Operating Systems Project 1

Alejandro Lopez

Nova Southeastern University

Table of Contents

[Overview 3](#_Overview:)

[Features Not Completed 4](#_Features_not_completed:)

[Classes & Methods 5](#_Classes_&_Methods:)

[Pseudo Code 7](#_Pseudo_Code:_1)

[User Manual 11](#_User_Manual)

# **Overview:**

A simulated virtual machine called SharkOS that has registers that work to go around the array(memory) where all the microinstructions are stored. The programs start loading the programs from txt files that have the microinstructions inside them, and with the static value of a starting point as an index, the programs get stored inside the memory array. Once the programs are loaded, the timer will start which means that microinstructions have begun to make operations inside the machine’s memory through the Programs that were loaded in. Each program has an arrival time, which was supposed to work as a round robin operation, however, due to the time scope allocated I could not implement that the same way. Instead, based on the program’s arrival time whichever one is first starts running, then when a program arrives at a certain arrival time, the initial program gets interrupted. The initial program saves a temporary PSIAR and ACC value to resume working after all the jobs finish arriving. Each time a new job “arrives”, the previous one gets interrupted and lets the one that arrived run. Once all arrival times are finished, the machine will resume working on the programs that were interrupted.

# **Features not completed:**

* Due to midterms and the time scope allocated, I was not able to fully implement a round robin interrupt system. Instead, each time a file “arrives” the previous gets interrupted and lets the one that arrived run. This process is continued until the last file with an arrival time finish running. After that, the programs that were interrupted resume working based on the temporary PSIAR that was stored for them.
* One more feature that was semi-implemented was the automatic display of the STR operations where the result of the program gets stored inside a certain index in memory. I was able to get all those values and store them inside an array, however, some programs iterate over the STR procedure to get the programs result, therefore, the same STR got stored inside the array. In the end to display the results I just displayed all the values inside the STR array without duplicates.

# **Classes & Methods:**

* Main.java: Creates an instance of the class MainMemory to run its methods SharkOS()
* MainMemory.java: Contains all the methods that make SharkOS() function as it should.
  + SharkOS(): Calls INIT\_System and calls RUN\_SHARKOS() to initiate the program.
  + INIT\_System(): Initializes all the variables and arrays to either 0 or “”.
  + RUN\_SHARKOS(): This is the method that displays the before and after of the program. It displays the values before they were modified by the micro instructions and after.
  + LOAD\_SHARKOS\_PROGRAMS(): This function loads all the microinstructions stored inside the txt files that represent each program. Each line in the text file is stored inside the memory array based on a starting index that is hard coded.
  + INSTRUCTION\_SETUP(): Once all the array is populated with all the values from the txt files, this method starts checking the opcode of each microinstruction to then execute it and change memory based on the criteria of the program. While this method is checking for each microinstruction, it also checks for the arrival time of each program. If a program is currently running and another arrives, the program yields to the next program and saves a temporary PSIAR to resume working later.
  + GETPSIAR(): This method gets each starting location for each program and gives them to INSTRUCTION\_SETUP() to know which file was the one that interrupted the previous file that was running and know where to start running the next one.
  + divideInstruction(): This method divides the microinstructions inside the text files into two parts. The first part is the name of the microinstruction, the second is the value that comes along with that instruction. For example, ADD 200. ADD would be separated from 200.
  + divideProgram(): This function divides the hardcoded String that contains the locations of the text file as well as its starting index. For example, "src/Resource/Program1.txt 10" would get divided into src/Resource/Program1.txt and then into 10.
  + GET\_OPCODE(): This method sets an opcode for each micro instruction. For example, ADD would be given an opcode of 1 while STR is given an opcode of two. This helps identify each microinstruction to then create a switch case that checks for each one.
  + EXIT\_SHARKOS(): Once the program is finished executing, this method makes sure to exit the program.
  + ADD, STR, LDA, BRH, CBR, LDI, SUB, HALT are all microinstructions.
    - HALT: While it marks when a program finishes, it is also a method used to return either true or false depending on whether there are still programs to be ran.

# **Pseudo Code:**

* SHARKOS()
  + CALL INIT \_SYSTEM()
  + CALL RUN\_SHARKOS()
* RUN\_SHARKOS()
  + DISPLAY “SharkOS” AND “Loading Programs”
  + CALL SHARKOS\_PROGRAMS()
  + DISPLAY Programs Loaded In Memory
  + CALL INSTRUCTION\_SETUP()
  + DISPLAY “Fetching Results”
  + DISPLAY All The Populated Memory
* LOAD\_SHARKOS\_PROGRAMS()
  + FOR( i To Programs.Lenght )
    - CALL DIVIDE PROGRAMS()
      * GET Startprogramindex
      * GET Filename
    - SET Psiar > Startprogramindex
    - IF (Arrivaltimes[i] is 0)
      * SET Startingpoint > Startprogramindex
    - ELSE IF (Arrivaltimes[i] Is –1)
      * SET Startingpoint > Startprogramindex
    - DISPLAY "Program Loaded" + Filename + "Program Details"
    - READ FILE WHILE (There Is Next Line)
      * SET Rawdata > Scanner's Next Line
      * CALL DIVIDE\_INSTRUCTION ()
        + GET Opcode
        + GET Value
      * SET Memory[Psiar] > Value
      * SET Instructions[Psiar] > Opcode
* INSTRUCTION\_SETUP()
  + DISPLAY "Instruction execution begin"
  + WHILE (Working)
    - INCREASE Timer
    - FOR (i To Arrival times number)
      * IF (Timer IS ArrivalTime)
        + SET Interrupt > True
        + SET Psiartemp > Psiar
        + SET Acctemp > Acc
        + CALL GETPSIAR()

