CSCI 1730 Project 4

Multithreaded Diagonal Sums

Learning Outcomes

- Design and implement a multithreaded program.
- Design and implement programs in a Unix environment that open, close, read, and write files.
- Demonstrate an understanding of the Unix file system by changing file permissions to allow programs to open, close, read, and write files.
- Design and implement a program that uses pointers and dynamic memory allocation and deallocation.
- Use valgrind to find memory leaks in programs. Implement programs that use dynamic memory allocation and deallocation and have no memory leaks.

Problem / Exercise

The goal of this project is to give you systems programming experience writing a multithreaded program. Your program will find all of the diagonal sums equal to an input s, an unsigned long value, in an input text file consisting of a 2D grid of n-by-n digits (1 through 9), where $n \geq 1$, using t threads, where $1 \leq t \leq 3$. Your program should write the output of all of the diagonal sums equal to s to an output file.

Project Requirements

Your program must adhere to all instructions in this document, all instructions in the provided files, and all instructions stated by the lecture instructor.

1. Work on this project on a vcf node instead of odin. Put all files for this project in a folder named proj4.
2. Read through this document and the files provided for you on eLC for this project.
3. Design an algorithm to find all of the diagonal sums that equal s in an n-by-n grid using t threads that has a worst case runtime complexity of $O(n^3)$ and worst case space complexity of $O(n^2)$. We will test your code with large grid(s), and if your program doesn't complete in a reasonable amount of time, then you will lose points. Your algorithm should be based on examples provided in this document and anything discussed by the lecture instructor during lecture classes.
4. For large grids, your program using two threads should compute all of the diagonal sums faster than using a single thread. Also for large grids, your program using three threads should compute all of the diagonal sums faster than using two threads. However, this may NOT be true for small grids. Investigate this and report your findings in your README.txt file. In order to investigate this, you should create your own valid input files that should be different than the examples provided.
5. In a file named proj4.c, implement all functions found in proj4.h based on their comments and examples provided. Your proj4.c file and proj4.h file must compile with our provided main.c file on a vcf node using the provided Makefile. Do not modify the provided Makefile.
6. Your program must run correctly with the provided main.c without any modifications to the provided main.c. If your program doesn't run with the provided main.c file, then you will receive a zero on this project.
7. Use call(s) to pthread_create to create new POSIX thread(s) when more than one thread is used to compute all of the diagonal sums. Failure to use pthread_create call(s) when more than one thread is specified as a command line argument will result in a grade of zero on this project. Do not explicitly call fork anywhere in your source code. A single call to fork found anywhere in your source code will result in a grade of zero on this project.

8. You may assume command line arguments and input files will be valid. Your program must work for any valid sets of inputs and valid input files we use for grading.
9. If you encounter any runtime or logic errors while implementing your project, then use GDB to debug your program.
10. Your final program's executable, `proj4.out`, must have I/O that matches the examples exactly except for the timing output, which will vary each time your program is executed. Also, your program must correctly create (or overwrite) the output file. Failure to follow the I/O provided in the Examples section may result in a failing grade for this project.

Coding Style Requirements

1. All functions must be commented. Comments must include a brief description of what the function does, its input(s), its output, and any assumptions associated with calling it. If your function has a prototype and an implementation separated into different parts of your source code, then you only need to put the comments for that function above its prototype (there is no need to comment both the prototype and implementation; commenting the prototype is sufficient).
2. All structs, unions, and enums must be commented.
3. All global variables and static variables must be commented.
4. All identifiers must be named well to denote their functionality. Badly named identifiers are not allowed. For example, identifiers like a, aaa, b, bbb, bbbb are bad names for identifiers.
5. Every line of source code must be indented properly and consistently.

README.txt File Requirements

Make sure to include a `README.txt` file (use a .txt file extension) that includes the following information presented in a reasonably formatted way:

- Your First and Last Name (as they appear on eLC) and your 810/811#
- Instructions on how to compile and run your program. Since we are using a Makefile for this assignment, then these instructions should pretty simple based on the provided Makefile.
- Report your findings as aforementioned in your README.txt file.

