CSCI3120
Assignment 2

Instructor: Alex Brodsky

Due: 11:59 AM, Tuesday, June 18, 2019

Introduction

As we discussed in class, process become blocked whenever they are waiting for a resource or for an operation to complete. In air-travel a similar situation occurs when a plane for a particular flight is delayed (or is not available). This is a primary cause of increasing delays as the day progresses.

The purpose of this assignment is

1. Reinforce the idea of blocking and its effect on scheduling.
2. To implement blocking mechanisms similar to ones in an operating system.
3. To further improve your programming skills.

In this assignment you will extend the air transportation simulator by adding the ability to block flights when the plane required for the flight is not available.

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

This assignment assumes all the background described in Assignment 1.

Flights require an important piece of equipment to proceed: a plane. At the start of the day, a plane is assigned to the first flights. When a flight arrives at an airport, the plane is cleaned up ("groomed") and then used for another flight departing from the same airport. When that flight ends, the plane is again groomed and assigned to another flight departing from its new location. The plane identifier (PID) is used to uniquely identify each plane and is part of the flight information.

Naturally, if a plane is delayed or breaks down, flights that rely on it may be delayed (or canceled) as well. It is a common (unpleasant) experience to be waiting at the gate and being told that the plane has not yet arrived.

Once a plane arrives, it takes at least 30 minutes to groom the plane in preparation for departure. Once the plane is groomed, the flight that will use it can depart on or after its scheduled departure time. Thus, an additional 30 minutes must be allowed between when a plane arrives and departs. **Note:** The first time a plane is used, it is already groomed and can depart at the flight's appointed time.

Your task will be to implement this aspect of scheduling in the air traffic simulator called `atsim`. You can either extend your own version, which you submitted for Assignment 1, or the provided Assignment 1 solution.

Simulator Input

The input is the same as Assignment 1. Please see examples as shown in the provided test suite (`tests_2.zip`).

Processing and Semantics

All processing rules described in Assignment 1 apply, in addition to the following:

- Each plane is identified by a plane ID that is in the range $0 \ldots 999$.
- The plane used by a flight must be at the same airport as the origin of the flight.
- Each plane is initially at the same airport as the first flight that will use it. E.g., If the first flight to use plane 42 originates in YHZ, then initially plane 42 is at YHZ.
- If a plane is being use by a flight, no other flight can use it until the flight completes and the plane is groomed.
- After a flight completes (arrives at the gate), the plane associated with the flight takes 30 minutes to groom. I.e., there is a 30 minute interval between the arrival and departure of the same plane.
- Once a plane is groomed, the flight to use the plane may depart from the gate immediately. If no flight needs to use the plane immediately, the plane remains at the airport until it is used.
- Reminder: The simulation completes once all the flights have completed or the day has ended.

Expected Output

The output is the same as Assignment 1. Please see examples as shown in the provided test suite (`tests_2.zip`).

Nonfunctional Requirements

Same nonfunctional requirements as Assignment 1. Note: the starting code found in the zip file `assn2.zip`, on Brightspace, is a solution to Assignment 1 and a possible starting point for Assignment 2.

Test Your Code!

To test your code, a test suite (`tests_2.zip` is provided. Same test procedures as those in Assignment 1 are recommended.

Hints and Suggestions

1. Each airport will need a list to keep track of all blocked flights that are waiting for their plane.
2. Either a global or per-airport table of planes and their status will be useful.
3. The sample solution to Assignment 2 only required modifying the `airport.c` file and required 55 lines of additional code.

The Java Option

Same conditions apply as in Assignment 1. A penalty of 10% will be applied if the assignment is done in Java instead of C. The amount of code needed for the Java version is under 50 lines of code because the JDK already has a List implementation, ready to go.

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 (Simulator)	/20
Functionality (automated tests)	/20
Code Clarity	/10
Total	**/100**