CS 3341 FOUNDATIONS OF COMPUTING: LAB 1 MULTITHREADED JOB SYSTEM: COMPILER

Overview: Design and develop a general-purpose job system that is capable of compiling and executing c++ code, as well as providing a detailed report via a json file on any errors and warnings from the compiling and linking process.

Repo Link:

Due: October 2, 2023 @ 11:59pm.

Code Requirements (60%):

- Create a Multithread Job System in C++ that meets the following requirements
    - The system should be designed to be reusable and provide a singular interface for a developer to utilize in their own projects. The code should be encapsulated so it is easily integrated into other projects
    - The main system interface should allow for the following features
        - Create/Destroy Job System
        - Create/Destroy Threads
        - Create/Destroy/Track Jobs within the system
            - Allow for multiple types of custom jobs to be created and executed through the singular interface
            - Provide ability for user to get details of jobs running as well as historical information on job execution/status
        - Provide a clear thread safe memory and job management design.
            - Ensure that memory being used in main and child threads do not conflict, and are designed to prevent errors, such as race conditions and deadlocks. Utilize proper thread syncing techniques for all required memory operations. Also ensure that these thread syncing techniques are only applied on relevant and needed sections of code.
        - Code should have a clear and thoughtful design, utilizing design patterns when relevant
        - Code should be Object Oriented, and utilized class and header files, ensuring proper coding practices

- Create a Compiling system that is capable of executing a make file to compile a set (more than 1) of source files and provide detailed reporting of output. All task required should be completed in at least 1 child thread, but multiple threads can be created (see extra credit). The Compiling system should have the following properties.
    - Compile Job
        - The job should execute a make file passed to the job (a file path or file contents can be passed).
        - All errors and warning should be captured by the job and reported back to main thread
        - Job should be performed in c++
    - Error/Warning Parsing
        - All errors and warnings returned from Compile Job should be parsed into a JSON file. The file should contain the following information at a minimum
            - Error description
                - This can be broken into generic description, and then detailed description if desired
            - File path
            - Line number for error
            - Column number for error
        - All errors/warnings should be stored in JSON by file
        - Parsing task can be completed in any language, but must be automated through the system (i.e. you cannot manually call the methods, it must be part of the automated processing).
        - You can utilize any parsing or json libraries to support the requirements of the parsing job.
    - JSON Output
        - All errors returned by Parsing Job should be placed into a JSON output file
        - All errors/warnings should be sorted by file
        - Each error should include the line of text from source file where error occurred, plus at least 2 lines above and below the error line (if possible, e.g. error occurs on first or last line of file)
        - The final JSON object will be hierarchical nested arrays, with top level array elements associated to a specific file
        - All errors associated with a give file should be stored in a nested array
        - [[File 1], [File 2] … [File n]]
            - File 1: [{Error 1}, {Error 2}, … , {Error n}]
        - Output task can be completed in any language, but must be automated through the system (i.e. you can not manually call the methods, it must be part of the automated processing).

        - You can utilize any parsing or json libraries to support the requirements for this portion of the job.
    - **Extra Credit (Required for C5393):**
        - Make each of the above task a separate job, capable of being processed in parallel
            - Compile Job, Error/Warning Parsing Job, JSON Output Job
- You must provide a make file that will compile the target code within your project.
- You must provide example code to compile (that can be called from make file) that will show off outputs when errors and warnings are present as well as when the code compiles with no errors or warnings.
- You must demonstrate errors/warnings being captured from multiple files at same time.

Report Requirements (40%):

- Report should provide a detailed description of system architecture including the following as a minimum
    - Detailed description of the memory management and thread protections paradigms used
        - Describe tradeoffs with your approach compared to others, being sure to discuss both pros and cons of the methods utilized.
    - Provide a UML diagram of the Job System
    - Provide detailed description of the high-level interface of the Job System as well as all Job types
        - This section should be written as a tutorial, providing instructions on each method, what the method does, as well as providing information on inputs and outputs.
    - Explain and document your design decision. Should include write up that explains architecture decisions and how you designed for functionality and extensibility
- Report should provide detailed demonstration of the Compile system.
    - Examples should be provided, showing sample code, sample raw errors/warning, as well as final parsed outputs.
    - Full detailed explanations of each field in the output, as well as a description of the output object returned from a completed execution of the system (both with and without compile errors and warnings

All code in the "Code" folder, raw data files generated in "Data" and your final report in "Report" folder in the git repo. If your project fails the compile action upon final push to git repo, it will receive a 0. Be sure to resolve any issues with the compile action prior to deadline.