Operating System Simulation

Team 19

May 20, 2024

1 Team Members

- Youssef Mostafa Faizallah, T-22, 55-2952
- Ahmed Samy, T-23, 55-4895
- Peter Adel Makram , T-8, 55-4343
- Youssef Raafat, T-22, 55-5066
- Abdullah Hesham, T-22, 55-5012
- Seif Mohamed, T-6, 55-2821

2 Purpose

Our goal is to simulate an operating system's process and the execution of different instructions. A normal operating system manages the memory allocation for processes, the scheduling for the processes, and synchronization for the use of shared resources between the processes. We will simulate this operation on three programs as provided by the project's description.

3 Introduction

3.1 Structures

An operating system manages the processes using a Process Control Block (PCB), which stores different data for the process like its state, ID, and so on.It also stores the code and variables for the process inside the main memory within the process Address Space. Also, it manages how they're scheduled for concurrent execution, and synchronizes the use of shared resources between them. We will need to implement some data structures for each.

3.1.1 Process Control Block

The PCB data structure contains information about the process, so each process has it's own PCB to store its data.

- Process ID
- Process State
- Program Counter
- Memory Boundaries (Lower and Upper Bounds of the process' space in the memory)
- Variable Address

3.1.2 Memory Word

The memory word data structure is a structure that holds two variables in the form of the pair (Name, Value) that represent the name of the variable/instruction/data and the value.

3.1.3 Queue

Multiple queue data structure is created, and utilized so that we can implement the ready queue to store the processes with state ready which indicates that they are ready for execution. Also, we use the queues to implement the blocked queue of each shared resource to store the blocked processes to know which process is waiting for which resource to acquire. Therefore, the use of a Queue data structure and its signature methods like enqueue and dequeue is essential.

3.2 Main Memory

Main memory is an array of memory words used to store the PCB, code, variables for the process within its address space.

3.3 Mutexes

Three mutexes of type "bool" are used for each resource, so that we can control the blocking and unblocking of every process in each resource.

3.4 Queues

Three blocking queues are used for every shared resource to store the blocked processes for each resource and one general blocked queue. Also, one ready queue is used to store the ready processes.

4 Methodology

4.1 Initialization

Firstly, the method initializeMemory() fills the main memory with empty memory words so that all of the memory cells are filled with no data.
Then, the method initializeQueue() is used so that all 5 queues we use in the program are initialized.

During initialization of our program, the user is asked to input the quantum of execution for the processes execution scheduler, and the processes' arrival time . We assume a process arrives at $t \equiv 0$ since our clock cycle is only incremented by the execution of an instruction. Once the clock cycle is equivalent to the arrival time of a process, it's read into memory, and a PCB structure is created with its parameters. The instructions in the file are loaded within the memory along side the PCB of that process inside its address space. Moreover, space for 3 variables for each process is left.

4.2 Memory Allocation

Assuming our program is made for only 3 processes, the process address space is divided as follows :

- Process 1 : (0-19)
- Process 2 : (20-39)
- Process 3 : (40-59)

This is done by calculating the lower address limit, and the upper address limit for this process using it's ID. Since we have only 3 processes, we use process IDs 0, 1 and 2.

The lower limit is calculated as : $processID * 20$
The upper limit is calculated as : $((processID + 1) * 20) - 1$

Then for each individual process, its 20 respective addresses are divided as follows :

- PCB Data : First 5 addresses
- Variables : The following 3 addresses
- Code : The remaining 12 addresses

This is done by initializing the program counter to be equivalent to $lowerBound + 8$, and the variable address to be $lowerBound + 5$

4.3 Scheduler

The scheduler methods implements the round robin algorithm, it dequeues a process from the ready queue and executes the instruction pointed by the PC inside the PC of the calling process if and only if the process is not blocked and the quantum is not equal zero. Also checks if the process has ended execution if so the PCB of this process is removed from the ready queue then it is freed from memory.

4.4 Execution Stage

Firstly, we load the instruction corresponding to the PC value stored inside the PCB then we use a pre-defined method "sscanf" that splits the instruction to words determined by the spaces between the words in the instruction, so the first parameter(command) will always be the name of the functionality the instruction wants to implement. According to the command, the program chooses what functionality to implement and the other parameters define the variables to be placed inside the 3 empty locations we have left inside the PCB. Finally, the PC is incremented inside the PCB and the clock cycle is incremented.

4.5 Mutex Operations

All three mutexes for shared resources are initialized to be false, then whenever a process requests a shared resource, wait is called, and the mutex boolean is set to true if the lock is obtained. Otherwise, if another process already has the lock, the process requesting the shared resource is blocked and preempted from the ready queue. Then, after the process with the lock is done with the shared resource, a signal is called which removes the blocked process from the blocked queue and adds it back to the ready queue.

