Part 1: Scheduling

Team Member

shengbo lou: 28530995
zhongce ji: 28551884
yucheng lu: 28411887

Compile

Go to the lab02_part1 folder

- compile: `javac lab/*.java`
- run: `java lab/Main`

Design

We use the java language to complete the first part of the lab2.
Our implementation of the scheduler that supports the first come first served (FCFS) and round robin (RR) scheduling algorithms starts with the Job class, which represents each job in the input file.

- Job.java
  This file have a class which is called Job this class have constructor which will take 2 arguments, one is total which is total execution time of the job the other one is the arrvialTime which is the arrive time of the job.

```
public Job(int total,int arrvialTime){
  this.total=total;
  left = total;
  this.arrvialTime = arrvialTime;
  CompletionTime = 0;
}
```

We initialize a list of job called `jobs` contains all the jobs from a single input file. The job class has a attribute left which represents its remaining running and we can calculate the waiting time of a job by calculating:

```
total_wating_time += (jobs[i].getCompletionTime() - jobs[i].getArrvialTime() -
jobs[i].getTotal());
```

and the average waiting time by:

```
int average_waiting_time = (int)total_wating_time/compeleted_jobs;
```

At the end of the simulation time, we calculate the waiting time for every jobs in the readyqueue and get the average waiting time by dividing the total waiting time of all the jobs by the number of jobs in the readyqueue and completed jobs.
This is the way that we get our solution.

We have an important global variable, current_time, which simulates the running
time of one second of CPU and represents one second. We use the current_time to
determine when the job should be added into the readyqueue and set the completion time
a job when it is completed. So the program will be running under a while loop:
while(curr_time <=Simulation_Time)
There are two scheduling algorithms:

- First_Come_First_Serve.java

First_Come_First_Serve Scheduler will be simple then the Round_Robin,
because it will execute the job who comes first. We have some variables:
curr_time: int , this will record the current running time.
readyQueue: Queue<Job>, this will store the jobs that need to be executed.
In the while loop of First_Come_First_Serve class, we check if there is any
job coming at each second:

```
for (Job job : jobs) {
  if (job.getArrvialTime() == curr_time) {
    readyQueue.add(job);
  }
}
```

We add the current_time by one while processing each
job in the readyqueue for one second. If the job is completed, we remove the job
and start executing next job in the readyqueue. If the readyqueue is empty and
simulation time is not reached, we simply add current_time by one.

- Round_Robin.java

we assume the process time is 1. So every job will work 1 time and it will be put into
a readyQueue. Then we continue to do this until the Simulation_Time. We have some variables:
compeleted_jobs: int, this will record how many jobs have been completed.
readyQueue: Queue<Job>, this will store jobs that need to be waited.
first_arrival: int, this will record the first jobs' arrive time.
running_time: int, this will record the running time.
First we use scanner to get the data from the file. And then get first three ints, Then
use loop to get each jobs arrive_time and execution_time then create the jobs and put them
into a job array. Then we take the first job's arrive_time, then we check if the first job's
arrive_time is 0 then we will add jobs whose arrive time is 0 into the readyQueue. After that
we have a while loop, it will continue run until the running time reach the Simulation_Time. In
the while loop, if the readyQueue is not empty, we first remove a first jobs in the queue and
process it, add the running_time. Then we have to check at this time, if we have jobs whose
arrive_time is same as the running_time, which means at this time those jobs need to be
arrived
so we add them into the readyQueue. Then we check that if the job's left time is 0 which means
that the job is completed so we have to set the job's complication time as the current_time,
and we do not need to put the job back to the readyQueue. If the job is not completed then we
need to put the job back into the end of the readyQueue. Then if the readyQueue is empty,
we need to add the jobs whose arrive_time is current_time into the readyQueue. Then after
we jump out of the while loop, we just use the for loop to print out all jobs waiting time and
use the equation: total waiting = complication time - arrive time - execution time.

- Main.java

In this file we will print out the whole result of the lab2 In the main class,
we create Round_Robin object and print out the result of the input with Round_Robin Schedule.
Also we create First_Come_First_Serve object and print out the result of the input with First_Come_First_Serve Schedule.