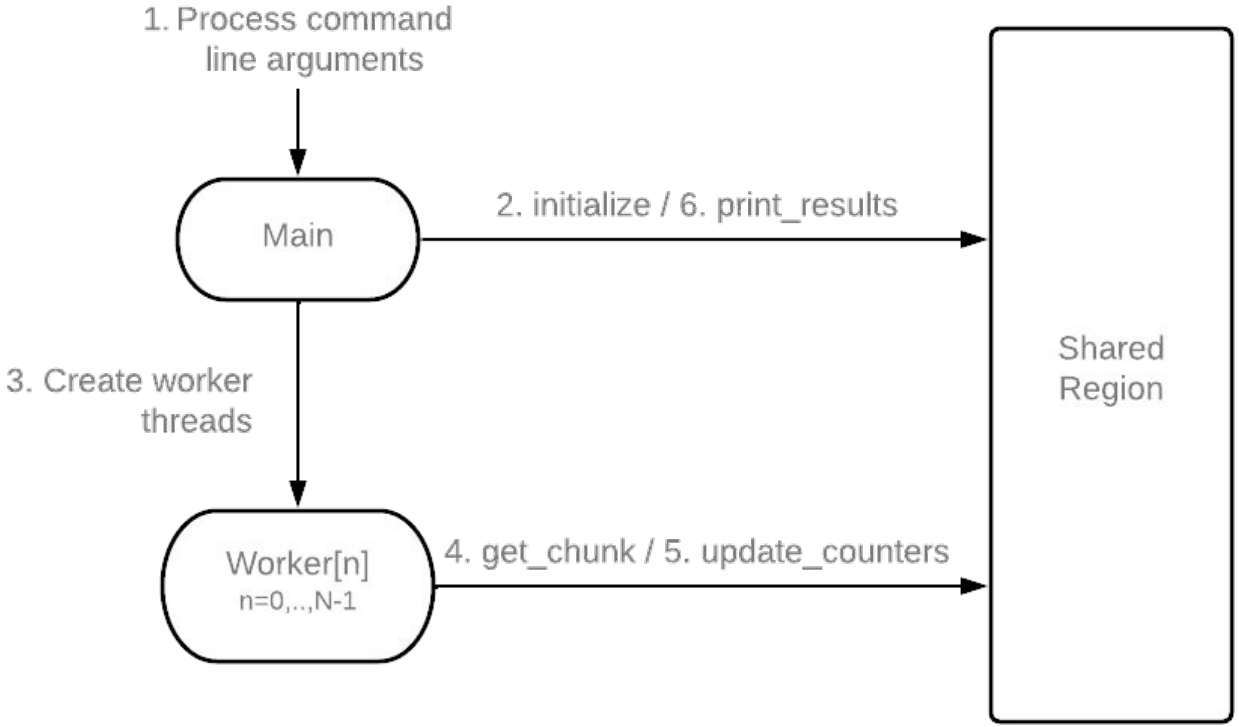
deti universidade de aveiro
departamento de electrónica, telecomunicações e informática

Multithreaded Applications

Computação em Larga Escala
Assignment 1

Artur Romão, 98470
João Reis, 98474

Text Processing Problem - Implementation



Text Processing Problem - Implementation

- We created **2 structures** to help us solving the problem.

```
struct File {
  char *file_name;
  FILE *file;
  int nWords;
  int nWordsA;
  int nWordsE;
  int nWordsI;
  int nWordsO;
  int nWordsU;
  int nWordsY;
};
```

```
struct ChunkData {
  int index;
  bool is_finished;
  unsigned int *chunk;
  int nWords;
  int nWordsA;
  int nWordsE;
  int nWordsI;
  int nWordsO;
  int nWordsU;
  int nWordsY;
};
```

- We use monitors and mutual exclusion when a worker needs to:
  - get a text chunk;
  - update the final counters.

```
static void *worker (void *worker_id) {
  unsigned int id = *((unsigned int *)worker_id); // worker id

  // structure that has file's chunk to process and the results of that processing
  struct ChunkData *chunk_data = (struct ChunkData *)malloc(sizeof(struct ChunkData));
  chunk_data->chunk = (unsigned int *)malloc(maxBytesPerChunk * sizeof(unsigned int));

  while (true) {
    // get a valid text chunk
    get_chunk(id, chunk_data);

    // get result of the chunk processing
    process_chunk(id, chunk_data);

    // update counters
    update_counters(id, chunk_data);

    // reset struct variables
    reset_struct(chunk_data);

    if (all_work_done) break;
  }

  workers_status[id] = EXIT_SUCCESS;

  free(chunk_data); // deallocate the structure memory
  pthread_exit(&workers_status[id]);
}
```

