

[Return to Classroom](#)

Process Monitor

REVIEW

CODE REVIEW 22

HISTORY

Requires Changes

2 specifications require 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 the completion of this project.

- You need to make sure that you do not multiply by 100 to get the process CPU usage percentage
- You need to make sure that you are able to show the correct system CPU usage

Disclaimer

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. Please note that all the required changes / suggestions given in your project is given with the intent to improve the software engineer within you. It is not meant to demotivate you or insult your work!

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:

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.To Do

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 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 al.](#)

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

```
rm -rf 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
-- Detecting C compiler ABI info
-- Detecting C compiler ABI info - done
-- Check for working C compiler: /usr/bin/cc - skipped
-- Detecting C compile features
-- Detecting C compile features - done
-- Detecting CXX compiler ABI info
-- Detecting CXX compiler ABI info - done
-- Check for working CXX compiler: /usr/bin/c++ - skipped
-- 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/work/Downloads/sys/3110_3252509/archive/MukuFlash03-Linux-Process-Monitor-eb1f577/build
make[1]: Entering directory '/home/work/Downloads/sys/3110_3252509/archive/MukuFlash03-Linux-Process-Monitor-eb1f577/build'
make[2]: Entering directory '/home/work/Downloads/sys/3110_3252509/archive/MukuFlash03-Linux-Process-Monitor-eb1f577/build'
make[3]: Entering directory '/home/work/Downloads/sys/3110_3252509/archive/MukuFlash03-Linux-Process-Monitor-eb1f577/build'
make[3]: Leaving directory '/home/work/Downloads/sys/3110_3252509/archive/MukuFlash03-Linux-Process-Monitor-eb1f577/build'
make[3]: Entering directory '/home/work/Downloads/sys/3110_3252509/archive/MukuFlash
```

```
03-Linux-Process-Monitor-eb1f577/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/work/Downloads/sys/3110_3252509/archive/MukuFlash03-Linux-Process-Monitor-eb1f577/build'
[100%] Built target monitor
make[2]: Leaving directory '/home/work/Downloads/sys/3110_3252509/archive/MukuFlash03-Linux-Process-Monitor-eb1f577/build'
make[1]: Leaving directory '/home/work/Downloads/sys/3110_3252509/archive/MukuFlash03-Linux-Process-Monitor-eb1f577/build'
```

Add To Knowledge : CMake and Make

In the beginning, it is usually not very clear what those build systems or build system generators are and how they can be beneficial to us.

I would like to share with you some resources to help with this.

- Overview of [CMake](#) and how it manages cross-platform build processes.
- Introduction to CMake by [examples](#) that apply single file project, and multiple directory projects.

Please note that CMake isn't a build system, it's a *build system generator*. This is why we need to invoke make after running cmake. Running cmake generates Makefiles with the appropriate platform dependencies, and running make actually uses them.

However we don't need to write Make files, as CMake does this for us, it is good to understand about build systems.

- Introduction to [Make](#) and how it uses the Makefile generated by CMake to build the system.
- How to [write Makefiles](#). This is just for your information. It is already automated through CMake.

The program must build without generating compiler warnings.

Tip: Verify the build log attached; the program should build without compiler warnings.

Well done!

As you can see in the build log 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

