Project 1
Systems Programming
CS3280
Fall 2023
100 points

It is all about having a positive attitude!

Purpose of this assignment

Hooray! It is time for your first project.

The purpose of this project is to help us understand multi-core programming and its benefits.

Specific objectives of the assignment are listed below -

1. Learn what type of programming can benefit from multi-core multithreaded programs.
2. Learn how to divide data for data parallelization to gain performance
3. Learn to create multithreaded programs in RUST
4. Learn to handle resource sharing in a multithreaded program
5. Learn File I/O in RUST

Description

Impressed by your awesome coding skills, company **Ay Caramba** hired you as a Systems Programmer. One of the first systems programs you will have to develop relates to sales data of their products sold in various branches of the company. The company has one (1) unique product and that product is sold in all of their 500 branches. But we will only deal with 40 branches to keep things simple.

Each branch sends weekly sales reports in text files to the main office. How they send the files is none of our concerns at the moment - it will be our concern when we delve into network programming. At this moment your only concern is to process this file to generate a complete sales report (text) file that will contain the branch code and the product code and the total quantity sold each week.

You do not have to worry about scheduling your program to run every week - all you have to do is write the program to process the files the way dictated by the company.

- **Input File Format**

  The format of the sales report file sent by each branch is given below -

  Branch code, product code, quantity sold, date sold

  Each file will have seven lines that will contain these data for 7 days.

  A concrete example of line is given below, which is taken from a file sent by the Albuquerque, NM branch -

  ALBNM, PROD001, 12, 2023-01-01

  A set of files will be given to you to test with.

The company has 500 branches - I am keeping the number of input files smaller (40) to keep things easy.
The name of each input file will be **branch_weekly_sales.txt**.

- **Output File Format**
  The format of the summary sales report file that your code will generate is given below -
  Branch code, product code, total quantity sold in a week
  Each output file will have 40 lines that will contain the total weekly sale of each branch..
  A concrete example of a line is given below -
  ALBNM, PROD001, 32
  The name of the output file has to be **weekly_sales_summary.txt.**
- **File locations**
  You will have a data folder that will contain 40 folders - one for each branch. Inside these folders, we will have the input file for the weekly sales report for the concerned branch. The input folders will be called branch1, branch2, .., branchN where N is the number of branches.
  You will have another folder in the root data folder named "**weekly_summary**" where your code will save the weekly summary report.
- **Processing Information**
  Your code will read each input file, extract the quantity sold data and add them for the seven days. This summary data has to be written to the output file in the format specified above. **Remember**, even though you have 40 input files for 40 branches, you will only generate one (1) output file with the summary that will contain each branch's total sale each week.
  We are not concerned about dates or weeks at the moment. We will simply generate an output file with total sales**.**

Details of what you will have to do

You will have to create **two versions** of the program to test which is better, following the two algorithms given.
Please do not hate me! It is actually not bad at all - much of the code in version 1 can be used in version 2.
Version 1 will be a linear program that will simply use a loop to go through all the input folders and read the files and generate the output file.
Version 2 will create 4 threads where each thread will deal with input files for 10 branches and do what the linear version of the program in version 1 does.
This will be a command line based program that will accept the name of the data folder as an argument.

The program will have a file called **main.rs** that will contain the **main function** to get the program started.
In a separate file named **lib.rs**, we will have two functions to begin with -

1. Function named **process_input_file** that will process the input file
2. Function named **write_to_summary_file** that will write to the output file.

See? It is not that bad 🙂

- **Version 1 algorithm**

Main.rs -

1. Accept, validate and parse the data folder name (with path) from the command line argument.
2. Check if the output folder exists in the data folder - if not, create it. Assume that all the input folders will be there for all the branches.
3. Start a timer
4. Call the file input function **process_input_file** in lib.rs and pass the list of folders (with path) for all the branches.
5. Output to console the message received from the input function in lib.rs file
6. Stop the timer and show the total time in the console
7. Print the following message to indicate processing of all files are done - "Phew! I am done."

lib.rs file

A. The **process_input_file** function
   a. Accept the list folders as parameters.
   b. Loop through the folder list and do the following -
      i. Open the input file in each folder
      ii. Read each line and extract the quantity sold and add it to a sum variable using an inner loop or some other better way if available..
      iii. Call the output function and pass it a string that will have the branch code, product code and the total sale separated by comma.
      Example: "ALBNM, PROD001, 32"
      iv. Close the input file.
   c. Once the outer loop is done return either of the following string message to the calling function (main) depending on the status of the file processing - "**OK**", "**ERROR**"

B. The **write_to_summary_file** function
   a. Accept a string that will have the branch code, product code and the total sale separated by comma.
   b. Open the output file

c. Write the string received to the output file
d. Close the file.

