H2 Computing Practical Worksheet – T1W6

1 One of the functions of an operating system is to manage process scheduling. For more detailed information, please refer to: https://en.wikipedia.org/wiki/Scheduling_(computing)

In this exercise, you are supposed to implement a Round-robin scheduling algorithm. For more details on this algorithm, please refer to: https://en.wikipedia.org/wiki/Round-robin_scheduling

1a Write a script that generates a list of 30 processes, and saves it in a text file, PROCESSES.TXT using one line for each process using the following format:

<Process ID>,<Arrival Time>,<Execution Time>

Note that:

- Process ID: a 4 digit positive integer (with preceding zeros)
- Arrival Time: a positive integer representing that start time of the process (in milliseconds); the range should be between 0 and 250
- Execution Time: a positive integer representing the amount of time required to execute the process; the range should be between 10 and 30

1b Write a script to read the contents of the text file PROCESSES.TXT, sort its contents by Arrival Time and then write the sorted processes to the new text file SORTED_PROCESSES.TXT.

Note that you should implement Quicksort to perform the sorting.

1c Write the object-oriented code for a singly-linked linked list. Your implementation should ensure the following methods:

- Fully object-oriented
- Able to store generic data objects
- Insert method - at the back of the list
- Find methods - returning True if the given data object is stored in the linked list, and False
- otherwise
- Delete method, which utilises the find method and then removes the given entry if possible; it must return True is successfully removed, or else will return False
- Print method should print the complete contents of the linked list, including any sentinel nodes and all linking data (i.e., there should be some formatting to indicate how the nodes are linked – ensure proper UI principles are adhered to when formatting this output)

1d Design and implement an object oriented Priority Queue that inherits the Linked List implemented in **1c**. It should include the following methods:

- Initialisation – initialises an empty Priority Queue
- Push – inserts a new instance of the object, which utilises a binary search method to determine where it should be inserted (based on the variable used to determine priority)
- Peek – returns an instance of the object currently at the front of the Priority Queue
- Pop – utilises pre-order traversal to print the contents of the BST
- Print – prints the contents of the Priority Queue from front to back; each instance should be printed on a new line (and printing each instance should use that object's own print function)

1e Write a script that simulates the Round Robin Process Scheduler by using the processes generated in SORTED_PROCESSES.TXT from **1b** and using the Queue implemented in **1d**.

You must design your own object oriented Process class (to be stored in the Priority Queue). It should include its own get/set methods and a print method. It must store the:

- Process ID
- Arrival Time
- Remaining Execution Time

Your simulation should perform the following:

- Populate the Priority Queue using the processes stored in SORTED_PROCESSES.TXT; you should ensure that all the processes are queued by arrival time priority
- Have a variable storing the currently running process and initially set this as empty
- Have another variable that stores the amount of time a process has been running and initially set this as empty
- Assume that your simulation begins at time index 0 (in milliseconds)
- Assume that your Round Robin Process Scheduler utilises a time quanta of 5 milliseconds
- For each time index (including 0), your simulation should check if a process is already running
    - If there are no processes running (i.e., the variable holding the currently running process is empty):
        - Peek at the process at the front of the queue
        - If the arrival time of that process is either equal to or is past the current time index, then pop that process and set it as the currently running process; also then set the variable storing the amount of time the currently running process has been running to 0
    - If a process is already running (i.e., the variable holding the currently running process is not empty):
        - Increment the variable storing the amount of time the currently running process has been running by 1
        - Check if the currently running process has a run time that is equal to the stipulated time quanta
        - If the so, replace it with a new process:
            - Peek at the process at the front of the queue
            - If the arrival time of that process is either equal to or is past the current time index, then pop that process and set it as the currently running process; also then set the variable storing the amount of time the currently running process has been running to 0
            - The process that was just running must then be updated and checked; if its remaining execution time is not 0, it must be re-added to the Priority Queue[1]
            - However, if the arrival time of the process at the start of the Priority Queue (determined via the peek method) is greater than the current time index, then you should continue running the current process for another time quanta

Note that you are to utilise proper UI principles when outputting all necessary messages for this simulation. Your messages should indicate whenever a new process starts or stops running, and also indicate when a process can been completed.

[1] Determining a new arrival time must be based on the current time index + the time quanta; by doing this you will ensure that the reinserted process does not cut in front of any processes that are already scheduled to run.