Lab : Queues

Information

In-class labs are meant to introduce you to a new topic and provide some practice with that new topic.

Topics: Queues, Scheduling schemes

Solo work: Labs should be worked on by each individual student, though asking others for help is permitted. Do not copy code from other sources, and do not give your code to other students. **Students who commit or aid in plagiarism will receive a 0% on the assignment and be reported.**

Building and running: If you are using Visual Studio, make sure to run with debugging. **Do not run without debugging!**

Using the debugger will help you find errors.
To prevent a program exit, use this before `return 0;`

```
cin.ignore();    cin.get();
```

Turn in: Once you're ready to turn in your code, prepare the files by doing the following: **(1)** Make a copy of your project folder and name it `LASTNAME-FIRSTNAME-LABNAME`. (Example: `HOPPER-GRACE-LAB-UNIT-TESTS`) **(2)** Make sure that all source files (.cpp, .hpp, and/or .h files) and the `Makefile` files are all present. **(3)** Remove all Visual Studio files - I only want the source files and Makefiles. **(4)** Zip your project folder as `LASTNAME-FIRSTNAME-LABNAME.zip`

Never turn in Visual Studio files!

Starter files: Download from GitHub.

Grading: Grading is based on completion, if the program functions as intended, and absense of errors. **Programs that don't build will receive 0%.** Besides build errors, runtime errors, logic errors, memory leaks, and ugly code will reduce your score.

Contents

1.1 About

1.1.1 Queues

A queue is a restricted-access data type that is "FIFO" (First-in, First-out). Like with a line at Micro Center, new nodes in the queue enter from the back of the "line", and the node at the front of the queue will leave it next once `Pop()` is called.

front	0	1	2	back

1. Empty queue

front	0	1	2	back
	A			

2. Pushed "A" into queue

front	0	1	2	back
	A	B		

3. Pushed "B" into queue

front	0	1	2	back
	B			

4. Pop removes "A"

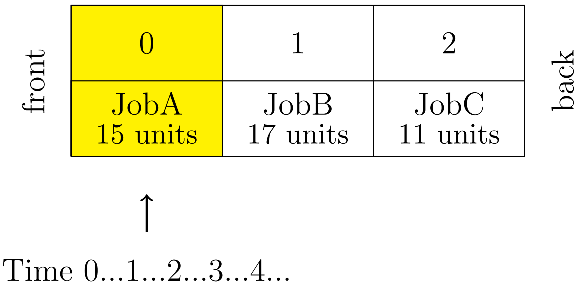
1.1.2 Scheduling

First come, first served: The first processors were simple and only allowed one program to run at a time, and everything else had to wait for it to complete before getting a chance to run. This can be considered "First Come, First Served" scheduling. This is also known as FIFO, since it is basically a vanilla queue.

Round Robin: Another way to handle scheduling, so that multiple processes can run is with "Round Robin". In this manner, you choose some *timeout*. Once the timer hits the value you selected, it moves the item currently being worked on to the back of the processing queue and spends time on the next thing.

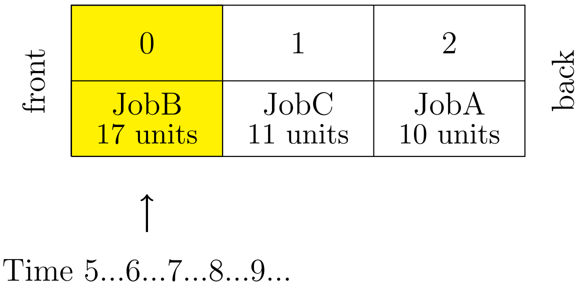
Let's say that we have a job queue, and ever 5 time units (you can think of seconds, but it would be *much, much* faster than that)

A job queue, with interval = 5 units



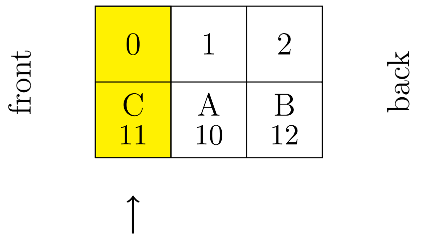
0	1	2
JobA 15 units	JobB 17 units	JobC 11 units

The processor would work for 5 units of time, then move JobA to the back of the queue. JobA's remaining time is now 10 units.



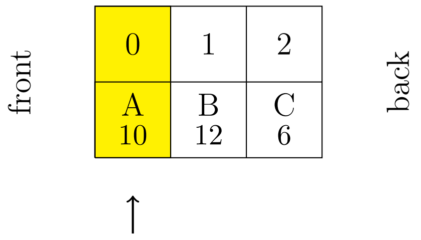
0	1	2
JobB 17 units	JobC 11 units	JobA 10 units

The process continues...



