

Processor Execution Simulation Assignment

Dr**.** Fahed Jubair

Mohammad Amjad AlQuraan

Configuration

* “configuration.txt”
* First line represents the number of processors ex: 5
* Each line after represents task information’s separated by comma using the following format:  
  Creation time, Requested time, Priority
* **example:**  
  5  
  20,5,high  
  23,3,low  
  30,10,high  
  ..etc.

Main Method:

* Reads configuration settings from a file “configuration.txt”
* Read the first line which represents number of Processors
* While end of file reads each line that represents Task information Creation Time, Requested Time, Priority and create new object of task for each line
* Start the Scheduler loop to simulate the process execution
* Handle the exception in case of any error like wrong configuration format

UML:

Diagram, table

Description automatically generated

Clock:

* Used singleton class to create clock class
* The instance of the clock used in the Simulator because it’s shared between all processors
* Attributes: Counter
* Methods: incrementCounter used to increment the counter at the end of each cycle

Task State:

* Enum used to implement the three states of the task
* WAITING, EXECUTING, COMPLETED

Processor:

* Class used to make instance for each processor in the scheduler
* Attributes:
  + ProcessorNum static variable increased every new object instantiation  
    used for assign ID for each new processor
  + ID
  + State Boolean true for busy false for idle
* Methods:
  + Constructor: sets the attribute ad push the processor to idleProcessors stack in the simulator
  + getID
  + setState
  + isBusy

Task:

* Class used to make instance for each task in the scheduler
* Attributes:
  + taskNum static variable increased every new object instantiation  
    used for assign ID for each new processor
  + ID
  + State of type TaskState
  + creationTime, requestedTime, CompletionTime
  + counter, to keep counting until the counter reaches the requested time needed
  + priority (true for high, false for low)
  + P, object of type Processor: assigned when the task is executing to a processor and assigned to null if its completed or in the queue
* Methods:
  + Constructor: sets the attribute ad push the Task to listOfTasks linkedList in the simulator
  + Getters and setters for the attributes
  + IncrementCounter: method used at the end of each cycle to increment the counter of task and checks if the counter==requested time to set the state to Completed

Simulator (Abstract class):

* Class used to define the data structures needed in the simulation process
* Attributes:
  + **LinkedList<Task> listOfTasks = new LinkedList<Task>**to store the tasks when created
  + **Queue<Task> highQueue = new ArrayDeque<Task>**  
    to store the high priority tasks in the state of WATING
  + **Queue<Task> lowQueue = new ArrayDeque<Task>**  
    to store the low priority tasks in the state of WATING
  + **Stack<Processor> idleProcessors = new Stack<Processor>**  
    to store the idles processors
  + **<Task> lowExecuting = new Stack<Task>**  
    to store the high priority tasks in the state of EXECUTING
  + **<Task> highExecuting = new Stack<Task>**  
    to store the low priority tasks in the state of EXECUTING
  + **<Task> completedTask = new Stack<Task>**  
    to store the tasks in the state of COMPLETED
  + **Clock clock = Clock.getInstance()**  
    shared clock in the simulation process
* Methods:
  + printCycle: used to print the cycle number on the screen
  + printTaskStatus: used to print the task and a message indicates what is going on with the task now
  + printSummary: print summary on the screen after the completion of simulation process

Scheduler (Abstract class):

* Class used to define the methods needed for the scheduling methods that are needed to start the simulation of scheduling process
* The class extends the Simulator class to inherit all the data structures that are needed
* Attributes:
  + sortedTasks: stack used to store the tasks sorted by the creation time
* Methods:
  + sortTasks: sort the listOftasks and store them in the sortedTasks Stack
  + addToQueue(Task): add the task to the appropriate queue based on the priority
  + isThereIdleProcessor: return Boolean if there is idle processor in the stack
  + isThereHighQueue: return Boolean if there is a high priority task WATING
  + isThereLowQueue: return Boolean if there is a low priority task WATING
  + isThereAssignedLowTask: return Boolean if there is low priority task EXECUTING for the case of INTERRUPT s
  + addToCompletedTasks: loop through EXECUTING tasks and check if there is COMPLETED tasks to add in the stack
  + assignToProcessor:  
    check if there are tasks WAITING and assign them to an idle processor then set their state to EXECUTING, and handle the INTERREPUT if needed
  + loop():
    - start the scheduler and start the clock cycle
    - check if there is a creation of a task at the current cycle
    - handle the scheduling process for the tasks
    - finish until all the tasks are completed
    - print the cycle, tasks, processors statues at each cycle
    - print summary at the end of the simulation