Jasen Ratnam
40094237

16/04/2020
COEN 346

Programming Assignment 3
Report

This programming assignment has the goal to implement a round-robin scheduler and a virtual memory simulator using threads. For this assignment, we want to have a CPU scheduler running on a machine with two CPU, therefore two threads (CPU) can be running at a time. I.e. we want our scheduler to give a process to one of the available CPUs for its quantum and let another process run in the other CPU, while both processes may have access to the virtual memory simulator, therefore, we must also create some kind of lock to allow one process to enter the virtual memory at a time. Then the scheduler should take back control of the CPUs and give them other processes with the shortest remaining time accounting for some fairness. We also note that each process has its own arrival time and cannot execute before it has arrived. Once the simulation has ended, the program should output in a text file the time at which a process has started, resumed, and finished along with the total waiting time of each process and in another text file the virtual hard disk simulated with the variables stored in them as stated below.

In addition, we needed to implement a Virtual Memory Management which consists of a fixed size main memory, simulated with a fix sized array in the program and unlimited disk space, simulated by a text file stored in the systems hard-drive named “vm.txt”. The size of the array simulating the main memory, i.e. the number of frames in the main memory, is given by an input text file named “memconfig.txt”. The Virtual Memory Managements principle job is to store, release and retrieve variables to/from the memory. The variables are stored in memory using pages that hold a variable, its id and the last time it was accessed. The simulation can be tested using a list of commands given to the program that will output a text file with detailed information on what the processes are doing with time.

The requirements for this assignment are that both the scheduler and processes must be simulated using threads, with two thread running at a time. Also, this solution must work for an arbitrary number of processes given simulating a Virtual Memory Management to handle variables.

The code attached to this document simulates a round-robin scheduler, with the inputted processes from a text file and virtual memory management.

The solution to this assignment uses a number of classes that talk to each other to simulate scheduling and virtual memory. With the main class being VirtualMem.java.

The main class first makes an IOFile class instance which is used to read the input files inputted by the user, the list of processes, the list of commands and memory configuration file, while it reads the file it creates instances of the process class and creates the process given by the user with its arrival time and burst time. It also reads the commands file and returns a list of commands to the main class along with the main memory size from the memory configuration file. The processes objects are then put into an ArrayList and sent back to the main class. Then a scheduler class instance is started which is also a thread, and the ArrayList of processes is sent to it. Then the scheduler thread is started. With the scheduler thread started, the main class makes a thread with MyThread class for every process in the ArrayList of process, starts it and then immediately suspends it waiting for the scheduler to call it. These threads are then given to the scheduler to handle. The scheduler class puts these threads in a circular list with the implemented circular list class from the book “Applied Operating Systems Concepts - John Wiley and Sons, Inc.” adapted for my use.

From here the main class does nothing else and gives full control to the scheduler thread. To start, the scheduler gives himself a priority of 6 so that it retake control of the CPU whenever it wakes up. Then it goes into an infinite while loop where it handles all the processes given by the main class until there are no more processes to run.

First, the scheduler finds the next process to give the CPU to using the nextProcess() method. This method looks in the ArrayList of processes given by the main class and finds the process with the least remaining time and returns its id. If two process has the same least remaining time, the method will choose the oldest process. This method also accounts for fairness and will choose a process that has been waiting for more than 5 rounds to get the CPU, regardless of its remaining time. If this method returns -1, there are no more processes to execute and the scheduler can end by outputting the waiting times of each process. This is repeated to find the process for the second CPU.

With the id of the next process to get the CPU gotten from the method above, the scheduler finds the thread associated with this process from the circular list queue created by the main, using the getProcess(int i) method from the circularList class. This thread is then given a priority of 4 to make sure it gets ran when the scheduler goes to sleep. The required outputs get printed depending if the process just started or resumed. Then the time is incremented by the quantum of the process chosen. The process is resumed and the scheduler goes to sleep for the quantum of the process. The scheduler also finds the thread associated with the second process for the second CPU.

