C++ Project

Plazza

Dan Baudry baudry_d@epitech.eu
Maxime Montinet zaz@epitech.net
David Giron thor@epitech.net

Abstract: The purpose of this project (on top of a tribute to our mess at the corner of the street) is to make you handle multi-process/multi-thread, through the simulation of a pizzeria with its kitchens. You will learn to manage various problems: load balancing, processes and threads synchronization, cook management etc.

Contents

Chapter I

Information

First of all, some reading to awake your appetite:

- Pizzeria : http://fr.wikipedia.org/wiki/Pizzeria
- Named pipe : http://en.wikipedia.org/wiki/Named_pipe
- Processes : `man fork, man exit, man wait, man ...`
- POSIX threads : `man pthread_*`

Chapter II

Generalities

The purpose of this project is to make you realize a simulation of a pizzeria, which is composed of the reception that accepts new commands, of several kitchens, themselves with several cooks, themselves cooking several pizzas.

This is an overview of the expected architecture:

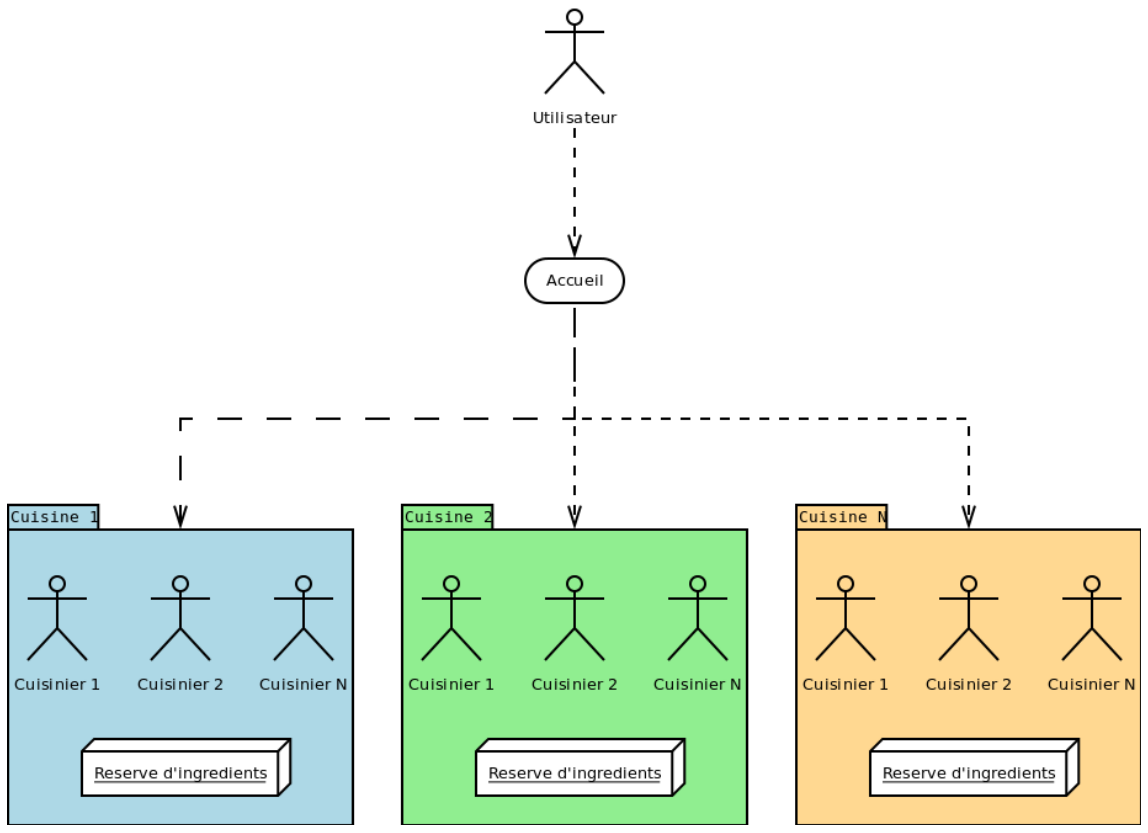

Figure II.1: `Architecture`

Chapter III

Mandatory section

III.1 The reception

The reception must be started using the command line the following way:

```
>./plazza 2 5 2000
```

- The first parameter is a multiplier for the cooking time of the pizzas. It is used to examine your program more easily, so it must **INEVITABLY** be implemented. Otherwise it will not be possible to grade you. Moreover this parameter MUST be able to accept numbers with value in between 0 to 1 to obtain a divisor of the pizzas cooking time... Cooking time is detailed later.
- The second parameter is the number of cooks per kitchen. Cook definition is detailed later.
- The third parameter is the time in milliseconds, used by the kitchen stock to replace ingredients. Ingredient definition is detailed later.

The reception **MUST** be interactive. It must present a UI allowing **at least** the following actions:

- Commands of a pizza by the user though command line, for example "regina XXL x7". This will be detailed later.
- Displays the kitchen status, including the current occupancy of the cooks, as well as theirs stocks of ingredients.

The library used to display the ordering is not important, There are only few points for this part.

