Process Monitor

REVIEW | CODE REVIEW 24 | HISTORY

Requires Changes

1 SPECIFICATION REQUIRES CHANGES

Almost There

You have done a good job on this project! You understand all parts of this project well.You are just one step away from completion of this project.

I have provided some suggestions on the other files which you have not made changes. I hope you will find them useful! Please refer to Code Review section

---

Add To Knowledge

1. Debugging

Hey i am giving you a very cool piece of code which will be very much helpful in debugging.

Let me give you an example:

```
#include <iostream>
#include <string>

#define printVariableNameAndValue(x) cout<<"The name of variable **"<<(#x)<<"** and the value of variable is => "<<
x<<"\n"
// I am talking about this piece of code ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
^^^^^^

using namespace std;


int main(){
    int n = 25;
    string myFirstStringEver = "Hello World";

    printVariableNameAndValue(n);
    printVariableNameAndValue(myFirstStringEver);

}
```

Output of the function

```
The name of variable **n** and the value of variable is => 25
The name of variable **myFirstStringEver** and the value of variable is => Hello World
```

So what this following piece of code does is that it takes the your variable and then by writing `(#x)` it converts the name of the variable
into a string and then obviously the variable x holds the value of x so the second half of the code `<<x<<` will print the value of x;

I think you should try to run the code once and then you will understand the working of the code.

The line you just use implements the use of preprocessor directives,commonly referred with the name macro.
For more you must visit this link.

2. Commenting

There is a nice article on the pros and cons of commenting, click here
Efficiency comes with practice (writing more and more programs).
There is a whole chapter on comments in Book **Clean Code** by Robert C. Martin.
Click here for the book.
Summary of the chapter:

> In his book Clean Code, advocates using comments for the following purposes:
> 1.Explanation of Intent
> 2.Clarification
> 3.Warnings
> 4.Amplification
> 5.ToDo
> At a minimum, comments should describe what each public member does and how to use it, and explain all parameters and return values with acceptable ranges (e.g. between 1 and 1000) for each.

3. References to a Live Project

Once you are done with this project then you should have a look at the htop package.
I have looked at all the files and found very great details, very minute details.
I was so impressed that I even made changes to my code according to those minute details.
It is good if you want to explore that too but i would recommend if you get stuck with
any terms then google it and even after that you do not get it then you should just leave it
like that because some of the terms might come up from Operating Systems and knowing just a
a little bit (You might understand it the wrong way) about a very important and heavy concept is
not recommended from my side.

4. Awesome Resources on Web

Here are some awesome resources I like to go through as additional material for C++ OOP:

1. You can check out from chapter 8-12 and even later chapters to do some more practice in OOP -- learncpp
2. Basic Concepts of Object Oriented Programming using C++
3. A short and quick but wonderful tutorial on OOP -- Object Oriented Programming
4. 4 Advantages of Object-Oriented Programming
5. Differences between Procedural and Object Oriented Programming
6. Modern C++ Object-Oriented Programming
7. Great series on basic C++ with some important OOP concepts throughout the series -- C++ Beginner Game Programming
8. A good overview of OOP though it's covered in Python you will find a lot of concepts well explained -- Object Oriented Programming
9. A good amount of OOP concepts are used while making this role-playing game. I am posting only the first part of the series but there are 4 parts for this series so do complete them. You will have a fun time doing it -- Code-It-Yourself! Role Playing Game
10. Another good resource on Polymorphism -- Polymorphism
11. A great video on Pointers -- What Are Pointers? C++

Here are a few additional resources you can look at it for more information on Object Oriented Programming:

1. C++ Core Guidelines
2. CppCon 2019 – Back to Basics Object Oriented Programming - This back to basics series is so much awesome. I would recommend you to go through all of the back to basics CppCon
3. SOLID
4. Design Patterns

You may also find the following reading material helpful:

1. Code Complete, Steve McConnell
2. Clean Code, Robert C Martin
3. The Pragmatic Programmer, Andrew Hunt and Dave Thomas.
4. Design Patterns: Elements of Reusable Object Oriented Software, Emich Gamma et all.

5. Some Linux Exclusive Resources

1. Shell Workshop
2. How to View Running processes on Linux! Please find the link
3. Learning Bash

PS - You can always attach a picture of your error (if there is any) in future in a folder and then mention about it in the notes to the reviewer section while submitting.
That way we can help you in a better way. Take care of that in the future. Wish for a happy time having the next submission

Did you love the review? Let me know by giving a little bit of your precious time in rating.

Basic Requirements

✓ **The program must build an executable system monitor.**

Well done! The code builds an executable system monitor by the following commands.

