COSC 3360 – 6310 – Fundamentals of Operating System
Assignment #3 for Spring 2019
The multithreaded Database Management System.
Due on Monday, April 29 at 11:59:59 PM

OBJECTIVE

This project will familiarize you with the use of pthreads, pthread mutex semaphores, and pthread condition variables.

THE PROBLEM

A multithreaded database management system receives requests from multiple users to access a database with ten data records. The DBMS must use concurrency mechanisms to guarantee consistency when multiples requests try to access the same data record at the same time. One of the features of the DBMS is an accounting module that prints all the information regarding how the requests from the clients are being processed by the DBMS. The goal of the assignment is to simulate the operation of the DBMS using POSIX threads, POSIX semaphores, and condition variables.

Your program should consist of:

1. The *main thread:* it creates the database and the request threads according to the input specifications.
2. One *request thread* per line in the input file (without including the first line): it simulates the request from a user to access a particular position of the database.

Your program must read the input using I/O redirection. The input file format is shown below:

```
1       // Starting group (1 or 2)
1 3 0 5 // User 1 from Group 1 requesting position 3 at time 0 for 5 sec.
2 3 2 5 // User 2 from Group 2 requesting position 3 at time 2 for 5 sec.
1 3 1 5 // User 3 from Group 1 requesting position 3 at time 3 for 5 sec.
2 1 3 1 // User 4 from Group 2 requesting position 1 at time 6 for 1 sec.
```

The value in the first line of the input file represents the group that will have access to the database first. Each user will belong to group 1 or group 2. The following lines represent the request from the users to access the database. Each request has the following format:

User group: 1 or 2
Database position: from 1 to 10
Request arrival time: it represents the elapsed amount of time (seconds) since the arrival of the previous request (use the sleep function to simulate this time).
Request time: it is the request's duration in seconds (use the sleep function to simulate this time).

Your program will terminate when all requests have been processed by the multithreaded DBMS.

YOUR OUTPUT

Your program should keep track of all the requests generated using the information from the input file. Each user request (handled by a thread) should print a message each time its status changes: (a) arrives at the DBMS; (b) uses the requested database position; and (c) finish its execution. It must also print a message when the request needs to wait due to: (a) the user does not belong to the initial group; (b) the requested position is being used by another user. Based on the previous input, we have the following output:

```
User 1 from Group 1 arrives to the DBMS
User 1 is accessing the position 3 of the database for 5 second(s)
User 2 from Group 2 arrives to the DBMS
User 2 is waiting due to its group
User 3 from Group 1 arrives to the DBMS
User 3 is waiting: position 3 of the database is being used by user 1
User 1 finished its execution
User 3 is accessing the position 3 of the database for 5 second(s)
User 4 from Group 2 arrives to the DBMS
User 4 is waiting due to its group
User 3 finished its execution

All users from Group 1 finished their execution
The users from Group 2 start their execution

User 2 is accessing the position 3 of the database for 5 second(s)
User 4 is accessing the position 1 of the database for 1 second(s)
User 4 finished its execution
User 2 finished its execution
```

At the end of the simulation, your main thread should print a summary with:
1. The total number of requests (classified by group);
2. The total number of requests that waited due to their group.
3. The total number of requests that waited because the position was being used by another user.

This summary could look like:

```
Total Requests:
        Group 1: 2
        Group 2: 2

Requests that waited:
        Due to its group: 2
        Due to a locked position: 1
```

Note: Your program must use I/O redirection. You can safely assume that the input files will always be in the proper format.

These specifications were written on **Friday, April 5, 2019.** Please refer to the course web site for corrections and updates.