`The graphic part IS NOT IMPORTANT. We strongly suggest to use a basic tool such as NCurses, NDK++, Qt, ...so that you are not wasting your time.`

Pizza ordering MUST respect the following grammar:

```
S := TYPE SIZE NUMBER [;TYPE SIZE NUMBER]*

TYPE := [a..zA..Z]+

SIZE := S|M|L|XL|XXL

NUMBER := x[1..9][0..9]*
```

Ordering example which is grammatically valid:

```
regina XXL x2; fantasia M x3; margarita S x1
```

`It is not because the grammar is very simple that your parser must be too basics! splits as well as others hacks must definitively be avoided ...`

- The user MUST be able to place more orders when the program is running.The program MUST be able to adapt.
- The reception MUST allocate pizza by pizza to kitchens when receiving an order. When all the kitchens are saturated, it MUST create a new one (do a fork as explained later.)
- The reception MUST always allocate pizza to kitchens so that the occupancy is as balanced as possible. You must not have one kitchen with all the pizzas and the others not doing anything!
- When an order is ready, the reception must display the information to the user and keep a record. (A log file on top of other displays should be a good idea ...)

III.2 Kitchens

Kitchen are a child process of the reception. Kitchens are created progressively, when needed. Kitchens possesses a predetermine number of cooks that is defined when the program is started.

Cooks MUST be represented by threads. When a cook does not have a task, he must yield. Cooks start to work one after the other, when order arrives.
These threads MUST be scheduled by a Thread Pool local to each kitchen.

You must propose an object encapsulation for the following notions:

- Processes
- Threads
- Mutex
- Conditional variables

⚠ `These 4 abstractions represents a very important part of the points available in the scale. You should execute this encapsulation intelligently...`

Moreover :

- Each kitchen CAN NOT accept more than 2 X N pizza, (meaning pizza to cook, or pizza waiting to be cooked) with N being the number of Cooks A kitchen must refuse any command of pizza over this number.
- The reception MUST open a new kitchen of the existing kitchen can't accept anymore order.
- Cook love their work and are accountable for it. A cook WILL NOT prepare more than one pizza at a time!
- Kitchens communicate with the reception thanks to `named pipes`. There are not other authorized means of communication. You MUST use `named pipes` for this purpose.
- You must propose an object encapsulation for the `named pipes`. This encapsulation CAN offer overload for the operators "«" and "»".

- If a kitchen doesn't work for more than 5 seconds, this kitchen MUST close.
- A kitchen possess a stock of ingredients that contains, when created, 5 units of each ingredient. The stock regenerate 1 units of each ingredients every N seconds. N being the number passed in the command line at the start of the program.
- Pipes MUST be unidirectional. This means one pipe per direction, against one pipe half duplex. Then there are two types of pipes: pipes "in" and pipes "out".

`Creation and destruction of kitchen means that there are communication problems that need to be sorted out and watched over very closely...`

III.3 Pizzas

As explained earlier, the reception must allocate order between kitchens, pizza by pizza. For example if one command is about 7 margaritas, these margaritas will be dispatched between 7 different kitchens (if there are 7 kitchen running at this point in time.)

When the information is flowing through the named pipes, information about the command and pizzas return MUST be serialized. You MUST use the following definition of value:

```
enum TypePizza
{
    Regina = 1,
    Margarita = 2,
    Americaine = 4,
    Fantasia = 8
};

enum TaillePizza
{
    S = 1,
    M = 2,
    L = 4,
    XL = 8,
    XXL = 16
};
```

Within communication, pizza are passing through, using the form of a type object opaque of your choice. It MUST be possible to use operators `pack` and `unpack` on this type to serialize or to unserialize data.

You MUST manage the following pizzas:

- `Margarita` : Contains doe, tomato and Gruyere. Baked in 1 sec * multiplier.
- `Regina` : Contains doe, tomato, gruyere, ham, mushrooms. Baked in 2 secs * multiplier.
- `Americana` : Contains Doe, tomato, gruyere, steak. Baked in 2 secs * multiplier.
- `Fantasia` : Contains Doe, tomato, eggplant, goat cheese, and chief love (and not goat love and chief cheese.) Baked in 4 secs * multiplier.

`You must ask yourself as early as possible how to represent time. This can save you lots of time...`

`Being able to add new pizzas very simply (abstraction?) is a very easy bonus to get.`

Chapter IV

Instructions

You are more or less free to implement your program how you want it. However there are certain rules:

- The only functions of the `libc` that are authorized are those one that encapsulate system calls, and that don't have a C++ equivalent.
- Any solution to a problem MUST be an object approach
- Each value passed by copy instead of reference or by pointer must be justified. Otherwise, you'll loose points.
- Each value non `const` passed as parameter must be justified. Otherwise, you'll loose points.
- Each member function or method that does not modify the current instance and which is not `const` must be justified. Otherwise, you'll loose points.
- There are no `C++` norm. However if a code is reckoned to be unreadable or dirty, this code will be penalized. Be serious please!
- It is FORBIDDEN to have connections superiors to `(if ... else if ... else ...)`. You must factorize!
- Keep an eye on this project instructions. It can change with time!
- We are very concern about the quality of the materials. Please, if you find out any spelling mistake, grammatical errors etc. please contact us at `koala@epitech.eu` so that we can do a proper correction within the day.
- You can contact authors thanks to the emails indicated in the header.
- The intranet forum holds information and answers to your questions. Make sure that answers to your question are not already there before contacting the authors. of this document.

Chapter V

Turn in instructions

You must turn in your project in the repository `SVN` provided by the `Koalab`. Your repository will be open a maximum of 48h after the date of the end of registration, `Epitech` intranet being the reference. It will not be possible to register to the project and to pass the oral defense once this date is expired.

Repository will be closed at the exact hour of the end of the project, `Epitech` intranet being the reference.It is useless to complain because at your own watch you were still on time. The only relevant time is the one from Epitech which has an automatic system.

Corollary to the Murphy's law: "If you turn in your work within the last hour, something will inevitably go wrong."
Only the code from your repository will be graded during the oral defense. The documentation about repository and provided by the `Koalab` comes with this project.

Good luck!