Process Monitor

REVIEW | CODE REVIEW 24 | HISTORY

src/linux_parser.cpp 14

src/system.cpp 2

```
#include "system.h"

#include <unistd.h>

#include <cstddef>
#include <set>
#include <string>
#include <vector>

#include "linux_parser.h"
#include "process.h"
#include "processor.h"

using std::set;
using std::size_t;
using std::string;
using std::vector;
```

SUGGE STION

You can just use the line using `namespace std;`
It will take care of all the std:: that you use as a prefix of many STLs.

Important

But take care of the following issue addressed here.

https://stackoverflow.com/questions/11271889/global-variable-count-ambiguous

So always take care of this while using

```
using namespace std;
```

```

// Define a constructor for System

System::System() { cpu_ = Processor(); }

// DONE: Return the system's CPU
Processor& System::Cpu() { return cpu_; }

// TODO: Return a container composed of the system's processes
vector<Process>& System::Processes() {
  processes_ = {};

  vector<int> Pids = LinuxParser::Pids();

  for (Process pid : Pids) {
    processes_.emplace_back(Process(pid));
```

AWE SOME

It is good to see that you have been using `emplace_back` instead of `push_back` .it is much more efficient than `push_back` .
push_back constructs a temporary object which then will need to get moved into the vector v whereas emplace_back just forwards the argument and construct it directly in place with no copies or moves needed.
In short, it is a good habit if you are using `emplace_back` instead of `push_back`

Add To Knowledge

You should know that you do not need to call the constructor when inserting an element using `emplace_back` into the vector of a some class if the constructor of the class has an appropriate definition which can be called on that set of argument!

For exmaple the following two set of code will behave the same!

Code 1

```
vector < Process > & System::Processes() {
    const vector < int > & pids = LinuxParser::Pids();
```

```
    for (const int & pid: pids) {
        Process process(pid);
        processes_.emplace_back(process);
    }
    return processes_;
}
```

Code 2

```
vector < Process > & System::Processes() {
    const vector < int > & pids = LinuxParser::Pids();
    for (const int & pid: pids) {
        processes_.emplace_back(pid);
    }
    return processes_;
}
```

This can be explained as when you `emplace_back` the pid by `processes_.emplace_back(pid)`, Constructor of the class Process

```
Process::Process (int pid){
  _pid = pid;
  ...
}
```

gets called and since we already have a constructor with definition which takes a pid and returns the Process Object!

Extended Example

Please run this example on your local terminal and see the output yourself!

```
#include <iostream>

#include <vector>

class A {
    public:
        A(int x_arg): x(x_arg) {
            std::cout << "A (x_arg)\n";
        }
    A() {
        x = 0;
        std::cout << "A ()\n";
    }
    private:
        int x;
};
int main() {
    std::vector < A > a;
    std::cout << "call emplace_back:\n";
    a.emplace_back(0);
}
```

Output

```
call emplace_back:
A (x_arg)
```

I hope I have given you something useful and interesting!

```
34    }
35
36    for (Process process : processes_) {
37      process.CpuUtilization();
38    }
39
40    sort(processes_.begin(), processes_.end(),
41         [](Process& a, Process& b) { return b < a; });
42
43    return processes_;
44  }
45
46  // DONE: Return the system's kernel identifier (string)
47  std::string System::Kernel() { return LinuxParser::Kernel(); }
48
49  // DONE: Return the system's memory utilization
50  float System::MemoryUtilization() { return LinuxParser::MemoryUtilization(); }
51
52  // Done: Return the operating system name
53  std::string System::OperatingSystem() { return LinuxParser::OperatingSystem(); }
54
55  // DONE: Return the number of processes actively running on the system
56  int System::RunningProcesses() { return LinuxParser::RunningProcesses(); }
57
58  // DONE: Return the total number of processes on the system
59  int System::TotalProcesses() { return LinuxParser::TotalProcesses(); }
60
61  // DONE: Return the number of seconds since the system started running
62  long int System::UpTime() { return LinuxParser::UpTime(); }
```

▸ **src/processor.cpp** 2

▸ **src/process.cpp** 2

▸ **src/format.cpp** 1

- Makefile 1
- include/ncurses_display.h 1
- include/linux_parser.h 1
- src/ncurses_display.cpp
- src/main.cpp
- README.md
- include/system.h
- include/processor.h
- include/process.h
- include/format.h
- CMakeLists.txt

Learn the best practices for revising and resubmitting your project.

RETURN TO PATH