5 Results

We're going to discuss some issues we faced during the implementation of this operating system, a long with the output after successful completion.

5.1 Memory Division

First, one of the issues we faced was dividing the address space appropriately for each process. Since we needed to include the PCB, variables, and the code for each process. That's why we came up with the division scheme that's explained in the Memory Allocation subsection in the Methodology section.

5.2 Pass By Reference

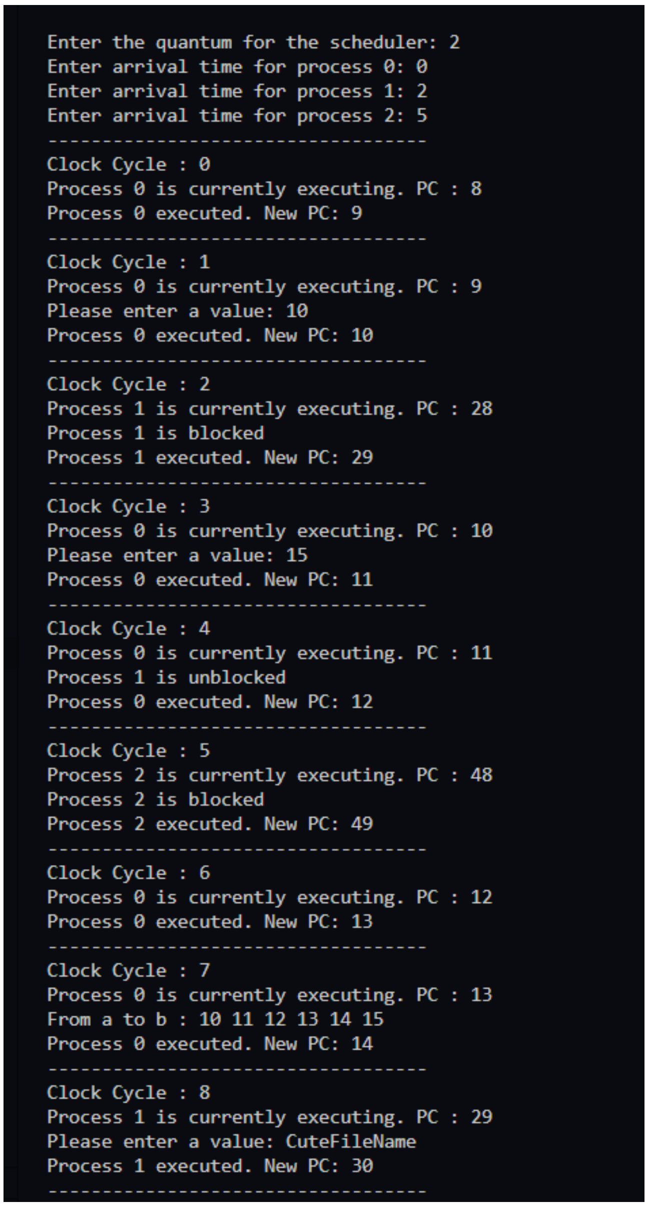
We faced multiple issues initially when attempting to modify data in structures used for the code. This was because we were using pass by value, which created a copy of the structure and the differences made within the function, were not referenced back to the original object. We fixed this by utilizing pointers and making the queues of type PCB*.

5.3 Arrival of New Processes

In round robin, during clock cycle t, if a process arrives, it's supposed to start executing immediately. However, initially this was not satisfied since the runScheduler method used to enqueue the preempted process to the ready queue, before the newly arriving process was inserted to the ready queue, and due to the expected FIFO behavior of the queue, the process preempted initially was dequeued first. We had to fix this by creating an algorithm that moves the last element inserted to the queue to be promoted to the start, so that it is dequeued first. We utilized an additional queue to store temporary PCB's.

