BLG 223E- Data Structures
Homework #1

Due: October, 27th 02:00

Submission Rules

- If explanations for this homework are not clear, you can ask your question on the message board for BLG 223E on NINOVA. Please check before writing your question whether your question has been asked by someone else. You can also contact T.A. Abdullah Sadık Satır – (**satir17@itu.edu.tr**)
- Make sure you write your name and number in all of the files of your project in the following format:
  /* @Author
  Student Name: $< student\_name >$
  Student ID : $< student\_id >$
  Date: $< date >$*/
- Academic dishonesty, including cheating, plagiarism, and direct copying, is unacceptable.
- Note that YOUR CODE WILL BE CHECKED WITH THE PLAGIARISM TOOLS!
- Use comments wherever necessary in your code to explain what you did.
- Your program should compile and run on a Linux environment. You can (and should) code and test your program on Docker's Virtual Environment. Do not use any pre-compiled header files or STL commands. Be sure that you have submitted all of your files.
- Only electronic submissions through Ninova will be accepted no later than the deadline.
- You may discuss the problems at an abstract level with your classmates, but you should not share or copy code from your classmates or the Internet. You should submit your own individual homework.

Implementation Notes

The implementation details below are included in the grading:

1. Please follow a consistent coding style (indentation, variable names, etc.) with comments otherwise, you will get **minus** points.
2. Make sure that there is no memory leak in your code; otherwise, you will get **minus** points.
3. Since your code will be tested automatically, make sure that your outputs match with the sample scenario for given inputs. You will be provided with a calico file to check your assignment output.

Definition

For this assignment, you are expected to simulate a single processor system that can execute jobs in the determined order by the input file. A multi-level scheduler will determine the giving order of **three types of process** to the processor. The rules determined in the proceeding sections will reveal the multi-level scheduler. Expectation details in your implementations are

- You should implement 4 classes; **ProcessRep**, **FIFORep**, **CPU - CPURep**, **Scheduler - SchedulerRep**.

- **ProcessRep** class should contain the process property and necessary part of your data structure.
- **FIFORep** class contains your linked list data structure used in the FIFO approach. The class adds a new item to only the end of the list and removes an item from the beginning of the list.
- **CPU - CPURep** class list all the finished process with your **FIFORep** data structure. If the process runs in the CPU, the class decreases its **Process Time** every step.
- **Scheduler - SchedulerRep** class accepts the arriving process, lists and schedules them and sends it to CPU when its turn.

1 Input Files

You are given some test cases as **.txt** file (**case1.txt**, **case2.txt**, ...). In these files, every line represents the jobs. Every column represents the various property of the jobs. The columns are **arriving time**, **process type**, **process ID**, **process time** respectively. When reading a job from a **.txt** file, if the simulation reaches the process' arrival time, then the process is involved in the simulation by sending it to the scheduler. The mean of all the columns can be listed as follows;

- **Arriving time (1st column)**, the time when the process is ready to run, and the scheduler saves the process to its own order for running. It always is increasingly ordered from top to bottom and starts with zero (0).
- **Process type (2nd column)** defines 3 kinds of processes **(A,B,C)** and also a **PRINTALL** instruction.
  - If the simulation encounters a processes type **(A, B, C)**, you have to send the process, which is read from the line, to the scheduler.
  - If the simulation encounters **PRINTALL** instruction, you have to print all the processes that are arrived into the scheduler. (The print function is given to you in the **main.cpp** file.)
- **Process ID (3rd column)** defines the process number, which is assigned independently for every process type. It also increases one by one from 1.
- **Process Time (4th column)** defines the process burst time, which is the amount of time required by a process for executing on the CPU.

2 Data Structure (FIFO)

For the FIFO approach, arriving new item is added to the end of the list and an item to be removed is dequeued from the beginning of the list. A data structure that follows this FIFO approach can be constructed by using the linked list. The following two subsections determine two classes that you need to generate the linked list with the FIFO approach.

2.1 ProcessRep

ProcessRep class containing process properties is a node for linked list data structure applying the FIFO approach.

Private Attributes:

- **mpNext:** Point to the next Process in the linked list.
- **mProcessType:** Process type which can be **A**,**B** and **C**.
- **mProcessID:** Process ID.
- **mDeliverTime:** Process arriving time.
- **mProcessTime:** CPU usage time.

__Public Attributes:__

- **remainingTime:** Remaining time to finish the process.
- **startTime:** Time when the process first utilizes CPU.
- **endTime:** Time when the remaining time becomes zero. (Also means the time when the process finishes its work.)

Info: **m** and **p** precede variable names for referring to private attributes of class and pointer respectively.

__Methods:__

1. **Constructor(s):** You are expected to write a default constructor to initialize all values.
2. **Getter(s):** You can (and should) implement getter methods for all private attributes (**getNext()**, **getProcessType()**, **getDeliverTime()**, ...).
3. **Setter(s):** You can (and should) implement setter methods for all private attributes (**setNext()**, **setProcessType()**, **setDeliverTime()**, ...) .

