UDACITY

Return to Classroom

DISCUSS ON STUDENT HUB

Process Monitor

REVIEW

CODE REVIEW 3

HISTORY

Meets Specifications

Congratulations

Well Done! Congratulations on completing this project. I see you have fixed all the required changes marked in the previous review.You have done a good job on this project! You understand all parts of this project well. All the required methods, functions, and attributes are implemented and the project executes well.

Since I have provided enough suggestions on your first submission I am skipping this time!

If you feel this project was mostly repetitive or boring at all.
Here is a problem statement which I am sure you will like it. Please find it here Elevator System
This question was asked from me when I was starting as a fresher. The problem is not difficult but the point is to write Clean modular code for future extensibility.
Please follow the instructions and you will be good to go.

---

1. Preparing ReadMe: Before posting your project online

Now you should post this project on GitHub with a very nice readme file so that even a newbie can understand this project.
Your code is well structured and formatted but i think it would have been much better if the code were well commented

commented.

1. Make Read Me
2. Here is my friend's blog on how to write a readme

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

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
-- 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
-- Looking for wsyncup in /usr/lib/x86_64-linux-gnu/libcurses.so
-- Looking for wsyncup in /usr/lib/x86_64-linux-gnu/libcurses.so - found
-- Looking for cbreak in /usr/lib/x86_64-linux-gnu/libncurses.so
-- Looking for cbreak in /usr/lib/x86_64-linux-gnu/libncurses.so - found
-- Found Curses: /usr/lib/x86_64-linux-gnu/libncurses.so
-- Configuring done
-- Generating done
-- Build files have been written to: /home/work/Downloads/sys/1709_3167270/archiv
```