1. make clean
2. make build

```
mkdir -p build
cd build && \
cmake .. && \
make
```

```
-- The C compiler identification is GNU 9.3.0
-- The CXX compiler identification is GNU 9.3.0
-- Check for working C compiler: /usr/bin/cc
-- Check for working C compiler: /usr/bin/cc - works
-- Detecting C compiler ABI info
-- Detecting C compiler ABI info - done
-- Detecting C compile features
-- Detecting C compile features - done
-- Check for working CXX compiler: /usr/bin/c++
-- Check for working CXX compiler: /usr/bin/c++ - works
-- Detecting CXX compiler ABI info
-- Detecting CXX compiler ABI info - done
-- Detecting CXX compile features
-- Detecting CXX compile features - done
-- Found Curses: /usr/lib/x86_64-linux-gnu/libcurses.so
-- Configuring done
-- Generating done
-- Build files have been written to: /home/consentsam/all-projects/02.Process Monitor/June/1306p2/CppN
D-System-Monitor-Project-Updated (1)/CppND-System-Monitor-Project-Updated/build
make[1]: Entering directory '/home/consentsam/all-projects/02.Process Monitor/June/1306p2/CppND-System
-Monitor-Project-Updated (1)/CppND-System-Monitor-Project-Updated/build'
make[2]: Entering directory '/home/consentsam/all-projects/02.Process Monitor/June/1306p2/CppND-System
-Monitor-Project-Updated (1)/CppND-System-Monitor-Project-Updated/build'
make[3]: Entering directory '/home/consentsam/all-projects/02.Process Monitor/June/1306p2/CppND-System
-Monitor-Project-Updated (1)/CppND-System-Monitor-Project-Updated/build'
Scanning dependencies of target monitor
make[3]: Leaving directory '/home/consentsam/all-projects/02.Process Monitor/June/1306p2/CppND-System-
Monitor-Project-Updated (1)/CppND-System-Monitor-Project-Updated/build'
make[3]: Entering directory '/home/consentsam/all-projects/02.Process Monitor/June/1306p2/CppND-System
-Monitor-Project-Updated (1)/CppND-System-Monitor-Project-Updated/build'
[ 12%] Building CXX object CMakeFiles/monitor.dir/src/format.cpp.o
[ 25%] Building CXX object CMakeFiles/monitor.dir/src/linux_parser.cpp.o
[ 37%] Building CXX object CMakeFiles/monitor.dir/src/main.cpp.o
[ 50%] Building CXX object CMakeFiles/monitor.dir/src/ncurses_display.cpp.o
[ 62%] Building CXX object CMakeFiles/monitor.dir/src/process.cpp.o
[ 75%] Building CXX object CMakeFiles/monitor.dir/src/processor.cpp.o
[ 87%] Building CXX object CMakeFiles/monitor.dir/src/system.cpp.o
[100%] Linking CXX executable monitor
make[3]: Leaving directory '/home/consentsam/all-projects/02.Process Monitor/June/1306p2/CppND-System-
Monitor-Project-Updated (1)/CppND-System-Monitor-Project-Updated/build'
[100%] Built target monitor
make[2]: Leaving directory '/home/consentsam/all-projects/02.Process Monitor/June/1306p2/CppND-System-
Monitor-Project-Updated (1)/CppND-System-Monitor-Project-Updated/build'
make[1]: Leaving directory '/home/consentsam/all-projects/02.Process Monitor/June/1306p2/CppND-System-
Monitor-Project-Updated (1)/CppND-System-Monitor-Project-Updated/build'
```

✓ **The program must build without generating compiler warnings.**

Well done!
As you can see in the screenshot attached the program builds without warning

Add To Knowledge

When you write a program, the compiler will check to ensure that you have followed the rules of the language in which you have written code.
If you have done something that definitively violates the rules of the language, during compilation the compiler will emit an error, providing both line number containing the error, and some text about what was expected vs what was found. The actual error may be on that line, or on a preceding line. Once you've identified and fixed the erroneous line(s) of code, you can try compiling again.

In other cases, the compiler may find code that seems like it might be in error, but the compiler can't be sure (remember the motto: "trust the programmer"). In such cases, the compiler may opt to issue a warning. Warnings do not halt compilation but are notices to the programmer that something seems amiss.

In most cases, warnings can be resolved either by fixing the error the warning is pointing out or by rewriting the line of code generating the warning in such a way that the warning is no longer generated.

In rare cases, it may be necessary to explicitly tell the compiler to not generate a particular warning for the line of code in question. C++ does not support an official way to do this, but many individual compilers (including Visual Studio and GCC) offer solutions (via non-portable #pragma directives) to temporarily disable warnings.

