UDACITY

‹ Return to "C++" in the classroom

Process Monitor

REVIEW

CODE REVIEW 16

HISTORY

▸ src/linux_parser.cpp 9

▸ src/system.cpp 2

▾ src/process.cpp 2

```
#include <unistd.h>
#include <cctype>
#include <sstream>
#include <string>
#include <vector>

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

using std::string;

// Constructor
Process::Process(int id) : id_(id) {}

// Return this process's ID
int Process::Pid() const {
   return id_;
}

// Return this process's CPU utilization
float Process::CpuUtilization() const {
   return LinuxParser::CpuUtilization(id_);
```

```
23 }
24
25 // Return the command that generated this process
26 string Process::Command() {
27    return LinuxParser::Command(id_);
```

SUGGESTION

Here you can make sure that you do not send command more than of 50 characters (or 40 characters - It i
You can append ... in the end if the command exceeds more than 40-50 characters and all.

```
28 }
29
30 // Return this process's memory utilization
31 string Process::Ram() {
32    return LinuxParser::Ram(id_);
33 }
34
35 //  Return the user (name) that generated this process
36 string Process::User() {
37    return LinuxParser::User(id_);
38 }
39
40 // Return the age of this process (in seconds)
41 long int Process::UpTime() {
42    return LinuxParser::UpTime(id_);
43 }
44
45 // Overload the "less than" comparison operator for Process objects
46 bool Process::operator<(Process const& a) const {
47    return a.CpuUtilization() < this->CpuUtilization();
```

SUGGESTION

It is so nice to see that you have overloaded less than a comparison operator for process objects.

Similarly, you can do so for any operator using the correct logic and application

Note : Following Operators cannot be overloaded:

1. `.` (Member Access or Dot operator)
2. `?:` (Ternary or Conditional Operator)
3. `::` (Scope Resolution Operator)
4. `.*` (Pointer-to-member Operator)
5. `sizeof` (Object size Operator)
6. `typeid` (Object type Operator)

   For greater than operator (You must be knowing this also)
   Similarly, you can implement the > operator and then make use of it.
   But before that, you should define this custom operator in `process.h`

```
48 }
49
```

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

RETURN TO PATH