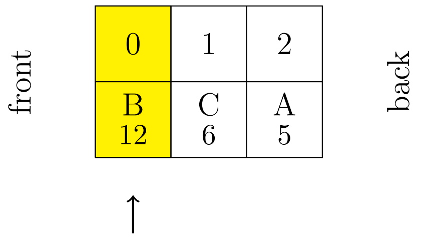
0	1	2
C 11	A 10	B 12

Time 10...11...12...13...14...



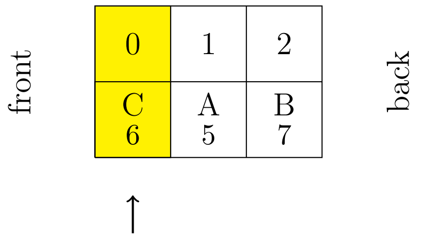
0	1	2
A 10	B 12	C 6

Time 15...16...17...18...19...



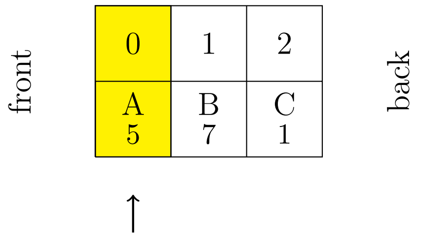
0	1	2
B 12	C 6	A 5

Time 20...21...22...23...24...



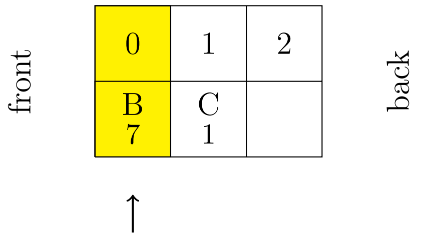
0	1	2
C 6	A 5	B 7

Time 25...26...27...28...29...



0	1	2
A 5	B 7	C 1

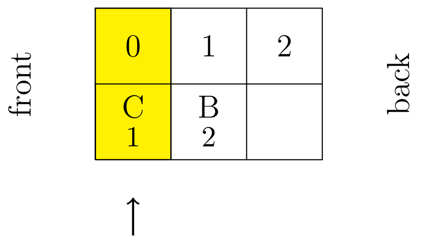
Time 30...31...32...33...34...



0	1	2
B 7	C 1	

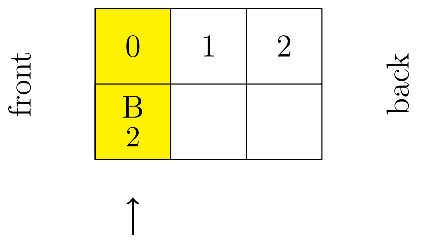
Time 35...36...37...38...39...

Once a process is done being processed, it is removed from the queue, and processing can continue with the remaining items.



0	1	2
C 1	B 2	

Time 40...41...42...43...44...



0	1	2
B 2		

Time 45...46...47...48...49...

More info: You can learn more about scheduling algorithms at Wikipedia: https://en.wikipedia.org/wiki/Scheduling_(computing)

1.2 Lab specifications

In this lab, you will have a list of Jobs to process. You will have a FCFS (First Come, First Served) Queue and a RR (Round Robin) Queue. Each of these will go through the job and "process" it.

1.2.1 Queue

Here we are going to implement the Queue using the LinkedList class included. Instead of doing it as **inheritance** (is-a relationship), we are going to use **composition** (has-a relationship).

First, give the Queue a private templated LinkedList item:

```
LinkedList<T> m_list;
```

Then each of the Queue's functions should do the following:

Push	Call `m_list.PushBack( data );`
Pop	Call `m_list.PopFront();`
Front	`return m_list.GetFirst();`
Size	`return m_list.Size();`

1.2.2 The Job struct

The Job struct looks like this:

```
struct Job
{
    Job();
    void Work( JobType type );
    void SetFinishTime( int time, JobType type );

    int id;

    int fcfs_timeRemaining;
    int fcfs_finishTime;
    bool fcfs_done;

    int rr_timeRemaining;
    int rr_finishTime;
    bool rr_done;
    int rr_timesInterrupted;
};
```

The `Work` function just counts down on the `timeRemaining` (either the fcfs or rr version, depending on what you pass in.) For example, you would process the next item in the FCFS queue like this:

```
jobQueue.Front()->Work( FCFS );
```

Once the time remaining hits 0, the Job will have its `done` boolean set to true, which can be used from the *processor* side to decide to remove it from the job queue.

```
jobQueue.Front()->SetFinishTime( cycles, FCFS );
jobQueue.Pop();
```

For the Round Robin processor, any time the job runs out of time and focus is given to a new job, you'll also want to keep track of how many times the job gets interrupted.

```
jobQueue.Front()->rr_timesInterrupted += 1;
```

1.2.3 The Processor

The processor is just a wrapper for two functions...

```
class Processor
{
    public:

    void FirstComeFirstServe( vector<Job>& allJobs,
        Queue<Job*>& jobQueue, const string& logFile );

    void RoundRobin( vector<Job>& allJobs,
        Queue<Job*>& jobQueue, int timePerProcess,
        const string& logFile );
};
```

You will implement these two functions. The parameters are...

