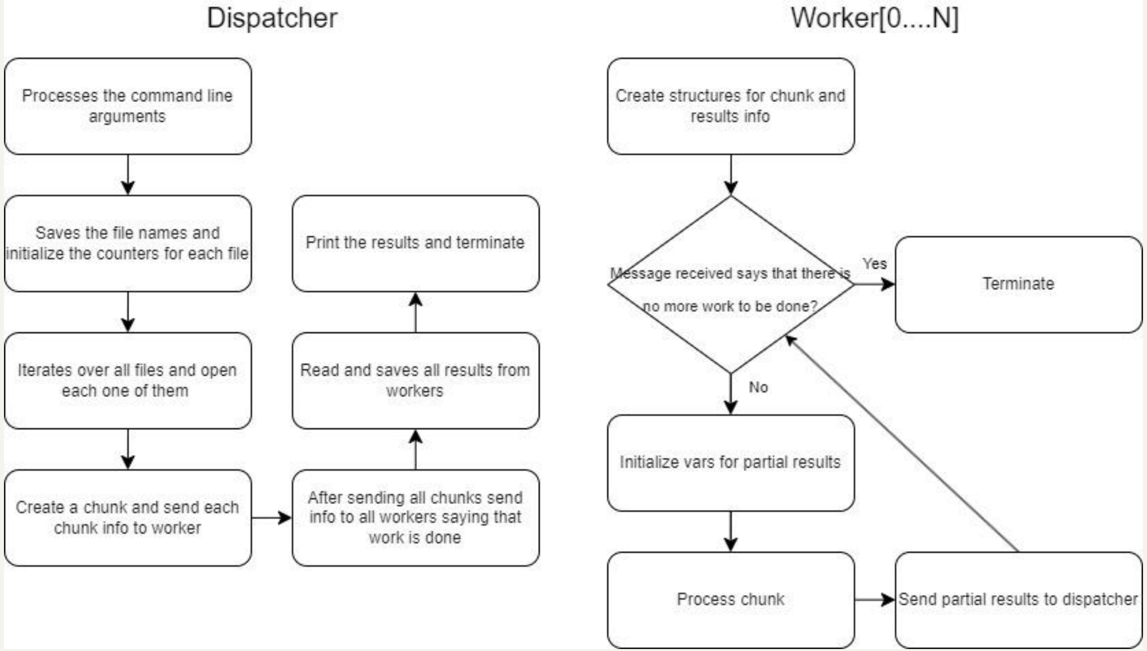
# Assignment 2 Multiprocessing

CLE - Computação em Larga Escala
Maio 2023

Eduardo Fernandes - 98512
Alexandre Pinto - 98401

# Text Processing - Multiprocessing Implementation

**Objective:** Given a text file count the total number of words, and also the number of words containing an a, e, i, o, u, y.
The multiprocessing implementation splits the work among the worker processes.



Dispatcher
Worker[0....N]
Processes the command line arguments
Saves the file names and initialize the counters for each file
Iterates over all files and open each one of them
Create a chunk and send each chunk info to worker
After sending all chunks send info to all workers saying that work is done
Read and saves all results from workers
Print the results and terminate
Create structures for chunk and results info
Message received says that there is no more work to be done?
Yes
Terminate
No
Initialize vars for partial results
Process chunk
Send partial results to dispatcher

# Text Processing - Results

Machine used in testing: Laptop LeNovo Legion, Intel Core i7-10750H Hexa Core 12 Threads, Nvidia GeForce RTX 2060, RAM 16GB

