Programming Assignment 1: Integer Sum and Average Calculation from Multiple Files

June 19, 2025

1 Important Dates

Release Date: Jun 18th 2025 11:59 PM

Phase 1 Due Date: Jun 29th 2025 11:59 PM

Phase 2 Due Date: Jul 10th 2025 11:59 PM

2 Overview

The project will be divided into two phases. In Phase 1, the focus will be on process APIs, and directory and file handling. Phase 2 will be concentrate on inter-process communication using pipes.

3 Project Specifications

Objective : Given a directory of files include multiple integers and each of the integer occupy a separate line, develop a program that calculates the sum and average of all the integers and print the results to terminal. The calculation should be done using multiple processes: each process calculates the sum of all integers in a single file and reports the result back to the main process, and then the main process summarizes all the data and calculates the sum and average of all integers in the directory. This kind of task is common in real-world applications such as databases and machine learning.

Advisory : Make sure that the project is tested on CSE lab machines running Linux before submission. Some system call APIs may not be supported on Mac/clang/Xcode environments.

4 Tasks

4.1 Task 1: Argument Parsing and Directory Handling in the Main Process

The main process should be able to receive the path of the target directory from command line in the following format:

```
./program PATH_TO_DIR
```

The main process should extract a list of all files in the given directory.

4.2 Task 2: Child Processes Creation and Termination

The main process should fork child processes to handle files. The number of child processes forked by the main process should be equal to the number of files in the directory. The implementation of the child process should be a separate C source code file different from the file of the main process.

The child process should be able to receive the path to the file it is supposed to calculate via command line arguments. After forking each child process, it should call a `exec*()` function to execute child process's code.

The main process is also responsible for handling termination of child processes, so it can only exit when all the child processes have terminated.

4.3 Task 3: Extract Numbers from Files and Calculation

The child process should extract numbers from the file, and it should also calculate the sum of them. Each integer is stored as its own separate line in the file.

Phase 1 : Since there is no inter-process communication in this phase, each child process needs to write out its own results to a file using the same name as the original file it reads from but a different extension. For example, if the original file is named as `file1.txt`, the child process should create a file called `file1.results` and write exactly two lines in the file: the first line should be the number of integers in the file, and the second line is the sum of all integers.

Phase 2 : Child processes DO NOT need to write our results to a file. They should send the results back to the main process. There should be two things included in the message sent back to the main process: the number of integers in the file and the sum of all integers. The main process handles outputing the results, and the format is specified in 4.7.

4.4 Task 4: Inter Process Communication with Pipes

The child process sends the sum of all numbers it read and the number of integers back to the main process through a pipe and terminates.

4.5 Useful APIs

You may use but not limited to the APIs mentioned below for the project.

- Processes: `fork()`, `exec*()`, `wait()`/`waitpid()`.
- System File I/O: `open()`, `close()`, `read()`, `write()`.
- Standard File I/O: `fopen()`, `fclose()`, `fread()`, `fwrite`, you may also refer to other standard file I/Os such as `fscanf()`, `fprintf()`, `fgets()`, `fputs`.
- Directory: `mkdir()`, `opendir()`, `readdir()`, `closedir()`.
- Pipes: `pipe()`.

4.6 Assumptions/Notes

Please adhere to following assumptions:

- The code execution commands will be run at the same level as the given Makefile.
- The maximum path length will be 1024.

4.7 Expected Output

The program should only output following lines as the result (assuming the sum is XXXXX and the average is YYY):

```
filename_1: number_of_integers_in_filename_1, sum_of_integers_in_filename_1
filename_2: number_of_integers_in_filename_2, sum_of_integers_in_filename_2
...
filename_n: number_of_integers_in_filename_n,  sum_of_integers_in_filename_2
Sum: XXXXX
Average: YYY
```

Please do not include any other output in your final submission.

5 Compilation and Execution

We have provided you with a Makefile that contains the necessary rules to build the target program and also clean them.

To compile the entire code, use the following command:

```
make all
```

After compilation, run the program with the target directory as an argument:

```
./calculator PATH_TO_DIR
```

To clean the compiled executables and object files; use:

```
make clean
```

Please DO NOT add or delete source code files in the skeleton code we provided, or make changes to the Makefile.

6 Testing

There will be just one sets of testcases for this project, but it will be used in both phases.

Testcases:

- Testcase 1: An empty directory.
- Testcase 2: A directory with a single nonempty file.
- Testcase 3: A directory with several files, but one of them is empty.
- Testcase 4: A directory with 5 files, and none of them is empty.
- Testcase 5: A directory with 15 files, and none of them is empty.

7 Submission

For each phase, create a README file that answers mainly the following:

1. Team members.
2. x500.
3. Division of task across members.
4. Lab machine used for testing purpose.
5. Design specifications: Mention your project design in details and mention any challenges faced.
6. Document and explain any usage of AI tools.
7. Anything the TA should know about.

Submit a single **.tar.gz** file containing the README and template folder for each phase to Gradescope. **Only one** member of a group should submit to Gradescope. Make sure to add your group members names in Gradescope at the time of submission.

7.1 Tasks in Each Phase

Phase 1: Task 1, Task 2, and Task 3 Phase 1. The code you submitted should be compiled with `make all`, and run with the command `./calculator testcase_dir`. Since there is no inter-process communication in this phase, each child process should write intermediate results to a file with a different filename extension. For example, if the original file is named as `A.txt`, the process that reads all integers in it should create a file named `A.results` and writes exactly two lines in the file: the first line should be the number of integers in the file, and the second line is the sum of all integers.

Phase 2: Task 3 Phase 2 and Task 4. In this phase, the temporary result files are not needed anymore, so please disable the code that creates files and write results to them. The code you submitted in this phase should be compiled with `make all` and run with `./calculator testcase_dir`, and the main process should output the results to the terminal in the format specified in Section 4.7. Another requirement is all child processes should end gracefully, and the main process should wait for them to finish before exiting.

7.2 Deadlines

- Phase 1: Jun 29th 2025, 11:59 PM.
- Phase 2: Jul 10th 2025, 11:59 PM.

8 Rubrics

Use of AI Tools : Artificial intelligence (AI) language models, such as ChatGPT or CoPilot, may be used in a limited manner for programming assignments with appropriate attribution and citation. For programming assignments, you may use AI tools to help with some basic helper function code (similar to library functions). You must NOT use these tools to design your solution, or to generate a significant part of your assignment code. You must design and implement the code yourself. You must clearly identify parts of the code that you used AI tools for, providing a justification for doing so. You must understand such code and be prepared to explain its functionality. Note that the goal of these assignments is to provide you with a deeper and hands-on understanding of the concepts you learn in the class, and not merely to produce "something that works".

If you are in doubt as to whether you are using AI language models appropriately in this course, please discuss your situation with the instructor or TA. Examples of citing AI language models are available at: libguides.umn.edu/chatgpt. You are responsible for fact checking statements and correctness of code composed by AI language models.

Phase 1: 60 points

- `exec*` usage: 10 points.
- Process spawning: 10 points.

- Directory handling, single file handling: 10 points
- README: 10 points.
- Provided testcases: 20 points

Phase 2: 50 points

- Pipes 25 points.
- README: 5 points.
- Provided Testcases: 20 points.