ITU Computer Engineering Department
BLG 223E Data Structures, Fall 2020
Homework #3

Problem Definition

A CPU(Central Processing Unit) is the main component that executes programs in a computer. Usually, there would be many programs who would like to be processed and this creates a resource allocation problem. Operating systems(OS) decide which and when a program to be executed by granting the access to CPU and create an execution order. To execute a program, first it is expressed as a process and it is added to the waiting list where it will wait OS to grant them CPU time. When OS grants CPU time to processes, they can get executed as long as the allocated CPU time. Waiting list would include many processes and OS is responsible to grant them fair amount of time of execution. OS is also responsible to grant CPU time to processes with higher priority faster than others while ensuring that processes with low priority does not wait indefinitely. Operating system solves this problem with scheduling algorithms.

In this homework, you are expected to solve the scheduling problem of an experimental operating system using MultiQueue. In this experimental operating system, a process includes sub tasks which are the individual sub programs that should not be interrupted. Because of this property, when a process has the access to the CPU, it executes one sub task before leaving the CPU free.

A sub task has the following properties:

- Name
- Duration

To finish its execution, a process has to execute all the sub tasks it has with an order. The order is represented with a Stack structure. When given CPU time, process executes the top sub task of the Stack and leaves the CPU free. To complete the execution, a process needs to be granted CPU time as much as number of the sub tasks.

A process has the following properties:

- Name
- Priority
- Arriving Time
- Deadline
- Sub Tasks Stack

A process would be initialized at Arriving Time and would like to finish the execution before the Deadline. Deadline is calculated with the following formula:

$$Deadline = ArrivingTime + \sum SubTaskDuration$$

Lateness of a process is the amount of time that past from the deadline when a process completes all of its sub tasks.. It can be calculated with the following formula:

$$Lateness = CompletionTime - Deadline$$

For scheduling processes, you are expected to use a MultiQueue with 3 Queues. Each Queue will represent different priorities of a process. Whenever a process arrives(t = Arriving Time), it will be queued to the proper Queue depending on its priority.

In this operating system, there will be 3 priorities and scheduling will be done by following the rules below:

- Priority 1 always gets the access to CPU.
- Priority 2 is more prior than Priority 3 if the special condition is not active.
- Special Condition: For fairness, Priority 3 will be more prior than Priority 2 when special condition is active. Special condition is activated if Priority 2 processes accessed the CPU two times from the last time the special condition was active or from the start. After CPU got accessed by Priority 3 process once, special condition is deactivated. If there is no process waiting with Priority 3 when the special condition is active, special condition is deactivated.

While scheduling, your program is expected to print the process name and the sub task name on each execution. When all process are scheduled, your program is expected to print the lateness of the created schedule.

Workflow

1. Data file will be given to the program as a command line input. Assuming the data is in "data.txt" file, it includes the properties of processes. Each process will be seperated by a new line. Properties of a process in data.txt will have the following pattern:

   process_name priority
   arriving_time number_of_subtasks
   subtask_name duration
   ...

   The processes in the data.txt will always be sorted by *arriving_time* and the *arriving_time* of the first process will always be 0.

2. Read the data.txt file into a Queue. This Queue will hold the processes that will arrive. Then create a MultiQueue. Start the time from 0.
3. Insert the process or processes into the MultiQueue for scheduling when they should arrive depending on the time.
4. Determine which process in the MultiQueue should get the CPU time. Run a sub task of it and change the current time. Print the *process_name* and *subtask_name*.
5. If the process does not have any subtask left, calculate the lateness and remove the process from the MultiQueue.
6. Repeat 3-5 until all the processes are scheduled.
7. Print the cumulative lateness.

Example Run

Input file ”data.txt”

```
process1 2
0 3
A 6
B 9
C 15

process2 3
5 2
A 1
B 3

process3 1
7 1
A 5

process4 2
8 3
A 6
B 9
C 5

process5 3
8 3
A 6
B 9
C 2

process6 1
15 1
A 5
```

Expected Output

```
process1 C
process3 A
process6 A
process1 B
process2 B
process1 A
process4 C
process2 A
process4 B
process4 A
process5 C
process5 B
process5 A
Cumulative Lateness: 158
```

Implementation

Implement the following structures and functions given in the structure in a cpp file with name data_structs.cpp and in a header file with name data_structs.h.

```
struct Process{
string name;
int arrival_time;
int deadline;
int task_count;
int priority;
Stack task_stack;
Process* next;
};

struct Subtask{
string name;
int duration;
Subtask* next;
};
struct Stack{
Subtask* head;
void init();
void close();
void push(Subtask* in);
Subtask* pop();
bool isEmpty();
};

struct Queue{
Process* head;
Process* tail;
void init();
void close();
void queue(Process* in);
Process* dequeue();
bool isEmpty();
Process* front();
};

struct MultiQueue{
Queue queues[3];
void init();
void close();
void queue(Process* in);
Process* dequeue(int priority);
bool isEmpty();
Process* front(int priority);
};
```

Although the parameters and return types of the functions are specified above, you can use different parameters and return types in these functions. You may also add extra functions when necessary.

Implement the main function and additional functions in a cpp file with name scheduler.cpp..

Submission Rules

- Make sure you write your name and ID number in all of the files of your project, in the following format:

  /* @Author
  Student Name: <student_name>
  Student ID : <student_id>
  Date: <date> */

- Use comments wherever necessary in your code to explain what you did.
- You are not allowed to include any Standard Template Library (STL) container.
- To develop your implementation, you can use any IDE (Integrated Developmet Environment) such as Visual Studio etc. But before submitting, your program should be compiled and ran on Linux environment using **g++ (version 4.8.5 or later)**. You can test your program on ITU's Linux Server using SSH protocol. To compile the code, use the following command:

  ```
  g++ -std=c++11 -Wall -Werror data_structs.cpp scheduler.cpp -o scheduler
  ```

  And you can execute your program by using the following command:

  ```
  ./scheduler input.txt
  ```

  Your program will be checked by using **Calico(https://bitbucket.org/uyar/calico)** automatic checker. Before submitting check your code using calico with the given test file. Make sure that your code can get 10 points from the calico with given test file.

- If explanations for this homework is not clear, you can ask your question under the thread that is specially started for homework 3 (About HW3) on the message board for BLG 223E on NINOVA. Please check this thread before writing your question whether your question is asked by someone else.

  Do not share any code or text that can be submitted as a part of an assignment

- Only electronic submissions through Ninova will be accepted no later than 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 from the Internet. You should submit your **own, individual** homework.
- Academic dishonesty, including cheating, plagiarism, and direct copying, is unacceptable.
- A different input file will be used for evaluating the homework.
- Note that **YOUR CODES WILL BE CHECKED WITH THE PLAGIARISM TOOLS!**