**The results below were obtained with a chunk size of 4096 and processing all text files.**

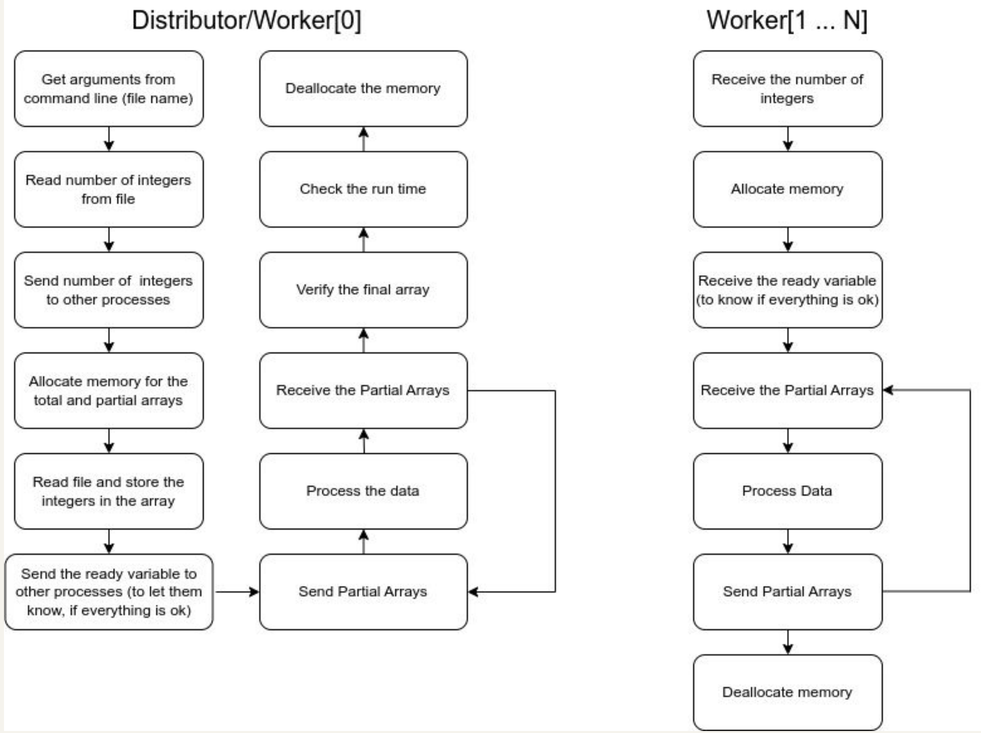
- **1 Worker** ( one dispatcher process and one worker process) : Elapsed Time = 1.52*10^-3 s;
- **2 Worker** ( one dispatcher process and two worker processes) : Elapsed Time = 8.26*10^-4 s;
- **4 Worker** ( one dispatcher process and four worker processes) : Elapsed Time = 7.52*10^-4 s;
- **8 Worker** ( one dispatcher process and eight worker processes) : Elapsed Time = **7.20*10^-4 s**;

Command:

```
        mpicc -Wall -O3 -o textProcessing textProcessing.c textProcessingFunctions.c
        mpiexec -n [number_processes] ./textProcessing -f [files to be processed]
```

# Integers Sorting - Multiprocessing Implementation

**Objective:** Given a binary file sort the integers there presented. The multiprocessing implementation uses a distributor process to split the work among the worker processes. However, the distributor is also responsible for some work, it is the worker[0].



Distributor/Worker[0]

Get arguments from command line (file name)

Read number of integers from file

Send number of integers to other processes

Allocate memory for the total and partial arrays

Read file and store the integers in the array

Send the ready variable to other processes (to let them know, if everything is ok)

Send Partial Arrays

Process the data

Receive the Partial Arrays

Verify the final array

Check the run time

Deallocate the memory

Worker[1 ... N]

Receive the number of integers

Allocate memory

Receive the ready variable (to know if everything is ok)

Receive the Partial Arrays

Process Data

Send Partial Arrays

Deallocate memory

# Integers Sorting - Results

Machine used in testing: Laptop LeNovo Legion, Intel Core i7-10750H Hexa Core 12 Threads, Nvidia GeForce RTX 2060, RAM 16GB

**Objective:** Given a binary file sort the integers there presented. The multiprocessing implementation uses a distributor process to split the work among the worker processes. However, the distributor is also responsible for some work, it is the worker[0].

| Number Processes<br>\<br>File Name | 1 | 2 | 4 | 8 |
|---|---|---|---|---|
| dataSeq32.bin | **5.23*10^-5** | 8.58*10^-5 | 9.19*10^-5 | 1.64*10^-4 |
| dataSeq256K.bin | 8.07*10^-2 | 6.70*10^-2 | **6.56*10^-2** | 8.86*10^-2 |
| dataSeq1M.bin | 3.20*10^-1 | 2.85*10^-1 | **2.44*10^-1** | 3.22*10^-1 |
| dataSeq16M.bin | 5.90 | 5.96 | **4.59** | 5.19 |

TIME

(s)

Commands:

```
mpicc -Wall -O3 sorting.c -o sorting -lm
mpirun -np [number_processes] ./sorting [file_name]
```

# Conclusion

After looking at our results, we came to the following conclusions:

- In the program 1, the increase on the number of workers will improve the run time because there are many processes doing the work.
- In the program 2, the run time tends to become faster when using more processes to sort the integers. However, in small files, the communications between processes can use more time than the sorting itself. So, it is faster to use less processes in these cases.
- Finally, the increase on the number of processes should be better when having big files to process.