Lab 1: MPI Homework

The Message Passing Interface (MPI) is a low-level library used for parallel programming. We use MPI with the C programming language.

Submit your solution through Moodle. Your submission must include a protocol, documenting how you solved the exercise (e.g., outputs, answers to questions, issues, etc.), as well as all source files and configuration files required to build your solution.

1) Ring Around The Rosie

Task Description

Create an MPI program that passes a Token around in a circle from process 0 to process n and back to process 0.

Process 0 initializes the token to some value (e.g., -1) and sends it to the next process. The last process sends the token back to process 0, which receives the token. Each time the token is sent or received, a debug message should be printed to the command line.

Additional Questions

What is the output of the program? Submit the output as part of your protocol.

Run the program multiple times: Does the message order change?

Explain the message order, and why it changes/does not change.

2) Counting Even Numbers

Task Description

Create an MPI program that counts the even numbers in arrays. The program should perform the following steps:

1. Process 0 must ensure that the program is called with exactly 10 processes.
    a. Otherwise abort (cf., MPI_Abort)
2. Process 0 reads a 2-dimensional array of numbers from a file (“input.txt”)
    a. Use the function
    `void readData(char* fname,int lines,int** nums,int* lens)`
    b. `fname`…name of the file (i.e., “`input.txt`”)
    c. `lines`…number of lines in file (i.e., `10`)
    d. `nums`…2d-array for numbers.
        i. Must be allocated before calling the function
        ii. e.g.,
        `numbers = (int**)malloc(sizeof(int*)*FILE_LINES);`
        iii. where
        `const int FILE_LINES = 10;`
        iv. Individual lines are dynamically allocated in the function.
    e. `lens`…array for storing lengths of lines
        i. Must be allocated before calling the function
        ii. e.g., fixed
        `int lens[FILE_LINES];`
3. Each process receives a line from the 2d-array (i.e., an array)
    a. Process 0 keeps an array for itself (no sending required).

   b. The arrays should be sent using the actual length from the `lens` array.
   c. The receiving processes should probe the array size before receiving.
4. Each process counts the **even numbers** in its array.
5. The processes synchronize after all are done counting (`MPI_Barrier`).
6. Use the program from part 1 to calculate the grand total of even numbers.
   a. Process 0 initializes the token to its count.
   b. Each process adds its count to the token and sends it to the next process.
   c. In the end, process 0 receives the final count and prints it to the command line.
7. Run the program with 10 processes (or rather, one process per line)

Input File

The file "`input.txt`" stores the numbers that are counted by the processes.

File Format

The file contains integer numbers encoded as plain text separated by single spaces. At the end of each line there is no space, but just a newline. The first number in each line represents the count of numbers that follow in this line. The number of lines should be `10` (or rather, it must match the constant used in the program, i.e., `FILE_LINES`).

Example: File "input.txt"

```
7 231 12 54323 4 654 32001 43
```

→ First integer is 7 → 7 Elements: 231, 12,54323, 4, 654, 32001, 43

```
8 43 42 65 2168 213 66 6548 1
```

→ First integer is 8 → 8 Elements: 43, 42, 65, 2168, 213, 66, 6548, 1

…

Example: Reading

```
const int FILE_LINES=10; // 10 lines fixed
int lens[FILE_LINES];  // array for the line lengths
int** numbers;  // 2d array

// allocate number of lines of 2d array
numbers = (int**)malloc(sizeof(int*)*FILE_LINES);

// function fills lines with data
readData("input.txt",FILE_LINES,numbers,lens);
```

// `lens` is now an array storing the first column of `input.txt` (i.e., the line lengths)
// `numbers` is now a 2d-array storing the remaining numbers from `input.txt`

Example: Result

The file "`input.txt`" contains 34 even numbers.

Additional Questions

Is the synchronization after counting the numbers necessary, or not? Explain your answer.

Is there another (better?) way to distribute the array among processes? Explain your answer.

Optional bonus task: if you found a better way to distribute the array, implement it!