# 

# **Punjab University College of**

# **Information Technology**

**Assignment – 02: MLFQ**

Course: Operating Systems

Instructor: Mrs. Saadia Shahzad

Submitted by: Muhammad Bilal (BCSF18M029)

# **Minor Adjustments:**

The time quantum for queue0 and queue1 have been reduced to 5 & 10 (integer units) respectively. The promotion policy waiting time for queue2 and queue3 have been reduced to 20 & 30 (integer units) respectively. This has been done in order to accommodate the inputs in which the created Gantt Chart spans full screen.

# **Source Code Files Explanation:**

The following source files have been included in the project. The functionality for all of them is explained as follows:

**Process.h:** A class for creating objects that house the PID, arrival time, burst time of a process, it has getter setter for each of the above-mentioned attributes, and an overloaded operator “==” which compares two processes and returns true if their attributes are same otherwise false.

**ProcessQueue.h:** A class that generates Process Queue. It has following attributes:

* An array of processes that hold the process objects.
* A processNum counter that holds the number of processes currently present in the array.
* A processSize counter that holds the maximum number of processes that an array can hold. (The array can actually hold any number of processes, as it resizes itself whenever an insertion operation takes place when the array is full, this attribute is taken as a reference for when the array should be resized).

The following functions can be found in the class:

* Getter setters for the above-mentioned attributes.
* Copy constructor, assignment operator, and destructor.
* Insertion and deletion functions: The insertion functions always insert processes at the end of the queue, while the remove function is overloaded and can either receive an index or a process as a parameter, which if then valid or found, will be removed.
* Overloaded operator [ ]: This operator acts as a getter+setter for the process array and throws an exception when the index passed to it is out of bound.
* Sorting functions such as sortByArrivalTime, sortByBurstTime which sort the process array by arrival and burst time respectively. Another sort function is used which sorts processes by burst time but can be used on specific indexes for a process queue; sortByBurstTimeFromIndex().
* Lastly, the function addIdleTime(): This function scans the process queue and searches for gaps in between process execution using arrival time and their burst time, and adds idle processes in the array.

**RR.h:** A class inherited from the ProcessQueue class. This class has one extra attribute of timeQuantum along with the getter setters of this attribute. This attribute holds the time quantum of the queue.

**SRTF.h:** A class inherited from the ProcessQueue class. It has no extra attribute or functionality. It is implemented only to improve readability of the code.

**MLFQ.h:** A class inherited from the Process Queue class. It has three extra attributes as following:

* queue0: An object of type RR initialized with a time quantum of 5.
* queue1: An object of type RR initialized with a time quantum of 10.
* queue2: An object of type SRTF.

The class contains following functionalities:

* **runQueues() function:** This function is the main part of the assignment. It prepares and returns the Gantt Chart queue which is then dumped onto the screen in the form of a Gantt Chart. The function uses following functions to perform this functionality:
  + **selectMinimumBurstProcess():** This function swaps the current iteration process in the queue2 with the least burst process available process in the queue2.
  + **runQueue0():** This function processes the most recent added process into queue0, one at a time. If more than one functions are inserted into it at a single time, it processes them one by one. It checks if the function burst time is less than queue0 quantum time or not. If the burst time is lower than queue0 quantum time it processes the allowed burst time in queue0 and demotes (inserts into queue1) it after subtracting the burst time from it i.e. equal to timeQuantum otherwise it executes the process without demoting it. It then adds the executed process to ganttQueue as well.
  + **runQueue1():** This function processes the most recent added process into queue1, one at a time. If more than one functions are inserted into it at a single time, it processes them one by one. It checks if the function burst time is less than queue1 quantum time or not. If the burst time is lower than queue1 quantum time it processes the allowed burst time in queue0 and demotes (inserts into queue1) it after subtracting the burst time from it i.e. equal to timeQuantum otherwise it executes the process without demoting it. It then adds the executed process to ganttQueue as well.
  + **runQueue2():** This function processes the most recent added process into queue2, one at a time. If more than one functions are inserted into it at a single time, it processes them one by one. It first calls the selectMinimumBurstProcess() and then checks if the original inputQueue for the arrival time of each process i.e. it checks if all the processes have arrived into queue0 or not. If all the processes have arrived, it executes all the burst of the current iteration process. Otherwise it executes the burst of that process equal to arivalTime of the next process minus burstTime of that process, if current time + burstTime is greater than the arrival time, and then adds the same process with its remaining burst time to queue2, otherwise it executes the whole of that process. The function then adds this executed process to the ganttQueue.
  + **promoteToQueue0():** This function checks if the process has waited in queue1 for more than 20 seconds by adding 20 to its arrival time in queue1 and then comparing it with current time, if it is less then it promotes that process to queue0.
  + **promoteToQueue1():** This function checks if the process has waited in queue2 for more than 30 seconds by adding 30 to its arrival time in queue2 and then comparing it with current time, if it is less then it promotes that process to queue1. It then calls the promoteToQueue0() method.
  + **dumpQueues():** This function is called whenever a function is added to queue0, queue1, or queue2 to simulate regular insertion and deletion in each queue.

The process runQueues() first calls the sortByArrivalTime() method to sort the inputQueue by its arrival time. It then creates and instance of ProcessQueue class named as ganttQueue, the purpose of which is mentioned above. The method then initializes 4 int variables that will be used to iterate each of the queues, (inputQueue, queue0, queue1, queue2). The function also creates a variable of time (int) that is used to keep the record of current time. A loop is used to process all the queues. The loop terminates when all the iterators have reached the end of their respective queues i.e. when all the process have been executed. The process then compares the current time with the arrival time of all the processes existing in the input queue and inserts processes to queue0 accordingly. An if/else condition that governs which queue is to be run next, the queue highest priority queue which has unexecuted process is ran.

**displayGanttChart():** This function calls the runQueues() method to generate a ganttQueue. This function then generates the index bounds of the next n processes to be dumped on the screen, where n is the number of processes that have a combined burst time of less than or equal to 60. The function then passes these bounds to the function dumpChart().

**dumpChart():** This function receives the indexes and the ganttQueue. The function then iterates three loops, first to display the processes PID with appropriately calculated amount of spaces, second to display **–** (hyphen for one burst unit) & **|** (pipe for interval), third to display the start time and end time of each process.

**dumpGanttChartStats():** This function is called from the displayGanttChart() and takes the ganttQueue as an argument. It then scans the ganttQueue for each process’ start time, finish time, and then calculates the following terms:

* + **Wait Time:** Final time – arrival time – burst time
  + **Turn Around Time:** Final time – arrival time
  + **Response Time:** Start time – arrival time
  + **Average Response Time**
  + **Average Waiting Time**

**startApp.h:** This class has two methods and one attribute of type MLFQ.

* **readFromFile():** This function reads from files and fills the MLFQ attribute. The delimiter in the file for pid, arrival time and burst time is colon “:” I.e an input file having a process specified as: 1:0:20, will indicate that the process has a pid 1, arrival time 0, burst time 20.
* **convertStrToNum():** This function converts the burst time and arrival time strings to integer.
* **displayGanttChart():** This function acts as a wrapper for the MLFQ::displayGanttChart function.

**driver.cpp:** This file is used to generate the 3 objects of the startApp class, for each input file.

**Execution Instructions:** The project was created in Visual Studio. To run the program, compile and run the driver.cpp file with all the other files as dependencies. After the execution of each file, the user will be prompted to press enter to start executing the next file until all 3 files have been executed.