Info: Setting and getting private attributes outside of classes, **Getter(s)** and **Setter(s)** methods are used.

2.2 FIFORep

FIFORep class uses the FIFO approach to generate data structure. It holds the head and tail of the list. Adds items to the end of the list and removes items from the beginning of the list.

__Private Attributes:__

- **mpHead:** The First pointer (head) of the linked list.
- **mpTail:** Last pointer (tail) of the linked list.

__Methods:__

1. **Constructor(s):** It has two constructors; one for assigning the private attributes as NULL other for assigning a pointer to these attributes.
2. **Getter(s),Setter(s):** You can (and should) implement getter and setter methods for all private attributes.
3. **queue:** adds a new process to the end of the process list.
4. **dequeue:** removes and returns the first process in the list.
5. **searchID:** search the list for a given process type and ID.
6. **printFIFO:** print all items in the list with the following format.

   <Process Type><ID><" ">**<start Time>**<" ">**<Finish Time>**<" ">

3 CPU - CPURep

CPURep class list all the finishing processes in the **mFinishedProcess** variables, and it also acts like it runs the process by decreasing **remainingTime** one.

Private Attributes:

- **mFinishedProcess:** It is an object of **FIFORep** which keeps the completed process.

Methods:

1. **Constructor(s):** One constructor for assigning private attributes.
2. **runCPU:** It execute the process by subtracting one from **remainingTime**. If process finishes, it add to **mFinishedProcess**.
3. **getFinishedProcess:** returns with **mFinishedProcess**.
4. **setFinishedProcess:** defines a new FIFO for **mFinishedProcess**.

4 Scheduler - SchedulerRep

SchedulerRep class organizes all the process inputs-outputs and CPU runs. For the organization of the process order, you have to follow the below rules;

- Scheduler has to organize 3 types of processes whose quantum (Q) differs from each other. It means that the scheduler has to keep the tree level (l) of FIFO whose quantum time changes according to the formula $(Q = 2^l)$.
- Every process should be written into their FIFO level.
- Quantum time of three levels and corresponding process types with FIFO index is given below.
  - level(l) = 1 keeps the process type **A** with Q = 2 (**mpProcessFIFO**[0]).
  - l = 2 keeps the process type **A** with Q = 4 (**mpProcessFIFO**[1]).
  - l = 3 keeps the process type **A** with Q = 8 (**mpProcessFIFO**[2]).

  Warning: quantum (time slice): After this amount of time, although process' remaining time doesn't become zero, the process is put back in the list.

- Prioritizing among the three level of process is $A > B > C$ or $l1 > l2 > l3$. These priorities mean that if the scheduler has a higher priority process, the next process that will be run is that.
- If the running process (B) is a lower priority than the arriving process (A), then the running process (B) is terminated, and arriving process (A) is sent to the CPU.
- If two processes arrive at the scheduler at the same time and need to add to the list; one from the CPU, and another from the file, the priority belongs to the one coming from the file.

Info: You can check the file (named **Scheduling Examples.pdf**) given in assignment file for scheduling example.

Private Attributes:

SchedulerRep class has

- **mpProcessFIFO[3]:** It is an 3 dimensional object array that contains three level of process from lower priority to higher.
- **mpRunningProcess:** It keeps current running process.

Public Attributes:

- **pCpuObj:** CPU object to representation of the CPU. Scheduler class send the next process to this object.
- **timeSliceCount:** This integer variable keeps track the running process' Quantum or time slice count..
- **totalTime:** The total time passed after the simulation starts.

Methods:

1. **Constructor(s):** You are expected to write a default constructor which will initialize all values.
2. **Getter(s),Setter(s):** You can (and should) implement getter and setter methods for all private attributes.
3. **checkTimeSlice:** It check the quantum time of running process. If it reaches the limits return true.
4. **sendProcessToCPU:** It sends process to **pCpuObj** object.
5. **schedule:** It takes the process properties and organize the process according to given rules.
6. **schedule:** It is a function overload and also copy of the previous schedule function. Just its arguments is changed.It is called only with NULL.

> **Notice:** You are already given a skeleton code. Please examine it before writing your own codes. Skeleton code for classes (header file) includes all necessary classes and variables. You are expected to implement the methods of these classes. You are also given the read file and print functions. You don't need to implement them.

5 How to compile, run and test your code

If you want to compile and run the provided code on terminal, you can use these commands:

```
g++ -Wall -Werror -g main.cpp SchedulerRep.cpp ProcessRep.cpp FIFORep.cpp CPURep.cpp -I ../include -o ../bin/main
../bin/main ../case1.txt
```

You can (and should) run the calico file on terminal to check your assignment with the command:

```
calico testcalico.yaml --debug
```

6 Submission

Submit all the files which you are given to Ninova. Before submitting, please make sure you are passing all cases in the calico file. The Calico file you received is not the grading file, so taking full points does not correspond to a full grade for your assignment. Passing all cases will show you that your output is compatible with the grading file. Otherwise, you might not receive a full grade for the assignment.