**COEN346**

**Programming Assignment 3-**

**Virtual Memory Manager**

**Report**

Dr. Ferhat Khendek

Section F

Nabila Tabassum ID:40109039

Jaskirat Kaur ID: 40138320

Laura Hang ID: 40203006

**Introduction:**

In this assignment, we simulated an operating system’s virtual memory management while considering the scheduling and synchronization of the processes that are to be executed. The virtual memory consists of the main memory and large disk space. Main memory has a limited number of pages whereas disk space has an unlimited number of pages. If the main memory is full, the variable will be stored in the disk memory. Whenever the user needs to access that variable, it is moved to the main memory and swapped with the least recently accessed variable. To be able to manage the variables, the virtual memory manager offers three APIs to the processes: Store, Release, and Lookup.

**Discussion:**

**Main:**

* We open our 3 .txt files (namely memconfig.txt, processes.txt, command.txt) to read the file contents and save their values in different variables.
* Processes read from the processes.txt file are initially inserted in the processes list.
* Commands.txt file is also read and inserted into a list.
* Main memory is created according to the indicated size in memconfig.txt file.
* We sort the processes according to their arrival time.
* Output file namely output.txt is opened.
* Then we initialize the clock, virtual memory, virtual memory manager and scheduler.
* We also create and start threads for virtual memory manager and scheduler.
* Clock threads wait for scheduler thread to start.
* After we join all the threads , the output.txt file is closed.

**Processes:**

def\_init\_\_( self, clock, manager, commands, active\_process,file\_out, proc\_id, start\_time, service\_time):

In this class, initialization of the following class has been done by setting the constructor to their default value. For example, name and process id has been set to process and process ID and so on.

Def\_run(self):   
Thread has been declared in this following , for starting, running and existing the threads according to the arrival time of the clock. When the timer for the thread is less than the terminate time, then it is going to run the command line by line.

Def\_run command(self):   
A lock state is being used to prevent the command from running more than once. In order to implement that, we use lock. acquire() and it immediately lock the command on the basis of the timer.Afterwards, lock.release() has been used within the lock state to change it to unlock state, i.e lock. release().

**Virtual Memory Manager(VMM):**

def store():

When the Main memory is full then it is going to add a page to the disk ( extra space).

def release():

When the main memory is full, then the page is going to release the commands.

deflookup():

When the memory is full, the instruction checks and returns its value. Otherwise, it will return -1.

def swap():

The instruction checks if the ID is in the main memory. If the instruction is not in the main memery but in the main memory then it swaps and returns the values.

Def call\_api():

When the length of the command is equal to 0 and length of the command is equal to 3, then it is going to store the value of command(1) and command(2).

**Main Memory:** This class is used to handle virtual memory. We first initialize the main memory and then we implement following methods::

* def isFullMemory: This function checks if the main memory is full or not. If all the pages in the main memory are filled with variables and no more space is left it will return true.
* def updateLastAccess: This function is used to update the last access time of the variable that has been recently accessed.
* def storeMemory: This function is implemented to store variable’s id, value and last access time. First, it checks if there is any empty space in the main memory. If there's an empty page available it will store that variable and its attributes.
* def freeMemory: If a user wants to release the variable from the memory space, this function removes the variable and frees the main memory space.
* def findVariableMemory: It is used to find the variable the user is looking for and update its last access time.
* We also have some appropriate getters and setters to facilitate these functions.

**Scheduler:**

def \_\_init\_\_():   
All the attributes were declared as constant. For an instance, the vmmManager was declared as vmm.

In the Def run(self), Logging. Info were used to start and finish the thread according to the clock cycle and to store the process ID. While the number of run process is less than the number of the core i, then it is going to start the run command. When the process is already in the list of main memory then, it is starting the thread process and pops it out if the process is already there in the main memory. .Setters were used update the values of stop thread

**Disk:**

In this class, we implemented disk virtual memory. We implemented 5 functions:

-def checkVariableExist: It verifies if the variable id exists in the disk memory.

-def addVariable: It is implemented to add variables to the disk memory.

-def findVariable: It is used to find Variable Id in the disk page.

-def removeVariable: This function removes the variable from the disk if it is no longer required or it has to be moved to the main memory.

-def getValue: This function returns the Id of a variable in the disk memory.

**Clock:**

This class is used to handle threaded clock. In this class, RunClock(self) function runs when the clock thread starts. It also contains appropriate getters and setters to update the time and to know the state of the thread.

**Flow of program:**

Our program does three main API calls: store, release, lookup. For store(), our virtual memory manager simply stores the variables in main memory. If there is no space left in main memory, the variable will be stored in the disk memory. And last access is updated every time any variable is stored. For lookup, our memory manager first searches in the main memory. If that variable is present in the main memory, it will return its value and last access is updated. Otherwise, it will search the variable in disk memory and return the value. For release, the manager simply removes the variable from the main memory or disk memory.

**Conclusion:**

The concept of virtual memory as seen in the class was further delved into with this programming assignment. Concepts of main memory and external disks were thoroughly understood and implemented in the assignment. We also learned the importance of process synchronization. Since multiple processes can run at same time and work on different commands. It was very important to ensure that one process runs one command at a time and also one command should be executed just once rather than multiple times.Therefore mutual exclusion was a crucial part to implement. We used locks (acquire() and release() functions) to ensure synchronization and mutual exclusion in our program.

**Contribution:**

| Laura Hang | Clock, Main, scheduler, VMM. main memory, process, disk |
| --- | --- |
| Nabila Tabassum | Main, scheduler, VMM. main memory, disk, clock |
| Jaskirat Kaur | Main, scheduler, VMM. main memory, processes, clock |