C++ Project

Plazza

Koalab koala@epitech.eu

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 a giant information collect made for an aggressive marketing campaign against the Fratello. You will learn to manage various problems: load balancing, processes and threads synchronization, communication 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
- UNIX sockets : https://en.wikipedia.org/wiki/Unix_domain_socket
- Internet sockets : https://en.wikipedia.org/wiki/Network_socket
- Processes : `man fork, man exit, man wait, man ...`
- POSIX threads : `man pthread_*`
- STL threads : http://www.cplusplus.com/reference/thread/thread/

Chapter II

Generalities

The purpose of this project is to make you realize a scrapper, which is composed of a main program, with an user interface, that accepts new commands, of several process, themselves with several threads, themselves looking for mails inside several files.

This is an overview of the expected architecture:

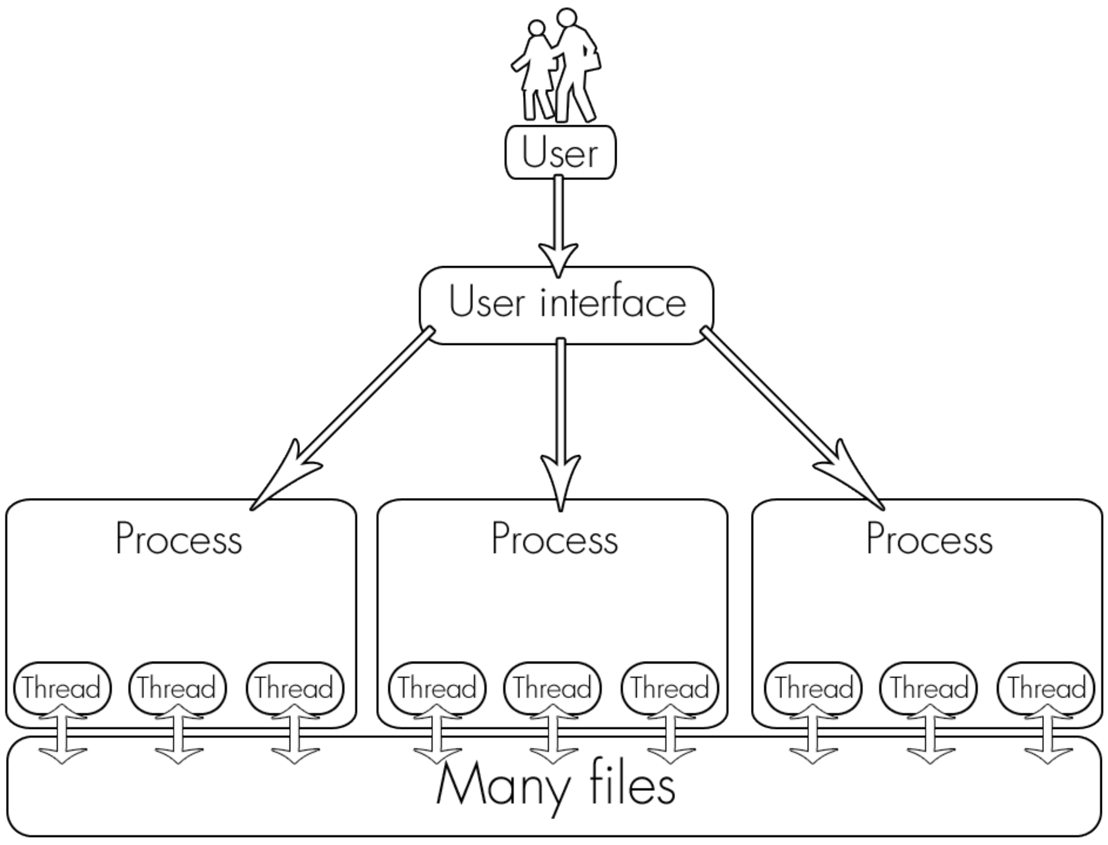

Figure II.1: `Architecture`

Chapter III

Mandatory section

III.1 The reception

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

```
    >./plazza 5
```

- The parameter is the number of threads per process. What a thread must do is detailed later.

