Concordia University
Engineering and Computer Science

PROJECT REPORT

Operating Systems

Course:	COEN 346	**Lab Section:**	FL-X
Assignment No.:	3		

Name:	Ravjotdeep Kaur	**ID No. :**	40108180
Name:	Kevin Phan	**ID No. :**	40097439
Name:	Samuel Lopez-Ferrada	**ID No. :**	40112861

Contribution by the group members

Group member	Tasks accomplished
Samuel Lopez-Ferrada	FIFO
Kevin Phan	Process and Clock class
Ravjotdeep Kaur	MMU

Introduction

This assignment consists of creating a process scheduler that simulates the use of memory management. The processes fetch and execute commands to the memory manager at runtime. They are constrained to a number of cores and run for a random duration between 10ms and 1 second. Shared access to the pages and disk space is managed with the use of semaphores.

Code Description

The following methods/functions/threads/data structures were used in this assignment. RunnableObject Class which implements the runnable interface

- ➢ Constructor
    - ■ Builds a Runnable Object with an array as the only parameter.
- ➢ Functions
    - ■ `public void run()`

      (it is overridden for the runnable interface)

      When an object implementing the Runnable interface is used to create a thread, starting the thread causes the object's *run* method to be called in that separately executing thread.
    - ■
      ```
      public   Process(int   arrival_time,     int   burst_time,   MMU
      pointer_to_MMU)
      ```

      The function is used to handle the processes in the system.
    - ■ `public String toString()`

The function is used to display the string value for the output. .

```
public Page()
```

This default construction initializes the page to the default. We used max integer value as a default to avoid conflict.

```
public void printPages()
```

This function is mostly used for debugging purposes and prints the current content of the pages.

```
public void initCommands()
```

This function reads the command.txt file and retrieves the essential. It stores each command as a String array and saves it to a linked list, for later use.

```
public MMU(int page_num) throws FileNotFoundException
```

Constructor for the MMU. This instantiates the MMU object and allows it to set up the commands and the page number

```
ynchronized void EXECUTE_NEXT_COMMAND() throws
InterruptedException, IOException
```

This executes the next available command from the commands list. It also ensures the commands rotate and makes the appropriate function call

```
synchronized void STORE (String variable_id, int value) throws
InterruptedException, IOException
```

This function stores the value on a page if it’s available. If there is no page available then it will directly input them into memory.

```
synchronized void RELEASE (String variable_id) throws
InterruptedException, IOException
```

This function releases the process.

```
synchronized int LOOKUP (String variable_id) throws
InterruptedException, IOException
```

This function checks if the given variable Id is stored in the memory and returns its value or -1 if it does not exist. If the Id exists in the main memory it returns its value. If the Id is not in the main memory but exists in disk space (i.e. page fault occurs), then it moves this variable into the memory and releases the assigned page in the virtual memory.

```
synchronized    int    SWAP    (String    variable_id)    throws
InterruptedException, IOException
```

This function needs to swap the variable with the least recently accessed variable, i.e. the variable with the smallest last access time, in the main memory.

```
synchronized  void  STORE_ON_DISK(String  variable_id,  int  values)
throws IOException
```

This function stores the given variable id and its value in the first unassigned spot in the memory.

```
public FIFO() throws FileNotFoundException
This function implements the FIFO algorithm.
public int readProcessFile() throws FileNotFoundException
```

This function reads the processes to the file.

```
public void transferArrivedProcesses()
```

This function transfers the arrived processes by comparing their arrival time and the current clock time. It directly adds them to the cpu core semaphore which is a FIFO queue.

```
public Clock(boolean trace)
```

