure (queue): We have to use queues in the round-robin and for each mutex there is a blocked queue so we had to implement a data structure to simulate the function of a queue because there is no queue in c.

Function (queueEmpty): This function checks if the queue given as input to the function is empty or not.

Function (queuefull): This function checks if the queue given as input to the function is full or not based on the value of end in the queue struct.

Function (enqueue): This function adds elements to end of the queue.

Function (dequeue): This function removes elements from the front of the queue.

Function (printQueue): This function prints the elements of the queue. We will use it to print the final answer and we will also use it in debugging.

Structure (mutex): This structure defines our mutex which we will use to mutual exclusion between the resources in this project. Each mutex has 2 variables (a queue for the process that gets blocked and an enum that determines if the resource is locked or not).

Function (semWaitB): This function executes the wait function on mutex. This function locks the resource by a process and it can't get unlocked by any other process and if any process tried to unlock it, it will get blocked.

Function (semSignalB): This function executes the signal function on mutex. This function is responsible for unlocking the resource for other processes to use after the one that was using it is done with it.

Function (readValueFromMemory): This function is used to read a value from memory.

Function (readnameFromMemory): This function is used to read a name from memory.

Function (writevalueintoMemory): This function is used to write a value to memory.

Function (writenameintoMemory): This function is used to write a name to memory.

Function (printFullMemory): This function prints the memory contents.

Function (intializeMemory): This function initializes the memory to empty:empty.

Function (readInstructionsFromTxtfile): This function reads the instructions in the text file and writes it into the memory.

Function (updatePcbmem): This function gets the values of the PCB from the PCB struct and puts them in the right position in memory according to the process it belongs to.

Function (createPcb): This function creates the PCB for the new process.

Function (createProcess): This function creates a process by calling createPcb and readInstructions-FromTxtfile.

Function (executeOneLine): This function checks the PC value, fetches the instruction from the memory, decodes and executes it.

Function (getPc): This function returns the PC value based on the process.

Function (stringSplit): This function takes a string and returns the words in it.

Function (runAll): This function creates processes and enqueues them in the ready queue.This function simulates the work done by the scheduler.