Two versions of your program must be done. The first one is the default one: a pure text mode program that read commands on STDIN.

This version **MUST** fully compile even if there is no graphic, gui or curses library installed on the system. It **MUST** write every extracted data on STDOUT, with a new line between every data, so the following command must work well:

cat commands | ./plazza 5 > datas

The second one is build by specifying "ui" to the Makefile. It builds your project with a user interface. Your UI must propose the following functionnalities:

- Ask to browse file by the user through command line.
- Displays the scrapper status, including the current occupancy of the threads, as well as their unachieved work..

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 so that you are not wasting your time. You have to install your library by yourself if it is not already installed on your computer.`

Orders sending MUST respect the following format: FILE+ INFORMATION_TO_GET
Several orders can be chained thanks to a “;”.
Ordering example which is grammatically valid:

```
    index.htm EMAIL_ADDRESS; company.csv memo.txt PHONE_NUMBER
```

`It is not because it 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 main program will schedule task one by one and dispatch them equally between processes. When all the process are saturated, it MUST create a new one.
- The main program MUST always allocate orders to process so that the occupancy is as balanced as possible. You must not have one process with all the files and the others not doing anything!
- When an order is ready, the main program 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 Process

Process are a child of the main program. Process are created progressively, when needed. Process possesses a predetermined number of threads that is defined when the program is started.

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

You must propose an object encapsulation for each of the following notions:

- Processes
- Threads
- Mutex
- Conditional variables

We also recommand you to offer an object oncapsulation for:

- Named Pipe
- Unix Socket
- Internet Socket

depending the communication method you use.

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

Moreover :

- Each process CAN NOT accept more than 2 X N orders, (meaning orders, or awaiting orders) with N being the number of threads. A process must refuse any orders over this number.
- The main program MUST open a new process if the existing process can't accept anymore order.

- Threads love their work and are accountable for it. A thread WILL NOT respond to more than one order at a time!
- Process communicate with the reception thanks to `named pipes`. You can also use `UNIX sockets` and `Internet sockets`.
- You must propose an object encapsulation for the `named pipes` if you use them. This encapsulation CAN offer overload for the operators "«" and "»".
- You must propose an object encapsulation for BOTH of the `UNIX sockets` and `Internet sockets`, allowing another programmer to switch from one to the other, if you chose to use sockets instead of pipes.
- If a process doesn't work for more than 5 seconds, this process MUST exit.
- Pipes MUST be unidirectional. This means one communication per direction, against one communication half duplex. Then there are two types of pipes: pipes "in" and pipes "out".

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

III.3 Orders

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

When the information is flowing through the communication channel, it MUST be serialized. You MUST use the following definition of value:

```

enum Information
  {
    PHONE_NUMBER,
    EMAIL_ADDRESS,
    IP_ADDRESS
  };
```

Within communication, orders are passing through, using the form of an opaque type object of your choice. It MUST be possible to use operators « and » on this type to serialize or to unserialize data.

You MUST manage the following :

- `PHONE_NUMBER` : May be a french number with 10 numbers; separated or not by one space.
- `EMAIL_ADDRESS` : [a-zA-Z0-9_.-]+ '@' [a-zA-Z0-9_.-]+
- `IP_ADDRESS` : [0-255].[0-255].[0-255].[0-255]

File may be ciphered with a XOR or a binary CAESAR ciphering. A XOR key will always be between 1 and 2 bytes long and a CAESAR key 1 byte long. You must of course uncipher the file to browse and get informations from it.

`Being able to add new orders 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 concerned 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 school social network 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 provided by `Epitech`. The repository name is `cpp_plazza`

Repository will be cloned 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.

Good luck!