Examples

The I/O of your final program must match the examples except for the timings, which will vary each time your program is executed. The input files and correct output files can be found on eLC in examples.tar.gz. The entire file, examples.tar.gz, should be uploaded to your account on a vcf node, and its contents should be extracted using the `tar` command below on a vcf node.

```
tar -xvf examples.tar.gz
```

Every byte in your output file must match the correct output file provided for each example (otherwise, you may lose a substantial amount of points).

```
./proj4.out in1.txt out1.txt 10 1; diff out1.txt correctOut1.txt | wc -c;
Computing the diagonal sums equal to 10 in a 5-by-5 grid using 1 thread(s).
Elapsed time for computing the diagonal sums using 1 thread(s): 0.000002 seconds.
Writing the diagonal sums equal to 10 to the file out1.txt.
Program is complete. Goodbye!
0

./proj4.out in2.txt out2.txt 19 2; diff out2.txt correctOut2.txt | wc -c;
Computing the diagonal sums equal to 19 in a 17-by-17 grid using 2 thread(s).
Elapsed time for computing the diagonal sums using 2 thread(s): 0.000130 seconds.
Writing the diagonal sums equal to 19 to the file out2.txt.
Program is complete. Goodbye!
0
```

```
./proj4.out in3.txt out3.txt 3 3; diff out3.txt correctOut3.txt | wc -c;
Computing the diagonal sums equal to 3 in a 672-by-672 grid using 3 thread(s).
Elapsed time for computing the diagonal sums using 3 thread(s): 0.003658 seconds.
Writing the diagonal sums equal to 3 to the file out3.txt.
Program is complete. Goodbye!
0

./proj4.out in4.txt out4.txt 13 2; diff out4.txt correctOut4.txt | wc -c;
Computing the diagonal sums equal to 13 in a 2778-by-2778 grid using 2 thread(s).
Elapsed time for computing the diagonal sums using 2 thread(s): 0.215538 seconds.
Writing the diagonal sums equal to 13 to the file out4.txt.
Program is complete. Goodbye!
0

./proj4.out in5.txt out5.txt 1222 3; diff out5.txt correctOut5.txt | wc -c;
Computing the diagonal sums equal to 1222 in a 3567-by-3567 grid using 3 thread(s).
Elapsed time for computing the diagonal sums using 3 thread(s): 11.546067 seconds.
Writing the diagonal sums equal to 1222 to the file out5.txt.
Program is complete. Goodbye!
0
```

Submission

Submit your files before the due date and due time stated on eLC. Submit your files on eLC under the Assignment named Project 4. Submit only the following files.

1. All source files required for this assignment: proj4.c and proj4.h
2. A README.txt file filled out correctly

Do not submit any compiled code. Also, do not submit any zipped files or directories. Do not submit a Makefile. Do not submit a main.c file. We will compile your submitted files with our own copies of main.c and Makefile. We only need the files mentioned above with the file extensions aforementioned.

1 Grading (30 points)

If your program does not compile on a vcf node using the provided Makefile, using the required compiler for this course (`gcc version 11.2.0`), and using an unmodified versions of the provided main.c, then you'll receive a grade of zero for this assignment. Otherwise, your program will be graded using the criteria below.

Program runs and outputs correct files with various test cases on a vcf node using 1 thread	6 points
Program runs and outputs correct files with various test cases on a vcf node using 2 threads	12 points
Program runs and outputs correct files with various test cases on a vcf node using 3 threads	12 points
README.txt file missing or incorrect in some way	-3 points
Not submitting to the correct Assignment on eLC	-3 points
Late penalty for submitting 0 hours to 24 hours late	-6 points
One or more compiler warnings	-6 points
Not adhering to one or more coding style requirements	-6 points
Valgrind identifies a memory leak (definitely, indirectly, or possibly lost)	-6 points
Source code does NOT use POSIX thread(s) when $t >= 2$	-30 points
Source code uses a call to fork	-30 points
Penalty for not following instructions (invalid I/O, etc.)	Penalty decided by grader

You must test, test, and retest your code to make sure it compiles and runs correctly on a vcf nodes with any given set of inputs. This means that you need to create many examples on your own (that are different than the aforementioned examples) to ensure you have a correctly working program. Your program's I/O must match the examples given in the Examples section except for the timings, which will vary each time your program is executed. You may assume inputs will be valid.