```
e/syashasvisai-SystemMonitor-e3b8e45/build
make[1]: Entering directory '/home/work/Downloads/sys/1709_3167270/archive/syasha
svisai-SystemMonitor-e3b8e45/build'
make[2]: Entering directory '/home/work/Downloads/sys/1709_3167270/archive/syasha
svisai-SystemMonitor-e3b8e45/build'
make[3]: Entering directory '/home/work/Downloads/sys/1709_3167270/archive/syasha
svisai-SystemMonitor-e3b8e45/build'
Scanning dependencies of target monitor
make[3]: Leaving directory '/home/work/Downloads/sys/1709_3167270/archive/syashas
visai-SystemMonitor-e3b8e45/build'
make[3]: Entering directory '/home/work/Downloads/sys/1709_3167270/archive/syasha
svisai-SystemMonitor-e3b8e45/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/1709_3167270/archive/syashas
visai-SystemMonitor-e3b8e45/build'
[100%] Built target monitor
make[2]: Leaving directory '/home/work/Downloads/sys/1709_3167270/archive/syashas
visai-SystemMonitor-e3b8e45/build'
make[1]: Leaving directory '/home/work/Downloads/sys/1709_3167270/archive/syashas
visai-SystemMonitor-e3b8e45/build'
```

✓

The program must build without generating 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

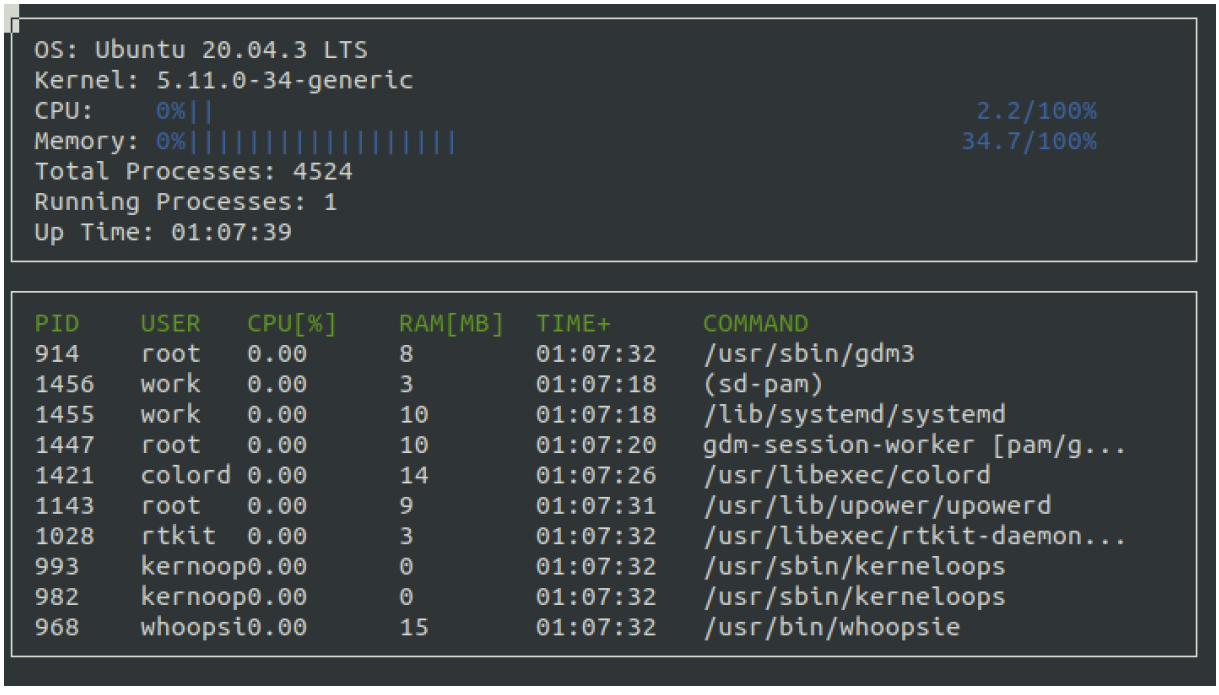
✓

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



I do not get any errors while running.

✓

The project should be organized into appropriate classes.

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.

Processor Requirements

✓

The system monitor should display the CPU utilization.

Well done! 👏👏
Your project displays the CPU utilization 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

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

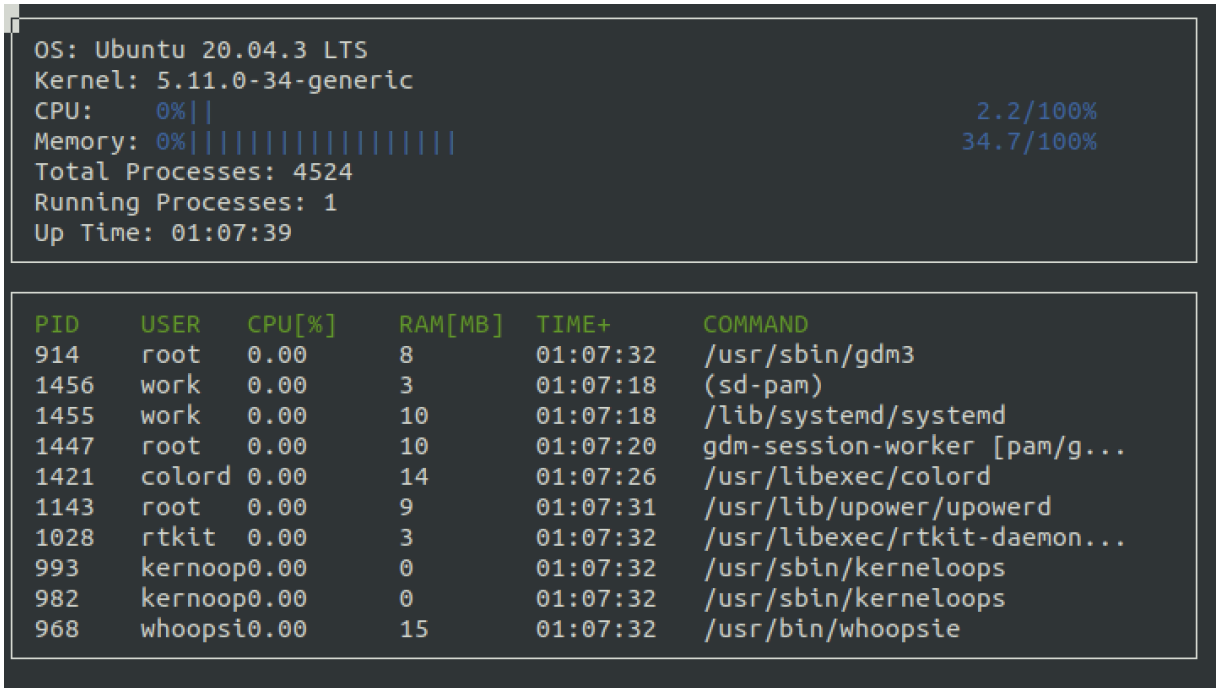
✓

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 show the following metrics on the terminal

1. PID
2. user
3. CPU utilization
4. memory utilization
5. up time
6. command

as can be seen in the following screenshot.



DOWNLOAD PROJECT

3 CODE REVIEW COMMENTS ›

RETURN TO PATH

Rate this review

START