Some of the best practices about compiler warnings :

1. **Don't let warnings pile up. Resolve them as you encounter them (as if they were errors).**
2. **Turn your warning levels up to the maximum, especially while you are learning. It will help you identify possible issues.**
3. **It is also possible to tell your compiler to treat all warnings as if they were errors (in which case, the compiler will halt compilation if it finds any warnings). This is a good way to enforce the recommendation that you should fix all warnings (if you lack self-discipline, which most of us do).**

For More Visit these links

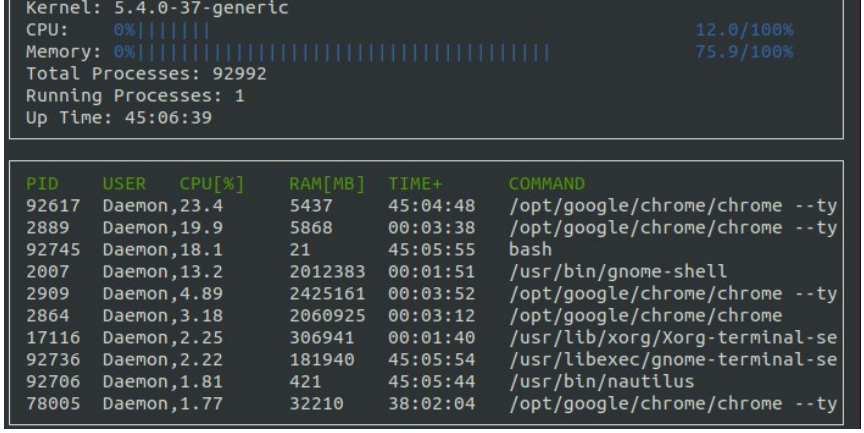
1. Link 1: Configuring your compiler: Warning and error levels
2. Link 2: Suppressing the warning
3. Link 3: Guidelines on warning
4. Intel® C++ Compiler 19.0 Developer Guide and Reference

✓ **The system monitor must run continuously without error, until the user terminates the program.**

Your program runs very much fine when i run the command

```
./build/monitor
```

```
OS: Ubuntu 20.04 LTS
```



I do not get any errors while running. I have left running your program for around 10 minutes and i do not see any errors on my system. I am using Ubuntu 20.04

I think you can shorten the length of the command by using the `substr` function and `if` statements because as you can see here it does not look nice as one some commands are too much longer in length and some are of small size.

Rest everything is pretty awesome!

✓ **The project should be organized into appropriate classes.**

Well done! 👏🏼👏🏼
Your project is well organised and meets the professional developer standard.

Add To Knowledge

Here is a link to describing the project structure in case you were wondering why the folders are named so and why it is arranged like that.

System Requirements

✓ **The system monitor program should list at least the operating system, kernel version, total number of processes, number of running processes, and up time.**

Well done! 👏🏼👏🏼
i see all the data like **operating system, kernel version, the total number of processes, number of running processes, and uptime** on the terminal easily.
Attached is the screenshot:

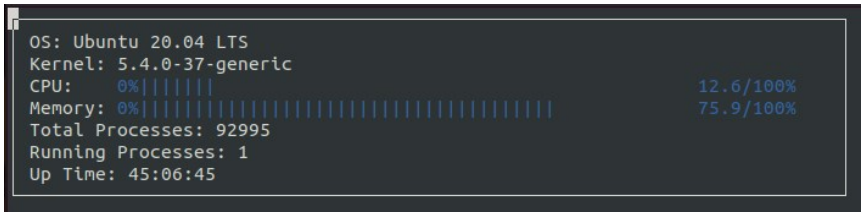
✓ **The System class should be composed of at least one other class.**

Well done! 👏🏼👏🏼
Your project structure meets the rubric specifications since the class LinuxParser has been used to define most of the functions in System class.

Processor Requirements

✓ **The system monitor should display the CPU utilization.**

Well done! 👏🏼👏🏼
Your project displays the CPU Utilisation of the system as can be seen in the attached screenshot.



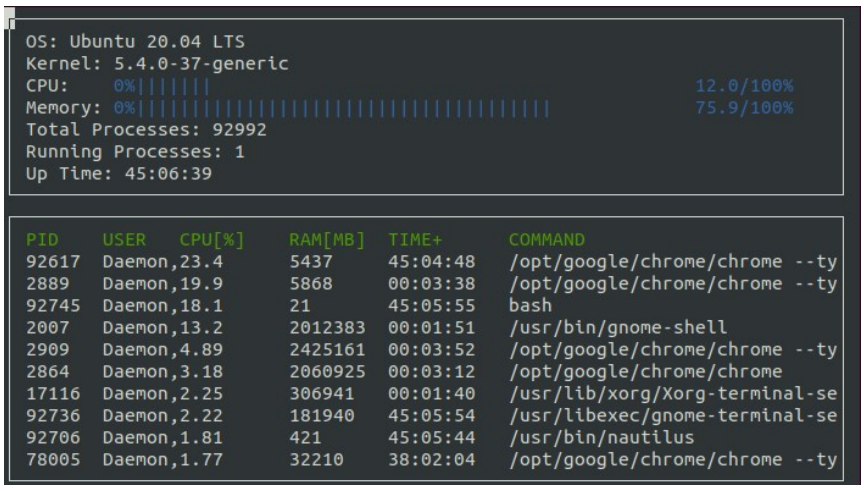
Process Requirements

✓ **The system monitor should display a partial list of processes running on the system.**

As mentioned in the previous rubrics your projects display a partial list of processes running on the system.

