CSCI3120
Assignment 1

Instructor: Alex Brodsky

Due: 11:59 AM, Tuesay, May 28, 2019

Introduction

The concepts applicable to operating systems are applicable in many other systems as well. For example, manufacturing, transportation, and bureaucracies. The air transportation system (planes and airports) is another prime example of a system that embodies many of the same concepts found in modern operating systems. If your flight has ever been delayed, canceled, or rescheduled, it was probably due as a result of the same decisions that are made by the OS on your computer.

The purpose of this assignment is

1. To refresh your coding abilities in C, and your programming skills in general.
2. To review using and implementing queues and priority queues, which are at the heart of every operating system
3. To serve as an example of the difficulty, style, and format of assignments that you can expect in this course.
4. To practice reading and implementing nontrivial specifications. This assignment is as much about reading and understanding this document, as it is about programming.

In this assignment you will implement a simple air transportation system simulator.

System Requirements

This assignment is intended to be done and executed in a Unix environment. You can use the `csci3120.cs.dal.ca` Unix server, a Mac (in *Terminal*), or your own Unix-based system, such as Linux, OpenBSD, FreeBSD, etc. This may also work in the Cygwin environment under Windows. **Note: Your code will be tested on `csci3120.cs.dal.ca`.**

Background

Scheduling flights is a challenging prospect. There are 100,000 flights per day among more than 17,000 airports. A good schedule minimizes the risk of delayed flights and the amount of time planes are sitting useless on the ground. A simulation of the air transportation system with a given schedule can be used to assess the delay and aircraft utilization by simulating all the scheduled flights.

In its basic form our air transport system comprises airports and planes that fly between airports, ferrying passengers. A flight consists of a plane that

1. departs from a gate at a specific airport and a scheduled time;
2. waits for permission to take off;
3. takes off;
4. flies to its destination;
5. waits for permission to land;
6. lands; and
7. taxis to the gate

The amount of time the plane spends flying depends on the distance between the airports and weather conditions; the amount of time waiting for permission to take-off or land depends on how busy the airport is; the amount of taxiing between the runways and gates depends on the size of the airport. A schedule needs to take all these things into account.

A flight is denoted by

Carrier Code: a two letter designator for the airline, e.g., AC for Air Canada

Flight Number: a positive integer denoting the flight number, e.g., 612

Origin: the three letter IATA airport code where the flight commences, e.g., YHZ for Halifax International Airport

Departure Time: scheduled time for departing from the gate in the 24-hour format *hh:mm*, e.g., 15:35, means a departure time of 3:35pm.

Expected Flight Time: amount of time (in minutes) spent flying from take-off to landing, e.g., 120

Destination: the three letter IATA airport code where the flight terminates, e.g., YYZ for Toronto International Airport

A flight schedule consists of 1 or more flights.

A basic simulation of a flight schedule simulates each minute of a single day. In each minute one of the following events can occur in *each* of the scheduled flights:

1. A flight may depart from the gate at its origin
2. A flight may take off
3. A flight may arrive at its destination
4. A flight may land
5. A flight may arrive at the gate of its destination

The flight terminates when it arrives at the gate of its destination.

One of the primary causes of delays is when multiple flights compete for the same resource, specifically. the runway. Only one flight may take off or land on the same runway per minute. Thus, if multiple flights are ready to take-off from the same runway at the same time, they will need to wait their turn for the use of the runway. Similarly, if multiple planes arrive at the same time, they will need to wait in a holding pattern for their turn to land on the same runway. If planes are waiting to take off and land on the same runway, planes take turns landing and taking off, i.e. plane A lands, plane B takes off, plane C lands, etc.

Once a flight terminates, the number of minutes that it was delayed can be computed. Once all flights have terminated, the simulation completes.

Your task will be to implement an air traffic simulator called `atsim`.

Simulator Input

Input is read in from the console (`stdin`). The input consists of one or more lines of text. Each line of text, except the last one represents a scheduled flight. Each flight has the following form:

$C\ N\ P\ O\ H{:}M\ T\ D$

where

C is the carrier code, e.g., `AC`
N is the flight number, e.g., `612`
P is the plane ID, a positive integer for future assignments, e.g., `42`
O is the origin, denoted by the three letter IATA airport code, e.g., `YHZ`
$H : M$ is the departure time in the 24 hour format, e.g., `15:35`
T is the flight time (in minutes), e.g., `120`
D is the destination, denoted by a three letter IATA airport code , e.g., `YYZ`

For example,

```
AC 621 42 YHZ 15:35 120 YYZ
```

The last line of the input is a single word "`end`" that denotes the end of the schedule. There are many examples in the test suite that is included as part of this assignment.

Processing and Semantics

- There will be no more than 1000 flights. All flights are guaranteed to terminate on the same day, so there is no need to worry about the time wrapping around.
- A flight takes 10 minutes to taxi from the gate to the runway and to taxi from the runway to the gate after it has landed. So, a flight that takes 60 minutes of flight time and is not delayed will take a total of 80 minutes gate to gate.
- Each airport is has its own runway. The runway can handle one flight per minute, either landing or taking off.
- Flights that are waiting to takeoff at the the same airport are processed in first-come-first-served order. If two or more flights request to take-off at the same time (in the same minute), the flight with the lowest flight number has priority. I.e., ties are broken by the flight number.
- Flights that are waiting to land at the same airport are also processed in first-come-first-served order. If two or more flights request to land at the same time, the flight with the lowest flight number has priority.
- If flights are waiting to both take-off and land at the same airport, then priority alternates between take-off and landing. For example, if the most recent runway event was a take-off, then a landing flight is given priority.