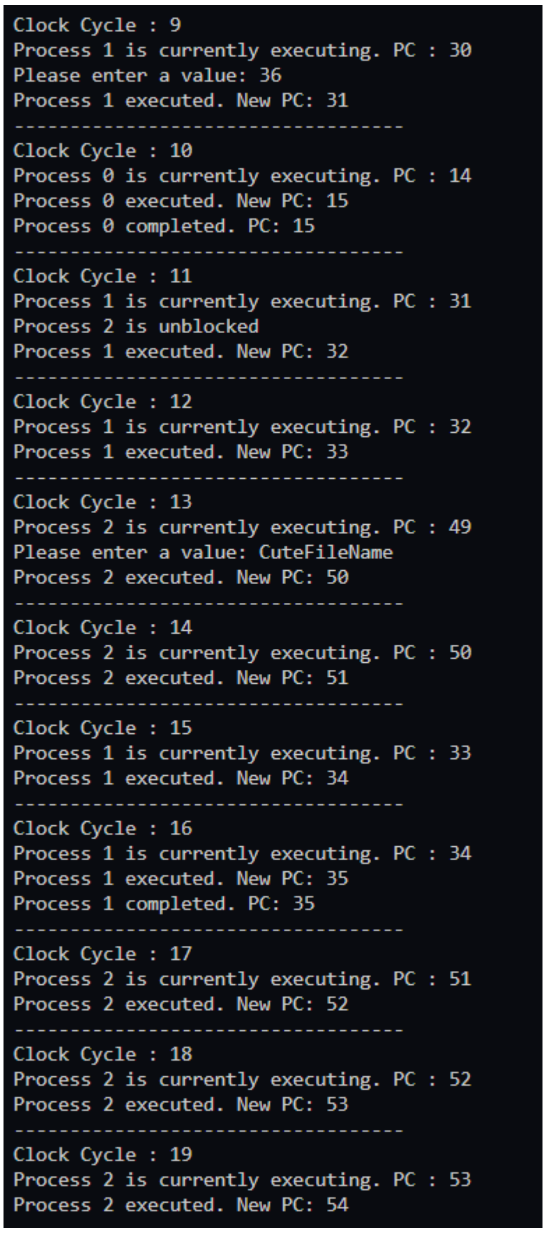
5.4 Program Results

We are using a quantum of 2 for the round robin scheduling, and the process 0 arrives at $t \equiv 0$, process 1 arrives at $t \equiv 2$, and process 2 arrives at $t \equiv 5$.



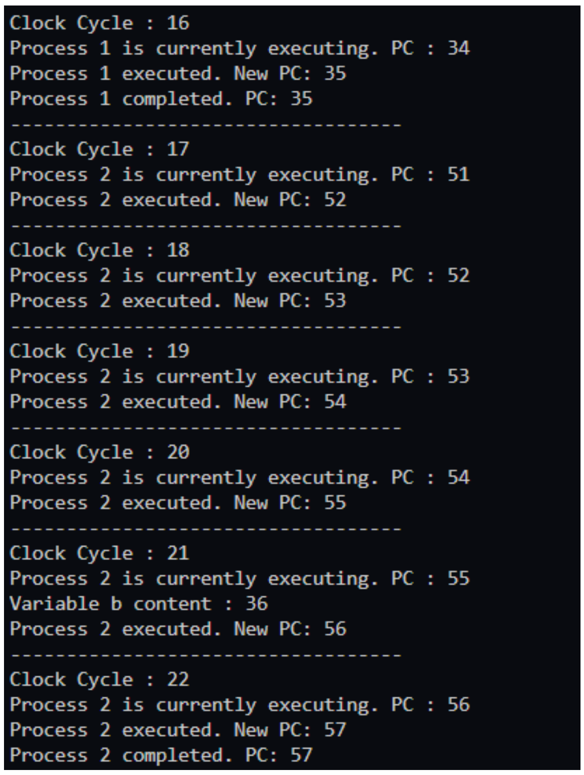
```
Enter the quantum for the scheduler: 2
Enter arrival time for process 0: 0
Enter arrival time for process 1: 2
Enter arrival time for process 2: 5
-----------------------------------
Clock Cycle : 0
Process 0 is currently executing. PC : 8
Process 0 executed. New PC: 9
-----------------------------------
Clock Cycle : 1
Process 0 is currently executing. PC : 9
Please enter a value: 10
Process 0 executed. New PC: 10
-----------------------------------
Clock Cycle : 2
Process 1 is currently executing. PC : 28
Process 1 is blocked
Process 1 executed. New PC: 29
-----------------------------------
Clock Cycle : 3
Process 0 is currently executing. PC : 10
Please enter a value: 15
Process 0 executed. New PC: 11
-----------------------------------
Clock Cycle : 4
Process 0 is currently executing. PC : 11
Process 1 is unblocked
Process 0 executed. New PC: 12
-----------------------------------
Clock Cycle : 5
Process 2 is currently executing. PC : 48
Process 2 is blocked
Process 2 executed. New PC: 49
-----------------------------------
Clock Cycle : 6
Process 0 is currently executing. PC : 12
Process 0 executed. New PC: 13
-----------------------------------
Clock Cycle : 7
Process 0 is currently executing. PC : 13
From a to b : 10 11 12 13 14 15
Process 0 executed. New PC: 14
-----------------------------------
Clock Cycle : 8
Process 1 is currently executing. PC : 29
Please enter a value: CuteFileName
Process 1 executed. New PC: 30
-----------------------------------
```