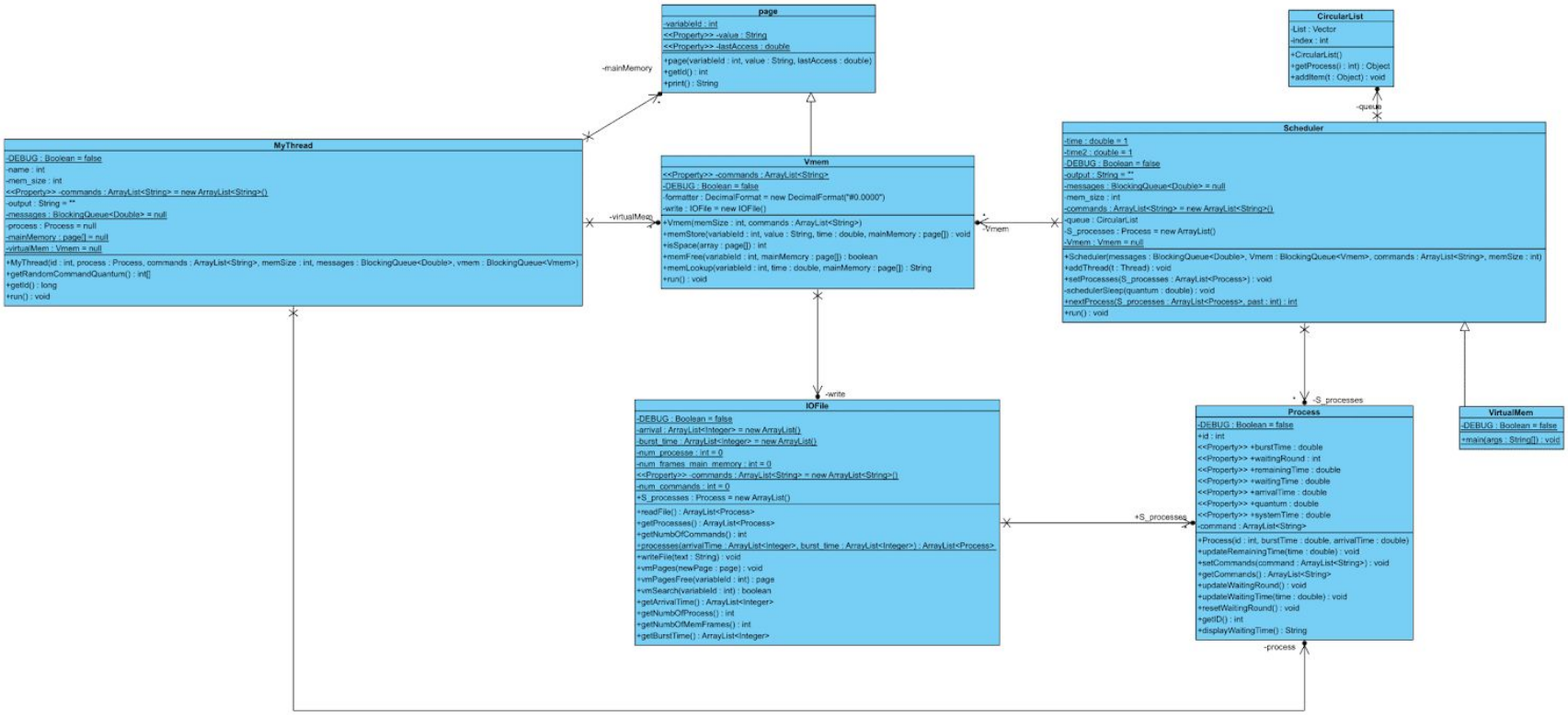
Once the scheduler goes to sleep, the chosen processes then gets both the CPUs since it has a higher priority than the other processes. Once it gets the CPU, the process will enter the while loop of the process if it’s the first time it gets the CPU and will stay in the loop until the remaining time of the process is 0. While in the loop, the process updates its remaining time and then suspends itself giving the CPU back to the scheduler. Both processes have the CPUs for its quantum times.

Once a process is given a CPU, it starts reading the list of commands given and executes them. This is where the error of my solution happens, every time a process gets a CPU it reads the entire list of commands given by the user instead of getting a part of the commands and giving the rest to the other processes or the next time the process is given the CPU. The solution currently repeats the same list of commands every time the CPU gets a process. This error happens because I could not figure out how to split the commands efficiently for every time the CPUs get a process.

While the process reads the commands, it executes them one by one using the virtual memory simulator. The commands can tell the process to store, look up and/or free up variables from memory. The variables are either stored in the main memory which is an array of the size given in memory config file or in the hard disk which is an unlimited size text file. After the process is done doing all the commands it returns the CPU to the scheduler.

The scheduler then outputs the pause time of the processes or its finish time and total waiting time, if the process finished. It then resets the priority of it to 2 and goes back to the start to find the next processes to give the CPUs and repeats until all processes have no more time remaining. Once all processes are done, it outputs the total waiting time of each process and exits the system.

