Laboratory Assignment 5: Synchronization between threads of a single process

Table of contents

1 Objectives

This practice aims to reinforce our knowledge of the POSIX system's thread and process synchronization mechanisms and usage schemes. In practice, we will make use of: mutex, condition variables, semaphores, and POSIX shared memory objects.

The archive p5files.tar.gz contains several files that can be used as a starting point for developing this practice, as well as some makefiles for the compilation of the different projects.

2 Exercise

We want to design the capacity control system of a discotheque in which the maximum number of customers that can be inside at a given time (capacity) is N. In addition, there are two types of customers: VIP and regular. The rules that the system must comply with are the following:

- If the club is full, new customers will have to wait for a customer to leave the club before they can enter.
- If there are VIP and regular customers waiting, priority will be given to VIP customers. The regular customers will have to wait for the VIP customers to enter first.
- Customers will enter one at a time in strict order of arrival according to their group (regular or VIP) as long as the number of occupants of the discotheque is less than the capacity (N).

The system must be simulated with a program that creates M threads representing the customers of the discotheque. These threads should execute an input function called **client** with the following structure:

```
void *client(void *arg)
{
    ...

    if ( isvip )
        enter_vip_client( ... );
    else
        enter_normal_client( ... );
    dance( ... );
    exit_client( ... );

    ...
}
```

In its creation, each thread will be given two integer arguments:

- id: an identifier corresponding to the thread creation order.
- isvip: a value indicating whether the client is vip or not.

Implement the **enter_vip_client**, **enter_normal_client**, and **exit_client** functions in such a way as to guarantee the system operating conditions described above. Add in these functions console messages with *printf* to track the program, indicating the client's id and what it is doing in each message.

The main program will receive from the command line the name of a file containing the number of clients to be created (M) and each client's VIP or regular character. The expected format of this file is as follows:

An ASCII file organized by lines - The first line indicates the number of customers to create (M). - The following M lines (one per customer) will take the value 1 or the value 0 to indicate the VIP character of that customer (note that 1 and 0 are ASCII characters).

For example, for 5 clients, 3 non-VIP and 2 VIP, the content of the file could be:

```
5
0
0
1
0
1
```

The file can be easily parsed using the standard C library function *fscanf*. See its manual page.