I see you have sorted the processes according to the CPU usage intensity, although that is not the part of the project rubric.It is so nice to see this temperament among students. ❤️

I think you have enjoyed developing this project.
❤️



🔄 **The system monitor should display the PID, user, CPU utilization, memory utilization, up time, and command for each process.**

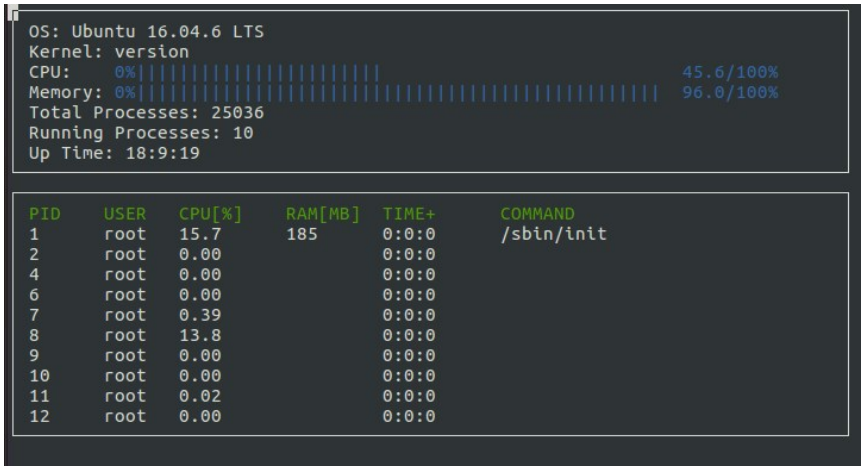
As far as i see your system monitor shows everything on terminal

1. PID
2. user (❌Please see the comment in the function `User:` in the file `linux_parser.cpp`
3. CPU utilization
4. memory utilization (❌ The process RAM Utilisation is being shown in KB and not MB which should be there because the `RAM(MB)` has been mentioned in the process monitor.So you will need to divide by 1024 in order to convert the KB into MB)
5. up time (❌ Please check the comment in the file `linux_parser.cpp` in the function `UpTime(int pid)`)
6. command

Add To Knowledge

If you would not have sorted the processes according to any condition then they must have been in the increasing order of PID and then you must have seen PID number 2,3 and then all.

You would see something like this:



In that case, you would notice that you are not able to see the command for most of the processes neither the RAM usage. There is nothing wrong in your implementation of any function command or anything.

It so happens that when you select the processes the processes get stored in the vector sorted by PID and so when you take 10 PIDs from that list then first tens which are running lists out.

if you investigate the files associated with these pids then you will notice that the files in some of the PIDs are different than the files in some normal processes PIDs.

There is an excerpt taken from Linux man pages:

```
/proc/[pid]/cmdline
              This read-only file holds the complete command line for the
```

```
                process, unless the [process is a zombie](https://en.wikipedia.org/wiki/Zombie
_process).

                In the latter case, there is nothing in this file: that is,
                a read on this file will return 0 characters.  The command-line
                arguments appear in this file as a set of strings separated by null
                bytes ('\0'), with a further null byte after the last string.
```

Zombie Process:
A process in Unix or Unix-like operating systems becomes a zombie process when it has completed execution but one or some of its entries are still in the process table. If a process is ended by an "exit" call, all memory associated with it is reallocated to a new process; in this way, the system saves memory. But the process' entry in the process table remains until the parent process acknowledges its execution, after which it is removed. The time between the execution and the acknowledgement of the process is the period when the process is in a zombie state.

When a process dies on Linux, it isn't all removed from memory immediately — its process descriptor stays in memory (the process descriptor only takes a tiny amount of memory - That is why you can see the memory column is showing almost nothing at all).

A zombie process is also known as a defunct process.

PS - Do not worry if you do not understand some of the terms and it is perfectly normal because these terms are taught in a course Operating Systems a that too is a vast course in itself. I just gave you a glimpse so that you know that there is nothing wrong in your implementation and let you know why it is happening.

Once you sort the processes based on some properties (maybe based on the usage of CPU utilisation) then vanishes away!

RESUBMIT PROJECT

DOWNLOAD PROJECT

24 CODE REVIEW COMMENTS

Learn the best practices for revising and resubmitting your project.

RETURN TO PATH