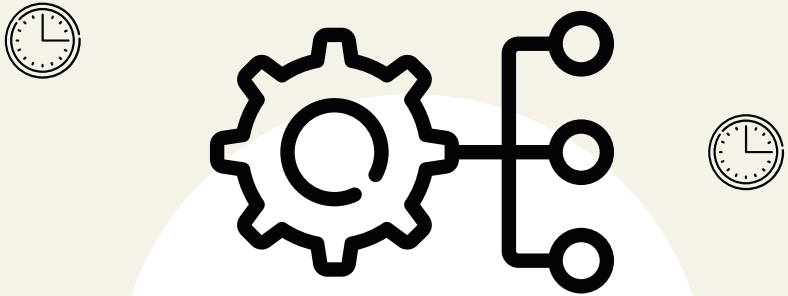
universidade de aveiro

MultiThreading

Computação em Larga Escala - Assignment I

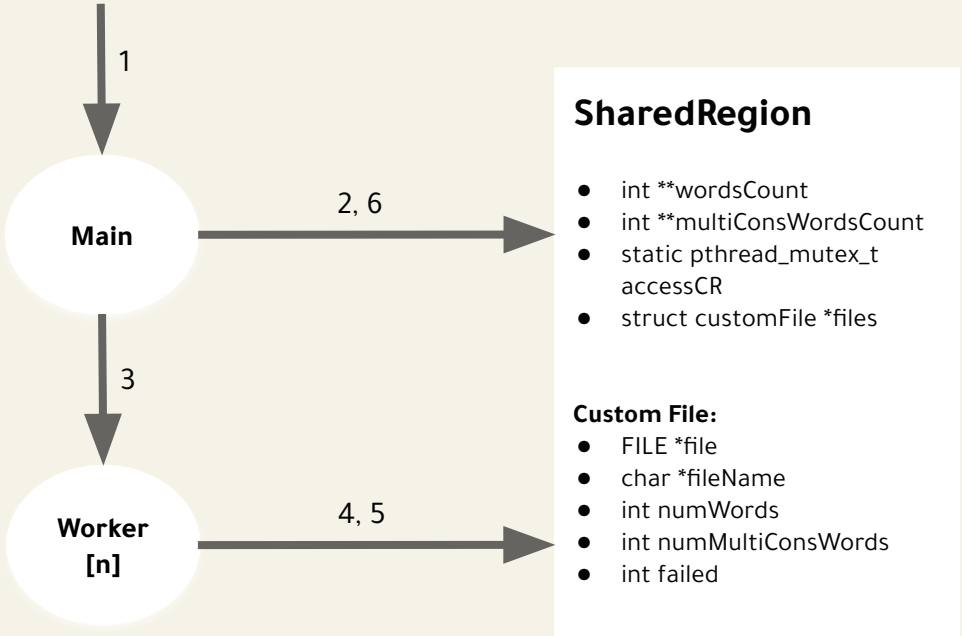


Diogo Magalhães 102470
Rafael Gil 118377

Problem 1 - Implementation

Objective: Count the total number of words and the number of words having at least two equal consonants in one or several text files.

1. Main thread processes the command line arguments.
2. Main thread adds the files to the shared region, starts the countings and the worker current file arrays.
3. Main thread creates the worker threads and waits for them to finish.
4. Worker thread gets the next chunk of data needed to be processing.
5. Worker thread processes the chunks of data and saves the partial results.
6. Main thread joins the partial results and outputs the results.



Problem 1 - Results

OS: Ubuntu **CPU:** 6 core 3.3GHz

Command: ./prog1/ex1 -f dataSet1/text*.txt -t <number-of-threads> -b <max-buffer-size>

	1 worker	*2 workers*	*4 workers*	*8 workers*
1024 bytes	0.004303	0.002503	0.001655	0.001273
8192 bytes	0.004092	0.002155	0.001426	0.000962

Command: ./prog1/ex1 -f <file> -t <number-of-threads> -b 8192

	1 worker	*2 workers*	*4 workers*	*8 workers*
text0.txt	0.000248	0.000288	0.000310	0.000524
text2.txt	0.002130	0.001182	0.000772	0.000612

Problem 1 - Conclusions

When we analyse the previous tables, we can conclude that:

- In the first table, it is possible to see the impact of using smaller and bigger buffer sizes. Since a thread, when requesting data, locks the access to data requests and since when requesting data the thread needs to process the buffer to ensure that it only contains portions with complete words, it is faster if bigger buffers are used.

