B4 - C++ Programming

B-CPP-401

Plazza

Who said anything about pizzas?

Plazza

binary name:	plazza
group size:	2-3
repository name:	cpp_plazza
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).

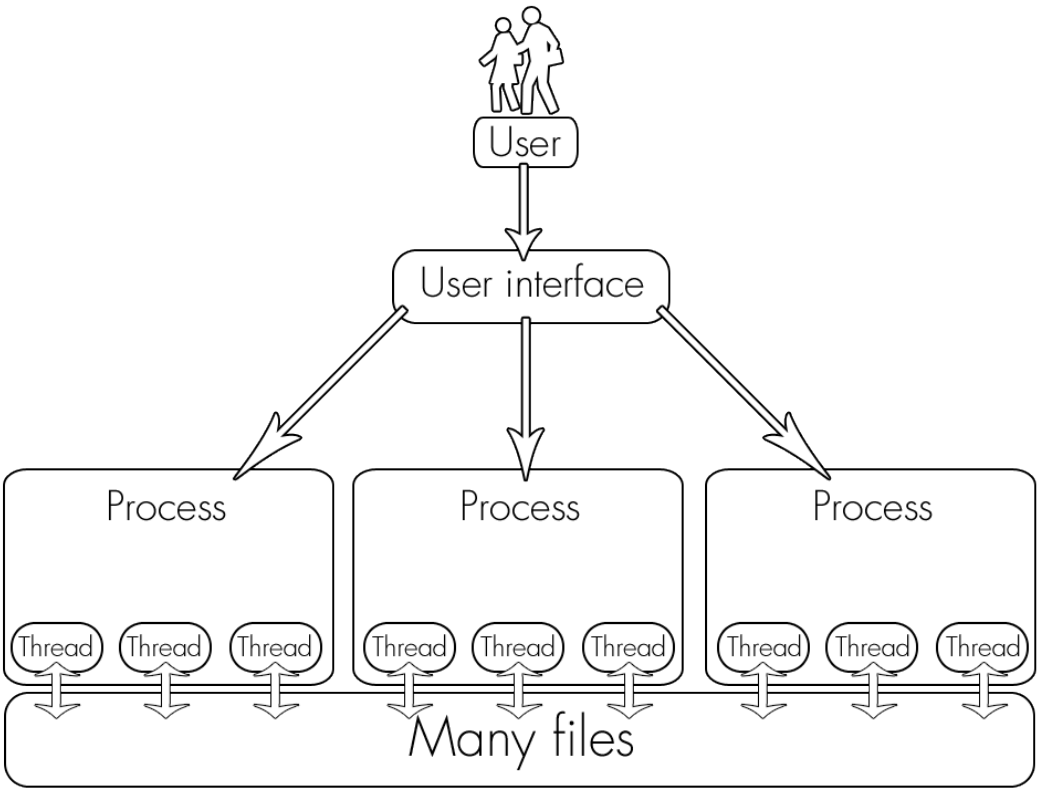
In this multi-process and multi-threaded program, you will deal with large amounts of information in the context of an aggressive marketing campaign against the Fratello (the other restaurant around the corner). You will learn to deal with various problems, including load balancing, process and thread synchronization and communication.

Before you get started, should take some time to read up on the following tools you'll need to use:

- Named pipes
- UNIX sockets
- Internet sockets
- Processes (`man fork`, `man exit`, `man wait`, `man ...`)
- STL threads
- POSIX threads (`man pthread_*`)

The purpose of this project is to implement a scrapper, composed of a master process with a user interface which receives commands and of several slave processes.
Each of these processes must be running several threads.
Each of these threads will be searching several files for specific information.

Here is an overview of the expected architecture:



THE MASTER PROCESS

The master process must be started using the command line like so:

```
Terminal
~/B-CPP-401> ./plazza 5
```

The argument is used to specify how many threads can be started by each slave process.
A thread's behavior will be detailed later.

You must implement two versions of your program:

- A CLI version that reads commands from the standard input
- A GUI version which should only be built by specifying the `ui` parameter to `make`

+ CLI VERSION

The CLI version of your program **MUST** compile even if there is no graphical, GUI or curses library installed on the system.
It **MUST** output all the data it extracts to the standard output, separated by newlines.

It must be possible to run your program like so:

```
Terminal
~/B-CPP-401> cat commands | ./plazza 5 > data
```

Commands will be formatted the following way:

```
FILE+ INFORMATION_TO_GET
```

Several commands may be given at once, separated by a *";"* character. E.g.:

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

+ GUI VERSION

The GUI version of your program must:

- let the user select a file to scrap,
- display the status of the scrcapper, including how busy each thread is and the amount of information left to process.

The graphics library you use is not important.

The graphics part is not the most important.
We strongly suggest you use simple graphics and do not waste your time.

+ BOTH VERSIONS

- It must be possible for the user to issue new commands whilst the program is running. The program MUST adapt.
- The master process must schedule tasks one by one and dispatch them equally among the slave processes.
  When all slaves are saturated, a new one MUST be created.
- The master process MUST ALWAYS dispatch orders to processes so that the load is as balanced as possible.
  It must not be possible for a single slave to be handling all the files while the othersr aren't doing anything.
- When information has been found, the main program must display it to the user and keep a record (a log file could be a good idea)

The slaves

Slave processes are children of the master process.
They must be created progressively, when needed.
Processes have a predertemined number of threads, which is provided as a startup parameter to the program.

When a thread has no tasks remaining, it must yield.
Threads start working one after the other, when commands are received.
These threads MUST be scheduled by a **Thread Pool**.

You must provide an object-oriented encapsulation for each of the following items you use:

- processes,
- named Pipes,
- Unix Sockets,
- Internet Sockets.

These abstractions are VERY important.
You should take some time to design them well.

Here are some requirements regarding the slaves:

- each slave CANNOT accept more than 2xN commands (be they executing or pending), N being the number of threads,
- the master MUST start a new slave if the existing slaves can't accept any more commands,
- slaves love their work and are accountable for it. A slave WILL NOT respond to more than one command at a time,
- slaves communicate with the master through **named pipes**, **UNIX sockets** and/or **Internet sockets**. You are free to use any number of these,
- if a slave is idle for over 5 seconds, it MUST exit,
- pipes MUST be unidirectional. This implies there are "in" pipes and "out" pipes.

Creating and destroying processes means that there may be communication problems to deal with…

Commands

As explained above, the master must dispatch commands among the slaves.
For instance, if a command requires 7 files to be processed, these files must be dispatched between 7 different slaves (if there are 7 slaves running at the time).

When the master and the slaves communicated, the information they send and receive MUST be **serialized**.
You MUST use the following definition:

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

During communication, commands shall transit under the form of an opaque type object of your design.
It MUST be possible to use the << and >> operators to serialize and unserialize data.

Your program MUST deal with the following types of information:

- `PHONE_NUMBER`
  May be a french number with 10 numbers, which may or may not be separated by a single space.
- `EMAIL_ADDRESS`
  [a-zA-Z0-9_.-]+ '**@**' [a-zA-Z0-9_.-]+
- `IP_ADDRESS`
  [0-255].[0-255].[0-255].[0-255]

Making it possible to add new types of information easily (using an abstraction?) is a very easy bonus to get.

Going further on

As a bonus, make your program able to process information from files ciphered with a XOR or binary CAESAR.

Of course, the program should not know in advance wether or not a file was ciphered.