- `vector<Job>& allJobs` The list of all jobs being used; can be used within the function to display a list of all jobs and their info.
- `Queue<Job*>& jobQueue` The queue of jobs waiting to be processed. You will be working with this structure, calling the Work() function on the front-most Job to process.
- `int timePerProcess` (Round Robin) This is the time increment for the Round Robin scheduler. After n units of time have passed, the current item is put at the back of the queue so a different job can have a chance to process.
- `const string& logFile` A string filename where the output file should be written to. You can display output with `cout`, but you should also be writing a report to an output text file.

Opening and writing to an output file

```
ofstream output( logFile );
output << "First Come First Served (FCFS)" << endl;
// ...
output.close();
```

Processor::FirstComeFirstServe

While the `jobQueue` is not empty...

- Process the front-most item.
- If the front-most item's `done` variable (for this type - `fcfs_done` or `rr_done`), then...
  - Set the front-most item's finished time via the `SetFinishTime` function.
  - Pop the item off the queue.
- Increment the cycle counter.

You will want to keep track of cycles, as a simple int counter. Each cycle is a unit of time. Increment the cycle counter after each iteration of the loop. Also display the following information to the output text file (not the cout): The current cycle, The current job's ID, and the amount of time the job has remaining.

Once the queue is empty, you will also display some result statistics about the processing. You can still access all the finished jobs via the `allJobs` vector. Display each job's ID and time to complete the job. As a summary, display the average time to complete all the jobs, and the total processing time.

See the **Example Output** for more.

Processor::RoundRobin

For this one, you'll keep a separate `cycles` counter, as well as a `timer`, which will keep track of the round time.

While the `jobQueue` is not empty...

- If the timer has hit the `timePerProcess` value...
    - Increment the front-most job's `r_timesInterrupted` by 1.
    - Push the front-most job to the back of the queue.
    - Pop the job off the front of the queue.
    - Reset your `timer` to 0.
- Process the front-most item.
- If the front-most item's `done` variable (for this type - `fcfs_done` or `rr_done`), then...
    - Set the front-most item's finished time via the `SetFinishTime` function.
    - Pop the item off the queue.
- Increment the cycle counter and the timer counter.

Once again you'll need to write the current cycle, the job's ID, the remaining time, and the amount of times the job has been interrupted so far to the text file.

Once the job queue is empty, you will output a summary. You can still access all the finished jobs via the `allJobs` vector. Display each job's ID, time to complete the job, and amount of times it was interrupted. As a summary, display the average time to complete all the jobs, the total processing time, and the round robin interval.

See the **Example Output** for more.

1.3 Example output

1.3.1 Running the program

```
-------------------------------------------------
 | Job Processor |
 -----------------

-------------------------------
 How many jobs? (More than 10)

 >> 20


-----------------------------
 Round Robin time interval?

 >> 5


Creating jobs...
Job 0, fcfs: 122, rr: 122
Job 1, fcfs: 93, rr: 93
Job 2, fcfs: 135, rr: 135
(etc)
Job 18, fcfs: 113, rr: 113
Job 19, fcfs: 102, rr: 102

Filling queues...

Processing with FCFS...

Processing with RR...

DONE
```

