B4 - Concurrent Programming

B-CCP-400

Plazza

Who said anything about pizzas?

2.0

{EPITECH.}

Plazza

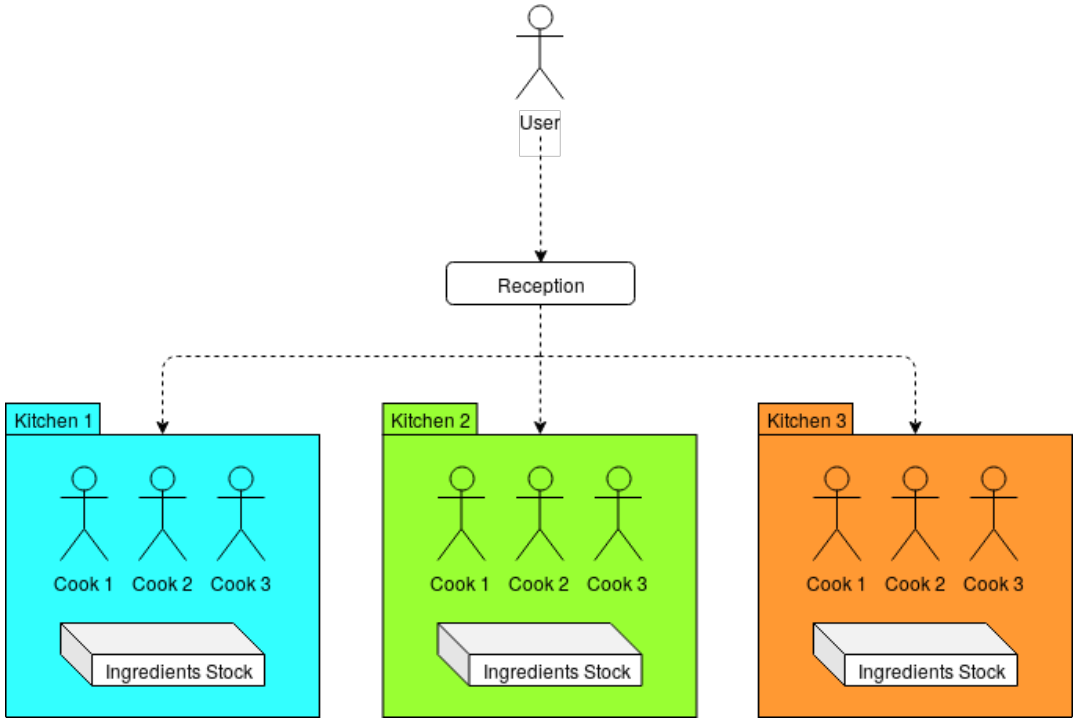
binary name:	plazza
repository name:	CCP_plazza_*$ACADEMICYEAR*
repository rights:	ramassage-tek
language:	C++

- Your repository must contain the totality of your source files, but no useless files (binary, temp files, obj files,...).
- All the bonus files (including a potential specific Makefile) should be in a directory named *bonus*.
- Error messages have to be written on the error output, and the program should then exit with the 84 error code (0 if there is no error).

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.
You will learn to deal with various problems, including load balancing, process and thread synchronization and communication.
Before you get started, should take some time to read up on the following tools you'll need to use:

- Processes (`man fork`, `man exit`, `man wait`, `man ...`)
- Inter-process communication (IPC)
- STL threads
- POSIX threads (`man pthread_*`)

Here is an overview of the expected architecture:



THE RECEPTION

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

```
Terminal
~/B-CCP-400> ./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 an interactive shell with 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 kitchens status, including the current occupancy of the cooks, as well as theirs stocks of ingredients. using the `status` command.

In the `bonus` directory you may add graphical version of the reception, that'll be a great bonus to make!

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 may be too basic! spits, as wall as other hacks, must definitively be avoided…

The reception **MUST**:

- be able to place more orders when the program is running. The program MUST be able to adapt.
- 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.)
- 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 dislpays should be a good idea…)

Kitchens

Kitchens are child processes of the reception. Kitchens are created progressively, when needed. Kitchens possesses a predetermined 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 each of 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 if 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 an IPC (choose the one to use wisely).
- You must propose an object encapsulation for the choosen IPC you're using. 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.

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

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 choosen IPC, information about the command and pizzas return MUST be serialized. You MUST use the following definition of value:

```
enum PizzaType
{
    Regina = 1,
    Margarita = 2,
    Americana = 4,
    Fantasia = 8
};

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

Within communication, pizza are passing through, using the form of an opaque object type 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 follwing 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. 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.