- **Version 2 algorithm**

In this version, you will move the output function (**write_to_summary_file**) to write to file from lib.rs to the main.rs file.

Main.rs -

1. Accept, validate and parse the data folder name (with path) from the command line argument.
2. Check if the output folder exists - if not, create it.
3. Create a channel to communicate with child threads that you will create in the next step. This channel is what children threads will use to pass the output string that will have the branch code, product code and the total sale separated by comma.
4. Split all the branch input folders into 4 groups with 10 folders in each group
5. Start a timer
6. In a loop create 4 threads to run the input processing function **process_input_file** in lib.rs on each of the group of folders you created in step 4 above.
    a. The file input function in lib.rs will accept a list of 10 folders (with path) for all the branches. Remember, in version 1 you already have this implemented.
    b. Make sure you wait for each thread to finish
7. Once all the threads are done processing their assigned folders,
    a. In a loop, process all the messages received in the channel from the children threads -
        i. Go through all the messages in a loop and
            1. print them on the console
            2. Write them to the output file using the **write_to_summary_file** file described in Version 1.
8. Stop the timer and print the total time in the console
9. Print the following message to indicate processing of all files are done - “Phew! I am done.”

lib.rs file

C. The **process_input_file** function

IN this version, this function will change a little bit. Instead of returning a message to the calling function, it will have to write the message to the channel received from the main thread.

a. Accept the list folders as parameters.
b. Loop through the folder list and do the following -

i. Open the input file in the folder
ii. Read each line and extract the quantity sold and add it to a sum variable
iii. Send a string that will have the branch code, product code and the total sale separated by comma to the calling thread (main) through the channel.
Example: "ALBNM, PROD001, 32"
iv. Log messages in a log file using any of the logging crates available for RUST. The name of the log file should be **log.txt** and should be placed in the root folder of the app.
    1. You should log any errors
    2. You should log the name of each of the folders you have processed in that thread.

c. Once the loop is done return either of the following string message to the calling function depending on the status of the file processing - "OK", "ERROR"

Notes:

- Both versions of your code must be in a **git repository** where you will give me access.
- We have omitted extensive error checking, unit testing, and many other standard coding practices to keep things simple so that we concentrate on our main task - multithreaded programming in RUST
- You can assume that all input files will always contain valid data
- You do not have to worry about dates, weeks, sorting by company code or anything like that.
- Make sure you use crates appropriately
- Your projects must be created and built with **cargo**
- **Do not lose these versions - we will use these code to learn I/O optimization later**

Resources you will need

- https://blog.logrocket.com/deep-dive-concurrency-rust-programming-language
- https://doc.rust-lang.org/book/ch16-02-message-passing.html
- https://doc.rust-lang.org/book/ch12-04-testing-the-librarys-functionality.html
- https://www.programiz.com/rust/thread
- https://doc.rust-lang.org/book/ch16-04-extensible-concurrency-sync-and-send.html
- https://doc.rust-lang.org/rust-by-example/std_misc/channels.html
- https://play.rust-lang.org/?version=stable&mode=debug&edition=2021&gist=a67eea2acb00d92ca6041bd185685664

- **https://medium.com/nerd-for-tech/logging-in-rust-e529c241f92e**

What you will submit

1. A zip file containing -
    a. Both versions of the code - the complete projects so that I can simply build and run them without having to do any setups.
    b. A readme file
        i. telling me how to run the programs
        ii. Telling me if there was any performance gain in version 2 over version 1.

How you will submit

- The file listed in the "**What you will submit**" section must be submitted to Moodle.
- You must name the file **sysprog_P1_stuname_stuid.zip**

How your work will be graded

Your work will be graded based on the following criteria -

1. Both versions -
    a. Contains the needed code implementing the algorithms given and compiles error free (30 points)
    b. Runs error free (15)
    c. Does what they are supposed to do (40)
    d. Logging (5 points)
2. Git repository - it is an inherent requirement that carries no points, but if you do not have it, there will be a **50 points deduction**.
3. Naming of file as specified in the "**What you will submit**" section. (5 points)
4. Clean coding, proper indentations, proper naming and so on (5 points)

I know you are awesome! Now let your work reflect this fact.