1.3.2 result-fcfs.txt

```
First Come First Served (FCFS)
Processing job #0...
    CYCLE  0         REMAINING:     94   ...
    CYCLE  1         REMAINING:     93   ...
    CYCLE  2         REMAINING:     92   ...
    CYCLE  3         REMAINING:     91   ...
    CYCLE  4         REMAINING:     90   ...
    CYCLE  5         REMAINING:     89   ...
    CYCLE  6         REMAINING:     88   ...
    CYCLE  7         REMAINING:     87   ...
    CYCLE  8         REMAINING:     86   ...
    CYCLE  9         REMAINING:     85   ...
    CYCLE  10        REMAINING:     84   ...
    CYCLE  11        REMAINING:     83   ...
    CYCLE  12        REMAINING:     82   ...
    CYCLE  13        REMAINING:     81   ...
    CYCLE  14        REMAINING:     80   ...
    CYCLE  15        REMAINING:     79   ...
    CYCLE  16        REMAINING:     78   ...
    CYCLE  17        REMAINING:     77   ...
    CYCLE  18        REMAINING:     76   ...
(etc)
    CYCLE  93        REMAINING:     1    ...
    CYCLE  94        REMAINING:     0    ...
Done



--------------------------------------------
Processing job #1...
    CYCLE  95        REMAINING:     101  ...
    CYCLE  96        REMAINING:     100  ...
    CYCLE  97        REMAINING:     99   ...
    CYCLE  98        REMAINING:     98   ...
    CYCLE  99        REMAINING:     97   ...
    CYCLE  100       REMAINING:     96   ...
    CYCLE  101       REMAINING:     95   ...
    CYCLE  102       REMAINING:     94   ...
    CYCLE  103       REMAINING:     93   ...
(etc)
    CYCLE  2026      REMAINING:     5    ...
    CYCLE  2027      REMAINING:     4    ...
    CYCLE  2028      REMAINING:     3    ...
    CYCLE  2029      REMAINING:     2    ...
    CYCLE  2030      REMAINING:     1    ...
    CYCLE  2031      REMAINING:     0    ...
Done
```

```
---------------------------------------------------------
---------------------------------------------------------


First come, first serve results:

JOB ID    TIME TO COMPLETE
0         94
1         196
2         308
3         400
4         500
5         569
6         623
7         677
8         824
9         889
10        980
11        1088
12        1213
13        1287
14        1421
15        1524
16        1671
17        1756
18        1899
19        2031

Total time: ............. 2032
    (Time for all jobs to complete processing)

Average time: ........... 997.5
    (The average time to complete, including the wait time
   while items
    are ahead of it in the queue.)
```

1.3.3 result-rr.txt

```
Round Robin (RR)
Processing job #0...
    CYCLE  0         REMAINING:      94   ...
    CYCLE  1         REMAINING:      93   ...
    CYCLE  2         REMAINING:      92   ...
    CYCLE  3         REMAINING:      91   ...
    CYCLE  4         REMAINING:      90   ...

-----------------------------------------------
Processing job #1...
    CYCLE  5         REMAINING:      101  ...
    CYCLE  6         REMAINING:      100  ...
    CYCLE  7         REMAINING:      99   ...
    CYCLE  8         REMAINING:      98   ...
    CYCLE  9         REMAINING:      97   ...

-----------------------------------------------
Processing job #2...
    CYCLE  10        REMAINING:      111  ...
    CYCLE  11        REMAINING:      110  ...
    CYCLE  12        REMAINING:      109  ...
    CYCLE  13        REMAINING:      108  ...
    CYCLE  14        REMAINING:      107  ...

-----------------------------------------------
Processing job #3...
    CYCLE  15        REMAINING:      91   ...
    CYCLE  16        REMAINING:      90   ...
    CYCLE  17        REMAINING:      89   ...
    CYCLE  18        REMAINING:      88   ...
    CYCLE  19        REMAINING:      87   ...

-----------------------------------------------
Processing job #4...
    CYCLE  20        REMAINING:      99   ...
    CYCLE  21        REMAINING:      98   ...
    CYCLE  22        REMAINING:      97   ...
    CYCLE  23        REMAINING:      96   ...
    CYCLE  24        REMAINING:      95   ...
(etc)

-----------------------------------------------
Processing job #8...
    CYCLE  2025      REMAINING:      6    ...
    CYCLE  2026      REMAINING:      5    ...
    CYCLE  2027      REMAINING:      4    ...
    CYCLE  2028      REMAINING:      3    ...
    CYCLE  2029      REMAINING:     2    ...

---------------------------------------------
Processing job #8...
    CYCLE  2030      REMAINING:     1    ...
    CYCLE  2031      REMAINING:     0    ...


--------------------------------------------------------
--------------------------------------------------------



Round Robin results:

JOB ID    TIME TO COMPLETE    TIMES INTERRUPTED
0         1644                18
1         1801                21
2         1838                22
3         1651                18
4         1766                20
5         1298                13
6         1033                10
7         1127                11
8         2031                33
9         1224                12
10        1720                19
11        1813                21
12        1933                25
13        1398                14
14        1996                28
15        1787                20
16        2021                30
17        1564                16
18        2017                29
19        1971                26

Total time: ............. 2032
    (Time for all jobs to complete processing)

Average time: ........... 1681.65
    (The average time to complete, including the wait time
   while items
    are ahead of it in the queue.)

Round robin interval: ... 5
    (Every n units of time, move the current item being
   processed
    to the back of the queue and start working on the next
   item)
```