This function implements the clock function

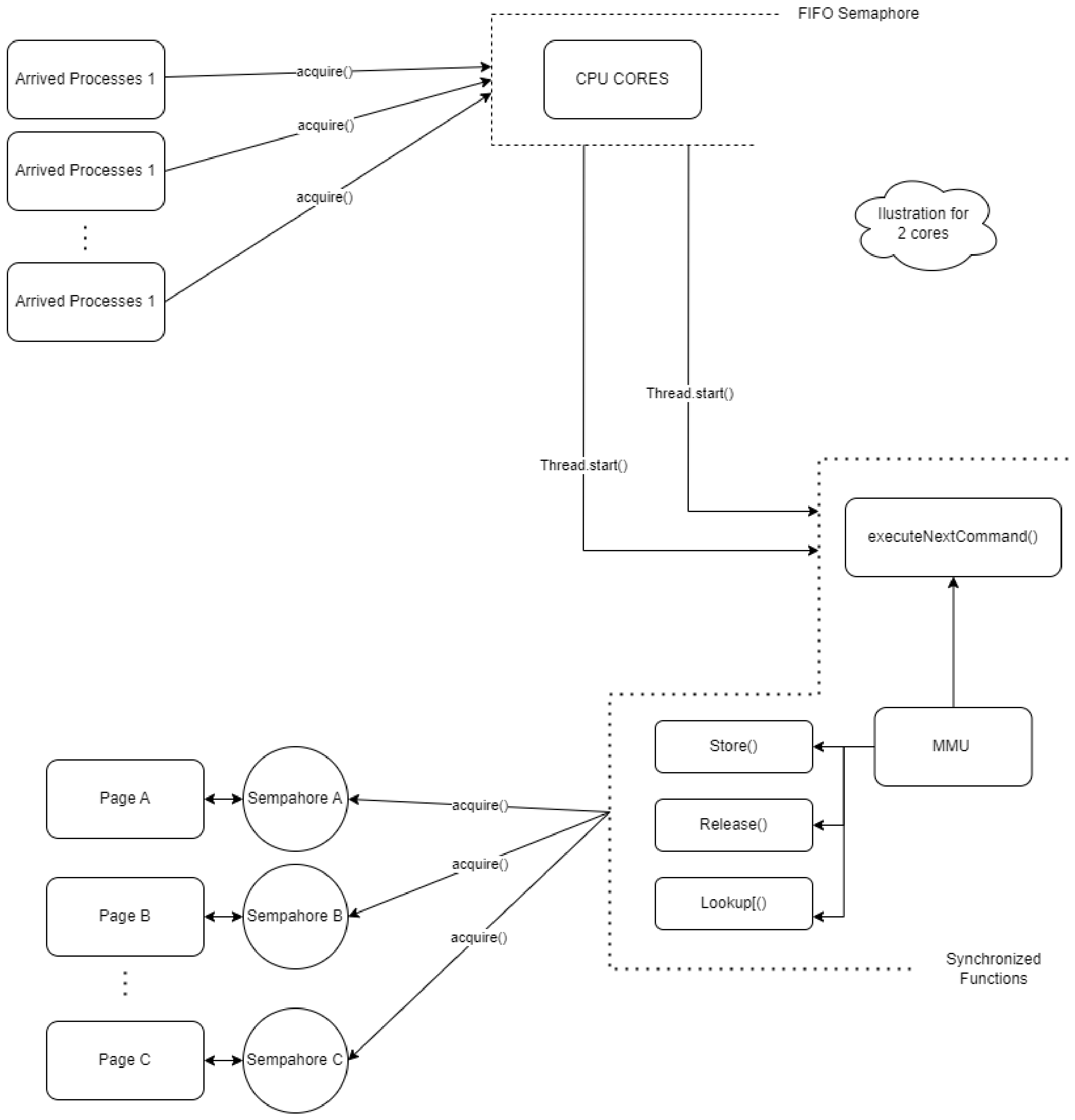
- Threads:

The threads in this project were implemented by passing them a runnable object and using the java thread class to instantiate a new Thread object. This approach limited the number of functions available from the Thread since we could only use the java functions start(), run() .. etc. This ensured encapsulation at run time. It forces us to contain all the runtime code to be executed within the run() function.

- Data Structures & Algorithms
  - FIFO

    Java already has a built-in structure for FIFO. The Semaphore class contains a fifo algorithm that can be set when creating new semaphores for the cpu cores. We passed in the number of resources we had and set the fairness to true. This implementation takes advantage of this built-in feature
  - Pages we built using an array to simulate ram. Its direct access simulates the speed of RAM.
  - Disk space was built using a linked list, simulating infinite disk space. Its time complexity also reflects the slow access to disk space.

Program Flow



Conclusion

In order to properly implement all the specifications of this assignment, we had to use our knowledge of Thread constructs, parallelism, critical section protection and the proper use of semaphores. The challenging parts were to coordinate various moving parts since every class or object had variables. The use of volatile made these variables change runtime which added a new layer of complexity and dynamism not encountered in previous classes or assignments.