Note: All outputs are done in a text file named output.txt that can be found in the same folder as the project, along with the virtual hard drive file named vm.txt.


page
-variableId : int
<<Property>> -value : String
<<Property>> -lastAccess : double
+page(variableId : int, value : String, lastAccess : double)
+getId() : int
+print() : String
-mainMemory
CircularList
-List : Vector
-index : int
+CircularList()
+getProcess() : int) : Object
+addItem(t : Object) : void
-queue
MyThread
-DEBUG : Boolean = false
-name : int
-mem_size : int
<<Property>> -commands : ArrayList<String> = new ArrayList<String>()
-output : String = ""
-messages : BlockingQueue<Double> = null
-process : Process = null
-mainMemory : page[] = null
-virtualMem : Vmem = null
+MyThread(id : int, process : Process, commands : ArrayList<String>, memSize : int, messages : BlockingQueue<Double>, vmem : BlockingQueue<Vmem>)
+getRandomCommandQuantum() : int[]
+getId() : long
+run() : void
-virtualMem
Vmem
<<Property>> -commands : ArrayList<String>
-DEBUG : Boolean = false
-formatter : DecimalFormat = new DecimalFormat("#0.0000")
-write : IOFile = new IOFile()
+Vmem(memSize : int, commands : ArrayList<String>)
+memStore(variableId : int, value : String, time : double, mainMemory : page[]) : void
+isSpace(array : page[]) : int
+memFree(variableId : int, mainMemory : page[]) : boolean
+memLookup(variableId : int, time : double, mainMemory : page[]) : String
+run() : void
-Vmem
Scheduler
-time : double = 1
-time2 : double = 1
-DEBUG : Boolean = false
-output : String = ""
-messages : BlockingQueue<Double> = null
-mem_size : int
-commands : ArrayList<String> = new ArrayList<String>()
-queue : CircularList
-S_processes : Process = new ArrayList()
-Vmem : Vmem = null
+Scheduler(messages : BlockingQueue<Double>, Vmem : BlockingQueue<Vmem>, commands : ArrayList<String>, memSize : int)
+addThread(t : Thread) : void
+setProcesses(S_processes : ArrayList<Process>) : void
-schedulerSleep(quantum : double) : void
+nextProcess(S_processes : ArrayList<Process>, past : int) : int
+run() : void
-write
IOFile
-DEBUG : Boolean = false
-arrival : ArrayList<Integer> = new ArrayList()
-burst_time : ArrayList<Integer> = new ArrayList()
-num_processe : int = 0
-num_frames_main_memory : int = 0
<<Property>> -commands : ArrayList<String> = new ArrayList<String>()
-num_commands : int = 0
+S_processes : Process = new ArrayList()
+readFile() : ArrayList<Process>
+getProcesses() : ArrayList<Process>
+getNumbOfCommands() : int
+processes(arrivalTime : ArrayList<Integer>, burst_time : ArrayList<Integer>) : ArrayList<Process>
+writeFile(text : String) : void
+vmPages(newPage : page) : void
+vmPagesFree(variableId : int) : page
+vmSearch(variableId : int) : boolean
+getArrivalTime() : ArrayList<Integer>
+getNumbOfProcess() : int
+getNumbOfMemFrames() : int
+getBurstTime() : ArrayList<Integer>
+S_processes
-S_processes
Process
-DEBUG : Boolean = false
+id : int
<<Property>> +burstTime : double
<<Property>> +waitingRound : int
<<Property>> +remainingTime : double
<<Property>> +waitingTime : double
<<Property>> +arrivalTime : double
<<Property>> +quantum : double
<<Property>> +systemTime : double
-command : ArrayList<String>
+Process(id : int, burstTime : double, arrivalTime : double)
+updateRemainingTime(time : double) : void
+setCommands(command : ArrayList<String>) : void
+getCommands() : ArrayList<String>
+updateWaitingRound() : void
+updateWaitingTime(time : double) : void
+resetWaitingRound() : void
+getID() : int
+displayWaitingTime() : String
-process
VirtualMem
-DEBUG : Boolean = false
+main(args : String[]) : void

Error in solution:

I was not able to separate the commands between the processes and their execution times. Hence, every time a process is executed, all commands are executed. Therefore, the same commands are repeated multiple times in simulation. This is wrong and shouldn't be working in this way but we can ignore this error and assume that the list of commands is actually multiple repetitions of the same commands for the sake of testing the functionality of the rest of the assignment. We can see from the output that our virtual memory management works as expected with the main memory of set size and unlimited disk space.

Conclusion

In this programming assignment, we learned how to use threading to accomplish round-robin simulation which is the shortest job remaining time scheduling along with virtual memory management simulation. This solution output shows that it is somewhat accurate since it gives two CPU to two processes with the least remaining time accounts for fairness since it does not let processes wait forever if there are smaller processes that will always go first. With each process doing the list of commands using virtual memory management. The solution does not output the desired answer since I could not figure out how to split the commands to the processes. Instead, my program does all the commands in every instance of a process, making the same commands being repeated multiple times. I was able to enable the mutual exclusion between threads using blocking queues to send the virtual memory instance and time instances between the threads. This method was efficent for my simple use of virtual memory simulation.