GET Next Psiar

* + - SET Csiar > Instructions[Psiar]
    - SWITCH CASE (Csiar 1-8)
      * 1
      * 2
      * …………
      * 8 – HALT
        + Working = HALT()
        + IF (Working AND NOT Interrupt)

SET Psiar > Next Psiar

* + - * + ELSE

SET Psiar > PsiarTemp

SET Acc > AccTemp

* GETPSIAR()
  + FOR i To Arrival Times Number
    - IF (Timer IS Arrival Time)
      * RETURN Next StartingPoint
* DIVIDE\_INSTRUCTION()
  + WHILE (String still has characters)
    - SPLIT STRING where it has a space
    - SET OpCode
    - SET Value
* DIVIDE\_PROGRAM()
  + WHILE (String still has characters)
    - SPLIT STRING where it has a space
    - SET FileName
    - SET Start Program Index
* GET\_OPCODE()
  + SET OpCode > 1 > ADD
  + SET OpCode > 2 > STR
  + SET OpCode > 3 > LDI
  + SET OpCode > 4 > LDA
  + SET OpCode > 5 > SUB
  + SET OpCode > 6 > CBR
  + SET OpCode > 7 > BRH
  + SET OpCode > 8 > HALT
* EXIT\_SHARKOS()
  + SYSTEM EXIT
* ADD()
* STR()
* LDA()
* BRH()
* CBR()
* LDI()
* SUB()
* HATL()
  + IF (There are more programs to run)
    - RETURN True
  + ELSE
    - RETURN False

# **User Manual**

//SET UP TO TEST - PLEASE READ ALL  
  
// File + Starting location for program  
public String[] Programs = new String[] {"src/Resource/Program1.txt 10", "src/Resource/Program2.txt 36", "src/Resource/Program3.txt 70"  
 ,"src/Resource/ProgramRequired1.txt 100", "src/Resource/ProgramRequired2.txt 500", "src/Resource/ProgramRequired3.txt 600"}; //The ProgramRequired(n) are the ones listed on Design\_Prog\_Project\_Assign\_1.pdf  
  
// THE ARRIVAL TIME IS BASED OFF THE MICRO INSTRUCTIONS ON EACH PROGRAM  
// THE FIRST PROGRAM MUST ARRIVE AT -1  
  
public int[] arrivalTimes = new int[] {-1, 6, 11, 17, 23, 29};

1. Things to Consider:
   1. The program was written in JAVA using IntelliJ
   2. The path of the file requires a resource folder

Graphical user interface, application

Description automatically generated

* 1. The path of each txt file is important otherwise the program will not work at all.
  2. Since the program produces such a large output on the console this is the order:
     1. After program is ran, it starts at the bottom where the results are shown.
        1. All the populated memory will be at the bottom.
        2. The STR locations/Results of the programs will be above all the populate memory.
     2. Keep scrolling up to see the execution of each program.

1. Understanding the output file:

---------------------------------------------

Loading programs..... : shows the programs loaded with their details

---------------------------------------------

---------------------------------------------

Instructions execution begin..... : Instructions and interrupts will be executed

---------------------------------------------

---------------------------------------------

INTERRUPT DUE TO ARRIVAL AT TIME: 6

SAVE PSIAR TEMP: 16 : Interrupt example

STARTING PROGRAM AT PSIAR: 36

---------------------------------------------

---------------------------------------------

YIELD TO NEXT PROGRAM :No interrupt, yield to next program

---------------------------------------------

---------------------------------------------

RESUME PROGRAM AT PSIAR: 16 :Resume working on interrupted program

---------------------------------------------

---------------------------------------------

Fetching Results..... :Displays results below this point

---------------------------------------------

1. Setting Up the Location of Each Program in Memory and Giving the Txt Path:
   1. The String Array called programs includes the path of the txt file separated by a space (“ ”) by the index where the program will be stored in. IMPORTANT: Make sure that the starting indexes of each program do not overlap each other.
2. Understanding and Setting Up the Arrival Times:
   1. The int Array called arrivalTimes includes the times at which each program arrives to be executed. Whichever file has an arrival time of -1 will be ran first. IMPORTANT: Order matters since in the example above the -1 corresponds to Program1.txt, while 23 corresponds to ProgramRequired2. IMPORTANT: The time of the system is decided by each instruction being ran. For example: ADD is ran -> timer +1, STR is ran -> timer +1. Therefore, if the first program (Program1.txt) is interrupted at time 6 which has instruction STR 32, that location will be saved and then the next program will start running.
3. Displaying the Results:
   1. Once the programs are loaded into memory, the program will automatically show the programs and their locations without any user input needed. However, when the results of all the programs are shown, this is still not fully automatic. This is because the index where each program will be stored will always remain a constant in the program, while the STR locations where the results of the programs are stored vary depending on the program. The STR locations will all automatically print out even if you change the starting positions for the programs, the programs themselves, or their starting position. However, they will not display which STR points corresponds to each program automatically and those STR points that loop will have duplicates be shown.