Universidade de Aveiro

Departamento de Electrónica, Telecomunicações e Informática

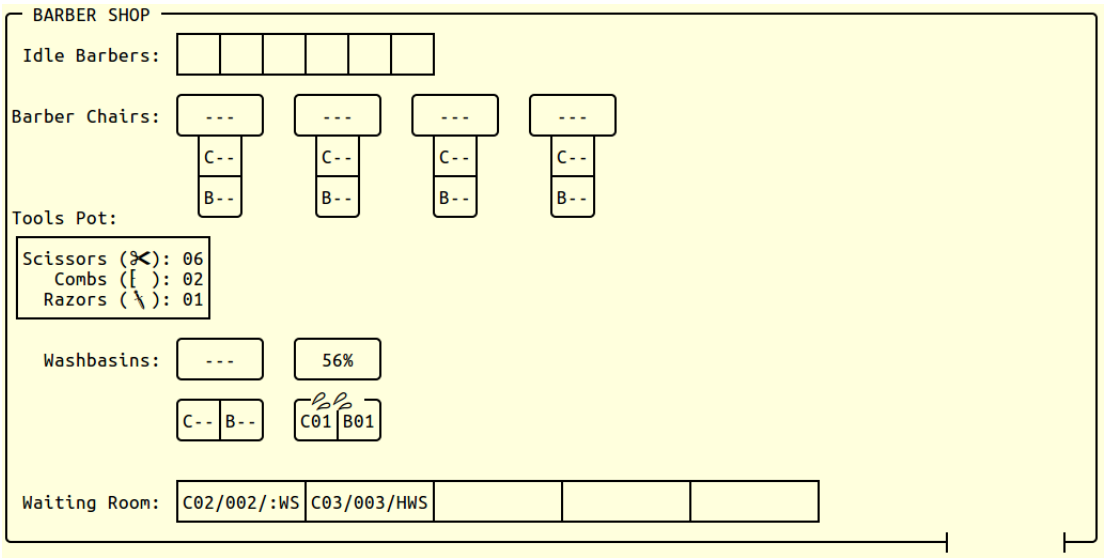
Sistemas de Operação

(Academic year of 2018/19)

assignment #2 December, 2018

Barber shop simulation – DRAFT!

In this paper we intend to simulate a barbershop. The barbershop belongs to a non-empty group of barbers, provides three possible services – haircut, shave, and hair wash – eventually applicable to the some client, and contains five different areas: A bench for free barbers, a group of barber chairs (for haircut and shaving), a tools pot (containing a certain amount of scissors, combs, and razors), a group of washbasins, and one or more benches for waiting clients.



Barbers and clients are active entities (thread or process based), and their activities are a random amount of time units (defined by the simulation parameters). Time units are converted to absolute time by a provided timer module.

The barbershop is a shared module between barbers and clients, and was build by composing a group of simpler modules (`barber-bench`, `barber-chair`, `tools-pot`, `washbasin`, and `client-benches`). The data structures of all these modules are mapped inside a contiguous structure within barbershop (named `BarberShop`).

Barbers and clients active entities are delayed a random amount of time units in their activities (bounded by the simulation parameters).

Source code is provided that implements many (sequential) aspects of the simulation (logger module, random functions, global parameters, program argument handling, barber and client life-cycle, etc.), and should be carefully analyzed and understood.

1 A barber life

The barber's life consists on working which the barbershop is opened. Basically, it sits on a bench, waiting for clients (or a closed shop). When a client arises, is performs the requested

services, one by one, and then it releases the client (and sits on the bench again). A client may request any combination (except the empty one) of the three services provided by the barbershop (haircut, shave, hair wash).

Each service type requires different resources:

- **Haircut**: a scissor and a comb are required, and the service is done in a barber chair.
- **Shave**: a razor is required, and the service is also done in a barber chair.
- **Hair wash**: no tool is required, only a washbasin.

Please note that any one of these services require proper coordination (synchronization) between the barber and its client. For instance, both are required to access the barber chair (or washbasin) before the service is performed.

The following code shows the life of a barber:

```
static void life(Barber* barber)
{
    assert (barber != NULL);

    sit_in_barber_bench(barber);
    wait_for_client(barber);
    while(work_available(barber))
    {
        rise_from_barber_bench(barber);
        process_resquests_from_client(barber);
        release_client(barber);
        sit_in_barber_bench(barber);
        wait_for_client(barber);
    }
    done(barber);
}

...
```

2 A client life

If you consider barber's life boring, the good news is that we have not yet presented the client's life. A client lives only for a given number (randomly defined according to the simulation parameters) of barbershop services. It basically, wanders a random amount of time (limited by the simulation parameters) outside the shop, and then if (randomly) selects a set of services it desires (corresponding to values in the interval $[1; 7]$) and then, tries to enter the barbershop (if and only if there is a random empty sit in the client's benches). If so, it will be sited in the client's benches, waiting (in queue) for its turn.

A barber will then interact (indirectly through the `BarberShop` shared structure) with him to perform the requested services (one by one).

3 The barbershop

The module barbershop contains (in a contiguous data structure) all the shared information regarding the five areas of the shop.

4 Other modules

Log entity

A fully working[1] concurrent logger module is given to observe the dynamics of the simulation in a clear and precise way. This module has two modes of operation – line and window – that allows alternative observing of the simulation (line mode, generates the classical list of lines describing the state of the simulation parts; and window mode provides an innovative alternative for a more interesting visualization – this last mode is exemplified in the presented figure).

Support code

To ease the development of this assignment, a fair amount of working source code is provided (look into it carefully). In particular, wrapping modules for the usage of all concurrent libraries (POSIX threads, System V IPC, and POSIX semaphores/shared-memory) are provided. These modules have functions with similar prototypes than those in the library, but with automatic error handling either by throwing exceptions (with `errno` value), or aborting the process with proper error messages. Hence, there will be no need for error checking when using these modules.

Also there are some utility modules for memory allocation, queue's, etc..

Assignment

Devise a possible solution and write two simulations for this problem, one using POSIX threads, mutexes and condition variables, and the other using UNIX processes, shared memory and semaphores (System V or POSIX).

The solutions should be implemented in C++ and be run in Linux. It is highly advised to use a utf8 friendly terminal such as `gnome-terminal` or `terminator`.

[1] Not counting possible bugs, of course.