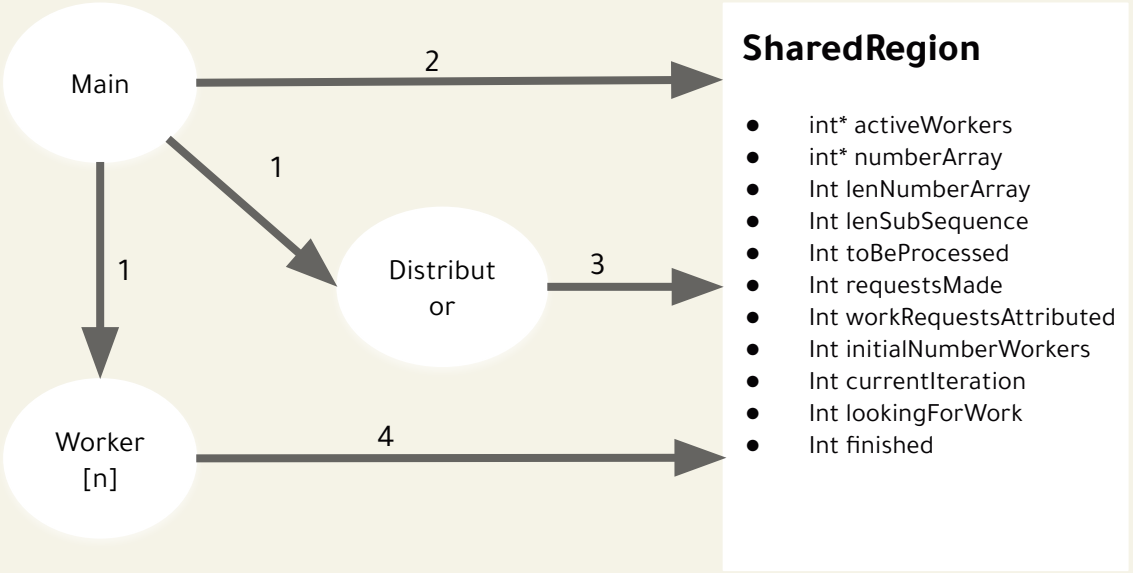
- In the second table, it is observed that depending on the size of the file, the usage of more threads might even be prejudicial to the performance of the program since for smaller files threads might be created but not used since all the file would already be processed by other threads.

- Overall, the results obtained and shown in both tables were expected and show that the performance of the text processing improved a lot with the usage of threads.

Problem 2 - Implementation

Objective: Sort, in descending order, a sequence of numbers using the bitonic sort algorithm.

1. Create the distributor and the worker threads, wait for them to terminate.
2. Store the name of the binary files whose contents is to be sorted, check in the end if the sequence of values is properly sorted.
3. Read the sequence of integers from the binary file, distribute sub-sequences of it to the worker threads.
4. Sort sub-sequences of values.



Problem 2 - Results

OS: Mint **CPU:** 8 core 2.2GHz

	1 worker	*2 workers*	*4 workers*	*8 workers*
32	0.00091	0.001268	0.001741	0.002865
256K	0.07482	0.04653	0.02213	0.02776
1M	0.2464	0.1312	0.0944	0.0907
16M	5.2986	2.9741	1.9119	1.6382

Problem 2 - Conclusions

When we analyse the time table, we can conclude that:

- When processing files of smaller dimension, it is not efficient to use several threads, as it can even lead to losing performance, as we can see with the result of using 8 worker threads in the smaller file.

- The bigger the file is, the more efficient the use of multiple worker threads become, gaining a considerable amount of performance time.

- The results obtained were to be expected, not having observed any values that deviate from the expected standard.