CSE 4034 – ADVANCED UNIX PROGRAMMING
Project # 2
(Due: 08.01.2023 23:59)

A car is manufactured at each stop on a conveyor belt in a car factory. A car is constructed from the following 6 parts: i) chassis, ii) tires, iii) seats, iv) engine (assume this includes everything under the hood and the steering wheel), v) top cover, and vi) painting. Thus, there are 6 tasks in manufacturing a car. However, *tires, seats* or *the engine* cannot be added until the *chassis* is placed on the belt. The *top cover* cannot be added until *tires, seats* and *the engine* are put in. Finally, the car cannot be *painted* until the *top cover* is put on. There are four types of technicians assigned to the above tasks in the car factory: **Type A** technician is skilled at adding *tires and painting*, **Type B** can only put the *chassis* on the belt, **Type C** only knows how to attach the *seats*, and **Type D** knows how to add the *engine* as well as how to add the *top cover*.

Your task is to simulate car factory by considering all necessary simulations for constructing *as much car as possible*. Write a Pthread code for types A, B, C and D technicians to be able to work on the car, without violating the task order outlined above. You can use semaphores and mutexes in your solution. Please not that *there can be multiples of each type of technicians!*

DETAILS

- You should implement each technician as a separate thread. The main thread is responsible for creating technician threads, managing them, and displaying the final results as soon as the simulation completes.
- Note that, shared resources among multiple threads are the cars. Additionally, *preserving consistency* and *preventing deadlocks* are major issues to be considered.
- Each operation is performed for a limited amount. For example, there can be at most 20 paintings, 11 top covers etc. The limitations are given in the input file.
- Multiple simultaneous operations on different cars should be allowed in your synchronization solution. However, operations in the same car, should be performed in a *mutually exclusive* way.

- You have to use synchronization primitives to follow operation order in a car (whether the tires, seats, and the engine are added etc.). In general, you have to use synchronization primitives (such as *mutexes*, *semaphores*, *condition variables*, and *barriers*) to synchronize your threads. Please do not synchronize them using global variables or flags!
- There will be a total of simulation days, D, where D is taken from the input file. All the transactions within that period will be logged in an output file.
- Assume that 3 seconds correspond to one simulation day.

Input File

1. The first line of the input file will have five integer values (in the given order):

   < x y z t D >

   a. There are **x** Type A technicians, **y** Type B technicians, **z** Type C technicians and **t** Type D technicians.

   b. The total simulation time is D days. All the transactions within that period will be logged in a file.

2. The second line gives the limitations of operations in the order of chassis, painting, tires, seats, engines and car tops. Consider the following file:

   120 180 130 115 160 220

   In this example, based on this line, it will support 120 chassis, 180 paintings, 130 tires, 115 seats, 160 engines and 220 car tops operations *per day.*

   An example input file is given below;

   2 3 4 2 10

   120 180 130 115 160 220

Output File

1. As the first part of the log file , there should be a line of information in the log file in the following format for each transaction:

   <Technician_ID Car_ID Operation Simulation_Day>

Type A - 1	1	tires	1
Type A - 2	1	painting	1
Type B - 1	1	chassis	1
.....	...	.....	....

2. The second part of the log file gives the total number of operations for each technician for the given simulation period (as total of T days).

3. The last part of the log file gives the total number of cars manufactured at each simulation day.

Important Notes:

- You can work in groups of 2 people.
- Please write a minimum 2-pages long project report that describes how you implemented the project and add several screenshots that show sample runs.
- We will use tools that automatically detect plagiarism among the submissions!
- In case of any form of copying and cheating on solutions, you will get **FF** grade from the course! You should submit your own work. In case of any forms of cheating or copying, both giver and receiver are equally culpable and suffer equal penalties.
- Please zip and submit your files using filename Student1Number_Student2Number_Project1.zip (ex: 150713852_150713098_Project1.zip) to Canvas system (under Assignments tab).
- No late submission will be accepted.