- In clock cycle 0, process 0 calls semWait on the userInput mutex. It obtains the lock successfully.
- In clock cycle 1, process 0 assigns variable a input.
- In clock cycle 2, process 1 arrives and tries to call semWait on the userInput mutex. It is blocked.
- In clock cycle 3, process 0 executes again, assigns variable b input.
- In clock cycle 4, process 0 executes semSignal on the userInput lock. Process 1 now is unblocked.
- In clock cycle 5, Process 2 arrives and tries to call semWait on userInput. However, since process 1 has that lock, process 2 is blocked.
- In clock cycle 6, Process 0 executes semWait on userOutput mutex. It obtains the lock successfully.
- In clock cycle 7, Process 0 executes print from a to b
- In clock cycle 8, Process 1 executes assign a input.



```
Clock Cycle : 9
Process 1 is currently executing. PC : 30
Please enter a value: 36
Process 1 executed. New PC: 31
-----------------------------------
Clock Cycle : 10
Process 0 is currently executing. PC : 14
Process 0 executed. New PC: 15
Process 0 completed. PC: 15
-----------------------------------
Clock Cycle : 11
Process 1 is currently executing. PC : 31
Process 2 is unblocked
Process 1 executed. New PC: 32
-----------------------------------
Clock Cycle : 12
Process 1 is currently executing. PC : 32
Process 1 executed. New PC: 33
-----------------------------------
Clock Cycle : 13
Process 2 is currently executing. PC : 49
Please enter a value: CuteFileName
Process 2 executed. New PC: 50
-----------------------------------
Clock Cycle : 14
Process 2 is currently executing. PC : 50
Process 2 executed. New PC: 51
-----------------------------------
Clock Cycle : 15
Process 1 is currently executing. PC : 33
Process 1 executed. New PC: 34
-----------------------------------
Clock Cycle : 16
Process 1 is currently executing. PC : 34
Process 1 executed. New PC: 35
Process 1 completed. PC: 35
-----------------------------------
Clock Cycle : 17
Process 2 is currently executing. PC : 51
Process 2 executed. New PC: 52
-----------------------------------
Clock Cycle : 18
Process 2 is currently executing. PC : 52
Process 2 executed. New PC: 53
-----------------------------------
Clock Cycle : 19
Process 2 is currently executing. PC : 53
Process 2 executed. New PC: 54
```

- In clock cycle 9, Process 1 executes assign b input.
- In clock cycle 10, Process 0 executes again since Process 1 hasn't called semSignal on the userInput yet, so process 2 is still blocked. It calls semSignal on userOutput, then it terminates and is removed from the ready queue.
- In clock cycle 11, Process 1 executes semSignal on userInput, so process 2 is unblocked.
- In clock cycle 12, Process 11 executes semWait on file.
- In clock cycle 13, Process 2 executes assign a input.
- In clock cycle 14, Process 2 executes semSignal on userInput.
- In clock cycle 15, Process 1 executes writeFile a b
- In clock cycle 16, Process 1 executes semSignal on file mutex.
- In clock cycle 17, Process 2 executes semWait on file.
- In clock cycle 18, Process 2 executes assign b readFile a
- In clock cycle 19, Process 2 executes semSignal file.



```
Clock Cycle : 16
Process 1 is currently executing. PC : 34
Process 1 executed. New PC: 35
Process 1 completed. PC: 35
----------------------------------
Clock Cycle : 17
Process 2 is currently executing. PC : 51
Process 2 executed. New PC: 52
----------------------------------
Clock Cycle : 18
Process 2 is currently executing. PC : 52
Process 2 executed. New PC: 53
----------------------------------
Clock Cycle : 19
Process 2 is currently executing. PC : 53
Process 2 executed. New PC: 54
----------------------------------
Clock Cycle : 20
Process 2 is currently executing. PC : 54
Process 2 executed. New PC: 55
----------------------------------
Clock Cycle : 21
Process 2 is currently executing. PC : 55
Variable b content : 36
Process 2 executed. New PC: 56
----------------------------------
Clock Cycle : 22
Process 2 is currently executing. PC : 56
Process 2 executed. New PC: 57
Process 2 completed. PC: 57
```

- In clock cycle 20, Process 2 now calls semWait on userOutput.
- In clock cycle 21, Process 2 now calls print b. The value that is read from the file is output correctly.
- Lastly, In clock cycle 22, Process 2 calls semSignal userOutput. The program then terminates.