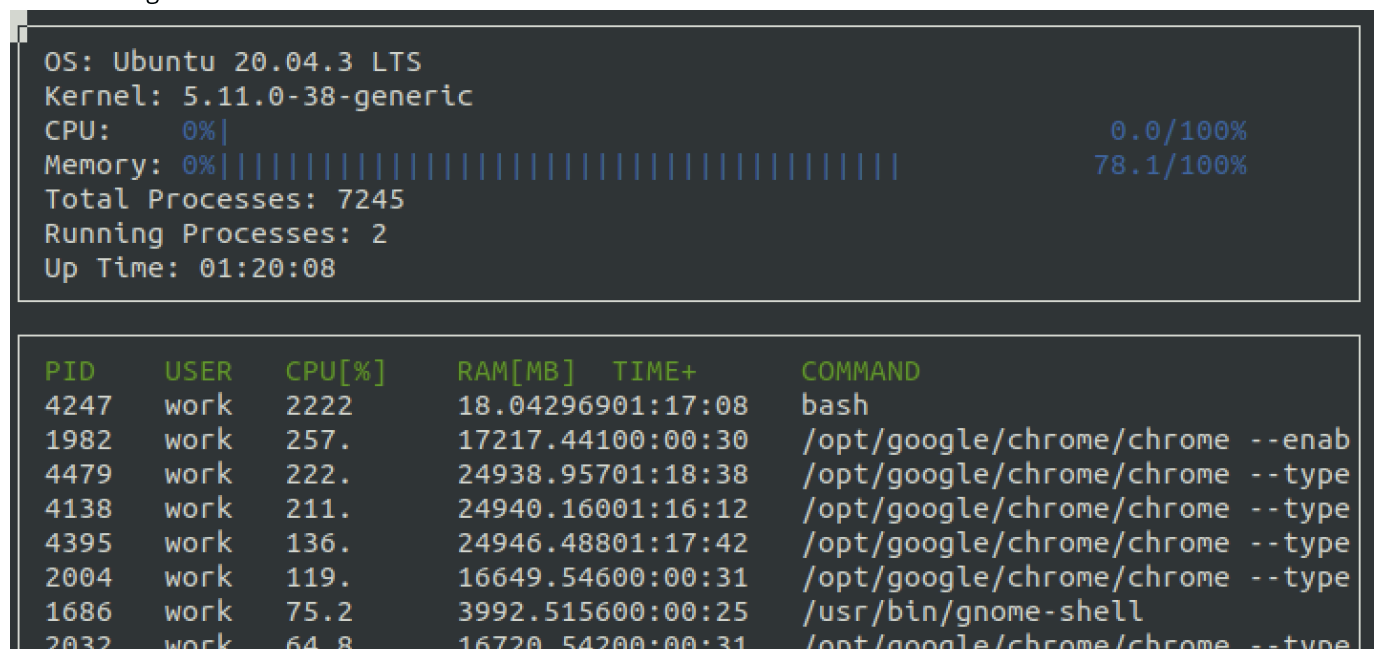
1. [Link 1: Configuring your compiler: Warning and error levels](#)
2. [Link 2: Suppressing the warning](#)
3. [Link 3: Guidelines on warning](#)
4. [Intel® C++ Compiler 19.0 Developer Guide and Reference](#)

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
```

The following is the attached screenshot



```
OS: Ubuntu 20.04.3 LTS
Kernel: 5.11.0-38-generic
CPU: 0%| 0.0/100%
Memory: 0%| 78.1/100%
Total Processes: 7245
Running Processes: 2
Up Time: 01:20:08
```

PID	USER	CPU[%]	RAM[MB]	TIME+	COMMAND
4247	work	2222	18.04	296901:17:08	bash
1982	work	257.	17217.44	100:00:30	/opt/google/chrome/chrome --enab
4479	work	222.	24938.95	701:18:38	/opt/google/chrome/chrome --type
4138	work	211.	24940.16	001:16:12	/opt/google/chrome/chrome --type
4395	work	136.	24946.48	801:17:42	/opt/google/chrome/chrome --type
2004	work	119.	16649.54	600:00:31	/opt/google/chrome/chrome --type
1686	work	75.2	3992.51	5600:00:25	/usr/bin/gnome-shell
2032	work	64.8	16720.54	200:00:31	/opt/google/chrome/chrome --type

```
2026 work 62.8 24949.79200:00:32 /opt/google/chrome/chrome --type
2026 work 60.7 16914.30800:00:31 /opt/google/chrome/chrome --type
```

I do not get any errors while running. I have left running your program for around 5 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.

Also go through this link: Writing Programs with [ncurses](#). This will give you an idea of how the ncurses library has been used in this project and then you can customize your project by making the changes in the `ncurses_display` class

The project should be organized into appropriate classes.

Well done! 🙌🙌

Your project is well organized 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, memory utilization, total number of processes, number of running processes, and uptime.

Tip:

- Do not hard code the source code, the data for each process should be read from the system only.
- You need to format the uptime properly. Refer to the comments mentioned in `format.cpp` for formatting the uptime.

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:

```
OS: Ubuntu 20.04.3 LTS
Kernel: 5.11.0-38-generic
CPU: 0% | 0.0/100%
```

```
Memory: 0%| 78.2/100%
Total Processes: 7246
Running Processes: 1
Up Time: 01:20:14
```

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.

Add To Knowledge : relationships among classes in object-oriented programming

Object-oriented programming generally supports 4 types of relationships that are: inheritance, association, composition, and aggregation. All these relationships is based on "is a" relationship, "has-a" relationship, and "part-of" relationship. This is a very important topic to get a good grip on. Please refer to this [page](#) for more details on it.

Processor Requirements

The system monitor should display the CPU utilization.

I am not able to see CPU percentage values other than 0%. Please debug where you are going wrong and then fix it! If you need any help/hint with the same, take the help of the mentors at [Knowledge Hub](#)

```
OS: Ubuntu 20.04.3 LTS
Kernel: 5.11.0-38-generic
CPU: 0%| 0.0/100%
Memory: 0%| 78.2/100%
Total Processes: 7246
Running Processes: 1
Up Time: 01:20:14
```

Process Requirements

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

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

```
OS: Ubuntu 20.04.3 LTS
Kernel: 5.11.0-38-generic
CPU: 0%| 0.0/100%
Memory: 0%| 78.1/100%
Total Processes: 7245
Running Processes: 2
Up Time: 01:20:08
```

PID	USER	CPU[%]	RAM[MB]	TIME+	COMMAND
4247	work	2222	18.042969	01:17:08	bash
1982	work	257.	17217.441	00:00:30	/opt/google/chrome/chrome --enab
4479	work	222.	24938.957	01:18:38	/opt/google/chrome/chrome --type
4138	work	211.	24940.160	01:16:12	/opt/google/chrome/chrome --type
4395	work	136.	24946.488	01:17:42	/opt/google/chrome/chrome --type
2004	work	119.	16649.546	00:00:31	/opt/google/chrome/chrome --type
1686	work	75.2	3992.515	00:00:25	/usr/bin/gnome-shell
2032	work	64.8	16720.542	00:00:31	/opt/google/chrome/chrome --type
2174	work	62.8	24949.792	00:00:32	/opt/google/chrome/chrome --type
2026	work	60.7	16914.308	00:00:31	/opt/google/chrome/chrome --type

Add To Knowledge: Resources related to processes

- Here is a general reference about what a [process](#) is from a computer science point of view.
- This resource looks at processes more from a [Linux-based operating system](#) which is very related to your project.

The system monitor should display the PID, user, CPU utilization, memory utilization, uptime, and command for each process.

Tip:

- Do not hard code the source code, the data for each process should be read from the system only.
- You need to properly format the uptime, refer to the comments mentioned in `format.cpp` for formatting the uptime.

As far as I see your system monitor show the following metrics on the terminal

1. PID
2. user
3. CPU utilization ❌ Please check out the comments in `process.cpp`. You need to make sure that you do not return it in the percentage because in the file `process_display.cpp` in its line number 78 for the processes it has

- return it in the percentage because in the file `courses_display.cpp` in its line number 76 for the processes it has already been multiplied by 100 to get the processes in 100 percent.)
- 4. memory utilization
 - 5. up time
 - 6. command

as can be seen in the following screenshot.

```
OS: Ubuntu 20.04.3 LTS
Kernel: 5.11.0-38-generic
CPU: 0%| 0.0/100%
Memory: 0%| 78.1/100%
Total Processes: 7245
Running Processes: 2
Up Time: 01:20:08
```

PID	USER	CPU[%]	RAM[MB]	TIME+	COMMAND
4247	work	2222	18.042969	01:17:08	bash
1982	work	257.	17217.44	100:00:30	/opt/google/chrome/chrome --enab
4479	work	222.	24938.95	701:18:38	/opt/google/chrome/chrome --type
4138	work	211.	24940.16	001:16:12	/opt/google/chrome/chrome --type
4395	work	136.	24946.48	801:17:42	/opt/google/chrome/chrome --type
2004	work	119.	16649.54	600:00:31	/opt/google/chrome/chrome --type
1686	work	75.2	3992.51	5600:00:25	/usr/bin/gnome-shell
2032	work	64.8	16720.54	200:00:31	/opt/google/chrome/chrome --type
2174	work	62.8	24949.79	200:00:32	/opt/google/chrome/chrome --type
2026	work	60.7	16914.30	800:00:31	/opt/google/chrome/chrome --type

 RESUBMIT

 DOWNLOAD PROJECT





Best practices for your project resubmission

Ben shares 5 helpful tips to get you through revising and resubmitting your project.

[Watch Video](#) (3:01)

RETURN TO PATH

Rate this review

START