- A plane's flight time begins at take-off and ends at landing.
- The delay of a flight is

$$\mathit{delay} = \mathit{arrival\_time} - \mathit{departure\_time} - \mathit{flight\_time} - 20$$

  where the 20 minutes is the time it takes to taxi to and from the runway.

Expected Output

All output must be performed to the console (`stdout`). The output consists of flight records, one per flight per line. The flight record has the format:

`[`h`:`m`]` C N `from` O `to` D`, departed` H`:`M`, delay` d`.`

where C, N, O, D, H, and M are as defined in the Input section; and where $h : m$ is the arrival time in the 24 hour format, e.g., `17:55`; and d is the delay (in minutes), e.g., `0`.

The flight records should be sorted in chronological order of arrival time, with tie breakers broken by the lexical order of the carrier code and then the flight number. e.g.,

```
[02:31] AC 308 from YUL to YYZ, departed 01:03, delay 0.
[02:32] AT 179 from YVR to YXE, departed 00:22, delay 0.
[02:32] CP 601 from YYC to YVR, departed 00:49, delay 0.
[02:33] AT 875 from YOW to YYZ, departed 01:15, delay 2.
```

Nonfunctional Requirements

- Please download the starting code found in the tarball `assn1.tar.gz`, which can be found on Brightspace.
- You **must** use the provided `makefile` to build your simulator simply by running `make`. You may extend the makefile as needed. Note: **The marker will use this makefile to build your assignment.**
- Included in the tarball is the `runme.sh` shell script. If you implement this assignment in Java, you will need to modify this shell script to run your Java program instead of the executable.

Test Your Code!

To test your code the tarball includes a test suite, which will run on the `csci3120.cs.dal.ca` Unix server (it will also run on other Linux and Mac systems as well). Please see the `README.txt` file in the tarball for details. The test suite works by comparing the output of your simulator to what is expected. The included `test.sh` script uses the `runme.sh` script to invoke your implementation and compare it against the expected results. Ideally, there should be no differences. If there are differences, then either your implementation, or the solution has a bug. For example, if your program fails a test 13, the input is found in `test.13.in`, the expected output is found in `test.13.gold`, and the output from your program is found in `test.13.out`. Thus, you can effectively test your implementation. **Note: We will be using similar tests to evaluate your assignment.**

Hints and Suggestions

1. Your program will rely on queues and/or priority queues. You should implement a general purpose queue and/or a general purpose priority queue that you will use in your program. Note: the solution just uses a priority queue implementation, which when used with a fixed priority behaves as a queue.
2. Use an airport struct to represent an airport. Your airport struct will need to contain a few different queues/priority queues to keep track of planes that are waiting to depart, waiting to take-off, are en route, and waiting to land. You will need to create new a airport each time you read in a flight that originates or terminates at a new airport.
3. Your main program should consists of two loops. The first loop reads in all the flights. The second loop simulates each minute of the air traffic system.
4. A starting point for your main program `atsim.c` is provided.
5. Start early, the solution totals less than 350 lines of code (100 for the priority queue and 250 for the rest). If you are not comfortable with C, it may take you a bit longer.

The Java Option

You may choose to implement this assignment in Java rather than C. However, there are two penalties for using Java. First, there is a 10 point penalty. I.e., Your grade on the assignment is reduced by 10 points. Second, the solution for Assignment 1, which is also the starting point for Assignment 2, is only provided in C. So, you will need to rely on your Java solution to Assignment 1 as a starting point for Assignment 2 if you decide to implement Assignment 2 in Java. The amount of code needed for the Java version is under 250 lines of code because the JDK already has a priority queue implementation, ready to go.

To use Java instead of C, you need to

1. Create a class called `ATSim.java` where the main program starts running.
2. Modify the `runme.sh` script to invoke your Java program instead of the C executable.

What to Hand In

You need only submit an electronic of your assignment. Submission must be done by 11:59 AM of the due date. Submission is done via Brightspace.

The assignment submission should be a single zip file (`.zip`) comprising the source code files that implement the simulator as well as the makefile and the `runme.sh` file.

Programming Guidelines

1. The program must read the input from the console (`stdin`) and must print out the results to the console (`stdout`) unless otherwise specified. Do not print out any output or request any input other than what is specified by the assignment.
2. Use the file and function names specified in the assignment.
3. Your programs will be compiled and tested on the `csci3120.cs.dal.ca` Unix server. In order to receive credit, your programs must compile and run on this server.
4. Your program must compile. **Programs that do not compile will receive a 0.**
5. The programs must be well commented, properly indented, and clearly written. Please see the style guidelines under Assignments or Resources on the course website.

Grading Scheme

The assignment will be graded based on three criteria:

Functionality "Does it work according to specifications?". This is determined in an automated fashion by running your program on a number of inputs and ensuring that the outputs match the expected outputs. The score is determined based on the number of tests that your program passes. So, if your program passes $\frac{t}{T}$ tests, you will receive that proportion of the marks.

Quality of Solution "Is it a good solution?" This considers whether the solution is correct, efficient, covers boundary conditions, does not have any obvious bugs, etc. This is determined by visual inspection of the code. Initially full marks are given to each solution and marks are deducted based on faults found in the solution.

Code Clarity "Is it well written?" This considers whether the solution is properly formatted, well documented, and follows coding style guidelines.

If your program does not compile, it is considered non-functional and of extremely poor quality, meaning you will receive 0 for the solution.

	Mark
Code Quality (Priority Queue)	/10
Code Quality (Simulator)	/10
Functionality (automated tests)	/20
Code Clarity	/10
Total	**/100**