Multi-threaded Programming

# Description

Build a simulated logging system for a security system. The idea is to build a multi-threaded program in a “safe” manner such that threads do not corrupt each other’s data.

The program will have multiple threads – each simulating a process that needs to write entries to a log file. They all write to the same log file.

## ~~SecuritySystem Class~~

~~This is the code that will actually be the simulated processes. Create a class with static properties and static methods. The class does not need a constructor as it is only intended to be used in a static manner.~~

~~The class has a static bool variable that gets set to true when the system should shut down.~~ ~~Provide a setter function for setting it. (It should start as false.)~~

~~SecuritySystem also has an output file stream pointer. Provide a static method to set this – the main program can open the file for writing, then pass a pointer to it using the set method.~~

### ~~monitor()~~

The class will also have a function called monitor(name). The parameter (a string) is the name of the item being monitored (ex. “server room door”, “air conditioning unit #5”). When called this function will

* ~~call the log(name, message) function to write a “starting up” message to the log file~~
* ~~then it will generate a random number (something between 0 and 2 seconds gives reasonable results~~ without having to wait too long to get results – if you aren’t seeing any thread collisions try a smaller max sleep time or add more threads) and sleep for that amount of time
  + pretend that this is a process waiting for some sort of notification event from a security sensor
  + the pretend event from the thing being monitored is simulated by the thread finishing sleeping
* ~~then it will call the log() function to write the “simulated event” entry~~
* ~~then it will check the shutdown property. If the shutdown property is false, it will loop back and generate a new random length of time to sleep (etc.)~~

~~When the shutdown property is true the function will write a “shutting down” message to the log file – and then end.~~

### ~~Log Entries~~

Log entries should contain the ~~date-time (see help notes at the end) of the entry, the name of the thing being monitored, and what the event was (for the simulated events – just write “simulated event”). They should also use a delimiter between the different values so they can be opened in a spreadsheet (etc.) for analysis. Mine uses tab “\t”.~~

#### Ex. Log Entry

|  |
| --- |
| 2019-11-28 20:37:46 Simulated Event Motion Sensor #2 |

### log() function

The log function takes ~~two string arguments~~: ~~the name of the monitor and the message~~. ~~It writes the date-time~~, ~~the message, and the name of the monitor~~.

First write this function in a “thread **unsafe**” manner where any thread could be executing it simultaneous with other threads – possibly interrupting their write. Examine the log file – you should get some bad results as long as the threads are awake at the same time. Copy this “bad” log file to be submitted with your source code.

Then modify the log() function in a “thread safe” manner where only one thread is allowed to execute the code that writes to the file at a time. Examine the log file again – you should not see any “clobbered” results. Upload a “good” log file with your source code as well.

## Main Program

~~The main program opens the file and calls the SecuritySystem setter function that sets the log file.~~

~~Then it creates at least five threads using the monitor() function. You can leave these threads as simple variables, or store them in an array or container.~~

~~Then the main function prints a message like “Press Enter to Continue” and waits for input from the user (you can do this any way you like as long as your threads continue running). When the user responds, the main function uses the SecuritySystem function that sets the shutdown property of the SecuritySystem to True – which will result in the entire system shutting down.~~

# Technical Requirements

Write this program using header (.h) files and separate each class into its own pair of files.

# Testing

Make sure that all of your code has been tested

* Every method in your code should be tested at least once.
* Remember when testing ranges of values to test above and below the allowable range, and exactly at the end points of the allowable range.

# Submission

Submit five files:

1. the source code for your main function
2. the .h file for the SecuritySystem class
3. the .cpp file for the SecuritySystem class
4. the log file with munged results
5. the good log file

# Style

Your program must be neat, well formatted, and readable (see the style guide on iLearn). Remember you can lose up to 30% for poor style. Don’t forget comments that identify the programmer, the program, and the date the program was written.

## Helpful code

Figuring out how to get formatted system date-time took a while – especially since there are “insecure” methods for this and a lot of help on the Internet using those insecure methods. Here is a function that returns the current time in 12-hour format using a secure method. Follow this [link](http://www.cplusplus.com/reference/ctime/strftime/) to see how to build other formats. You can use this function exactly as it is except with a modified format (you should credit the author – Jim – in your comments) or manage the date-time your own way.

|  |  |
| --- | --- |
| string timeString()  {  char timeDate[20];  time\_t rawtime = time(0);  struct tm timeinfo;  localtime\_s(&timeinfo, &rawtime);  strftime(timeDate, 20, "%I:%M:%S", &timeinfo);  return timeDate;  } | Code to return a time string  more formats at <http://www.cplusplus.com/reference/ctime/strftime/> |
| this\_thread::sleep\_for(chrono::milliseconds(500)); | Code to make the current thread sleep 500 milliseconds |