OS PROJECT REPORT

MILESTONE 0

BY:

- Yara Mohammed 40-1518
- Lama Ihab 40-1514
- Gasser Khaled 40-18970
- Ahmed Mohammed Salah 40-1316

We have decided to implement a university and payroll batch system. It is under the batch operating system type. Batch Operating Systems work periodically, doing the same task over and over again over a specified time frame.

We will be using the Java as the preferred programming language for our simulation.

Introduction:

A batch operating system is an operating systems in which same type of processes are batched together for execution. It's a relatively faster system than traditional system. Every specific period of time, some predefined processes are executed together only and once they are started, they cannot be changed or interrupted.

They tend to be useful in bulk database operations specifically due to the repetitive work that has to be frequently done. They're used mainly in business and product automation productions.

Methodology:

It will be responsible for calculating the yearly salary of employees (doctors and teaching assistants) as well as calculate the years of experience of the employee. It will also calculate the annual increase in the students' tuition fees (10%) and the GPA category the students belong to.

We will need a file management module to edit to and write in the spreadsheets of the respective tasks, we will not need memory management nor threading as only one process runs at a time. We will not need an Input/Output module as the changes take place inside the respective files and no output is needed to be displayed to the user.

2nd milestone - Initial implementation:

Created multiple classes:

- PCB.java (Process Control Block class): it has the attributes of every process which includes but not limited to - process_id, user_id, time the process needs to be executed as well as the period which is how often this process should take place.
- Process.java class: an abstract class that defines the main attribute every Process should have, which is the PCB. As well as an abstract method called “void run()” which is responsible for running the process.
- Multiple instances of different processes like PayRoll, TuitionFeesIncrease, etc. Where each class implements the run() method differently according to the class functionality.
- Scheduler.java class, which is responsible for literally “scheduling” the processes. It has an array list attribute which holds all the processes that are on this operating system and a ready queue that holds the processes that are actually due for running (by checking on their respective period attribute against the current day of the year).

Current Functionality:

Our basic operating system so far is very simple. We initialize an instance of the class Scheduler.java, it will run forever. Each day it will check on all the array list of processes, once a process that is due for working is found, it is immediately put into the readyQueue using the method called ***void Read(Process p)***, then it will be run using the method ***void Running()*** which removes the first element in the readyQueue and calls its respective run method, taking the required time to finish.

The Scheduler.java class “Schedules” the processes to run based on the FIFO (first come first served) principle.

The results on the console so far are:

- The day we are currently on.
- Which process is being run and its output, which is basically just a simple print statement.
- “Done!” which is printed after the process has been executed.

3rd Milestone: Memory and System Calls

- Created a class Memory that represents the physical memory of our system to store processes, which is an array of type Boolean with length 100 cells where if a cell contains true then some process is using it else if it’s false then it’s free to be used by any process. The class Memory has 3 methods. Check method that takes the number of contiguous cells a process will need to be stored in the memory and the method returns the index of the first available free cell to store the process or returns -1 when the memory doesn’t have space. Methods Use and Un-use are implemented to fill this part of the memory (according to length and first index) and un-fill it after finishing.
- Added attribute memorySize to PCB class to represent the size that each process will need to be stored in the memory.
- Added Class SystemCall which contains two methods that are called by any process when it needs to perform a system call to the operating system to write (print) or read something from memory.
- In class Scheduler: we are using FIFO or first come first served algorithm to run processes and we added the part in the main

method to check for an empty space in the memory first and then accordingly add it to ready queue and run it.

4th milestone – Graphical User Interface, IOModule, app functionalities and final simulation:

In this milestone we made the GUI that displays the output of running the OS. The GUI class consists of only 2 main elements; a constructor as well as an update method. The update method works by taking an instance of the Scheduler class as well as a Boolean flag that represents whether the CPU is currently active or not. It also has a method that updates the name of the current running process at the CPU.
Moreover, we discovered that we do in fact need an IOModule unlike what we first anticipated. The IOModule is responsible for dealing with all inputs and outputs (System calls) made to the disk. The IOModule also simply replaces the need for the initially implemented SystemCall.java class as it is redundant to have two classes almost simulating the same functionality.

Application functionalities:

- PayRoll.java: takes as input a .csv file that has all the Employees information. It then displays the total amount of money the university would have spent on Employee salaries.
- GPACategory.java: takes as input a .csv file that has all the Students information. It then calculates the specified category each student fits in. Writes the category to the file itself.
- TuitionIncrease.java: takes as input a .csv file that has all the Students information. It then updates the tuition fees the student has to pay by increasing 10% to the already present amount.

- ExperienceYears.java: takes as input the Employees.csv file and calculates the experience years of each employee. Updates the Employees file directly.

GUI components:

- A banner that displays what day of the year it is. Our OS simulations works independently of the real world's day, as in reality, it is supposed to be running forever because it is a batch system that does some specific tasks every specific interval of time, therefore the day displayed is relative to the day it was ran. We used 250 milliseconds to simulate the delay of a single day.
- A banner that displays the name of the current running process.
- A banner that shows if the processor is busy or idle.
- A progress bar that displays the utilization of the memory. It has 10 blocks, each representing 10 memory blocks. So if a memory is using, for example, 50 memory blocks, the GUI will show a 5 block utilization. The GUI displays memory block to the nearest 10, so if a process needs 31 blocks, it will show a 40 block utilization.

Log Files:

- One log file gets written with the CPU utilization every day that passes by. CPU utilization is either 100 or 0 as each process takes over the CPU completely and can only process only one process at a time.
- The other log file contains a complete logging of the processes and the days that they worked on. Basically re-writing all the print statements again in that file, funny enough.

<u>**Instructions on running the program**</u>:

- The main file to run is the Scheduler.java class.
- If you want to change anything, it is to be done on that file.
- If you want to add a process, just add it to the array of processes in the class. If you want to change anything regarding the process attributes, like TTF (time to finish) or period, it is to be changed inside the PCB constructor accompanying each process.