worker lifecycle

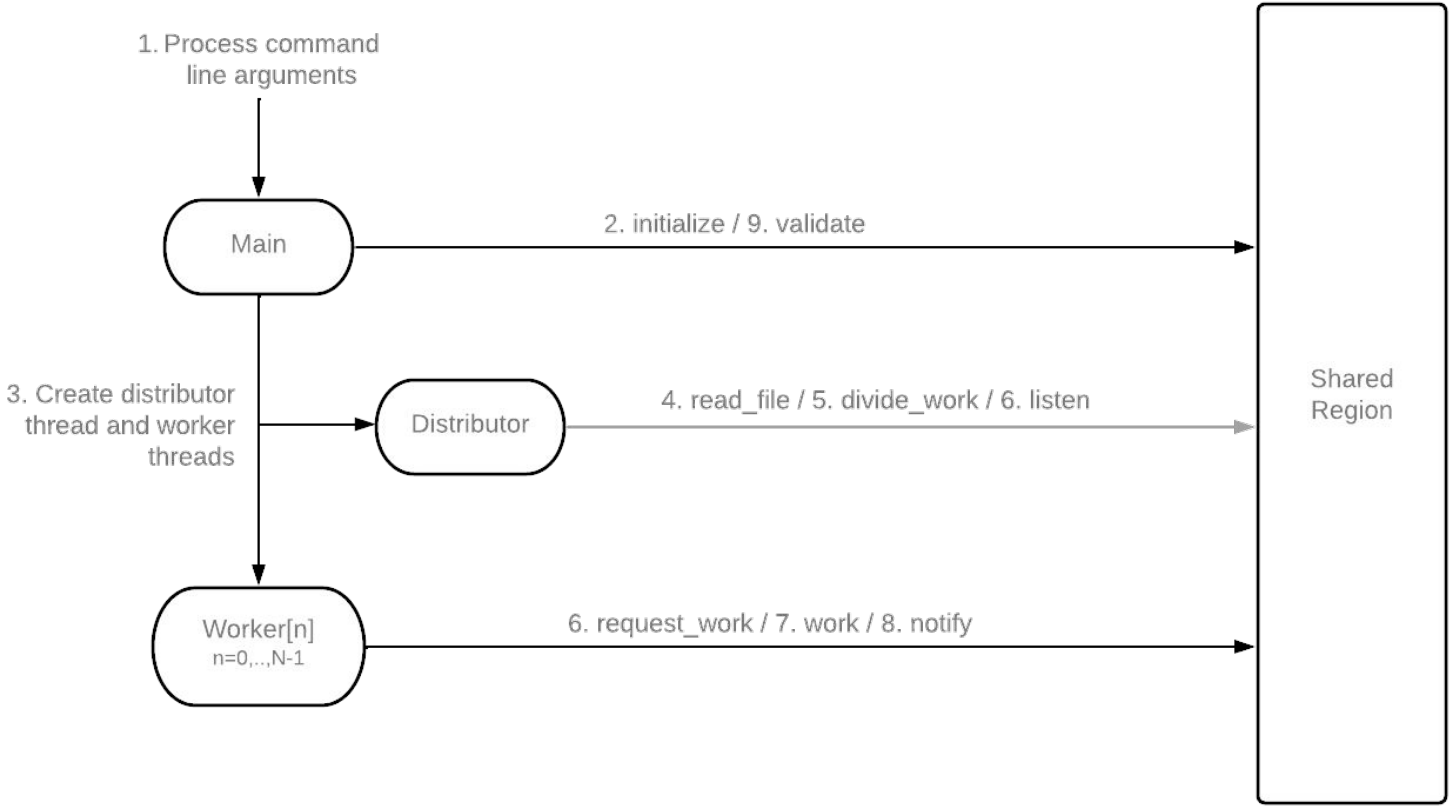
Text Processing Problem - Results

- Timing Results for processing all files and with a chunk size of 4k bytes
- We execute the next command 5 times and we calculate the results that are in the table

./prog1 -f dataset/text0.txt -f dataset/text1.txt -f dataset/text2.txt -f dataset/text3.txt -f dataset/text4.txt

Number of workers	Mean Execution Time	Standard deviation (σ)	Variation (σ2)
1	0.099819	0.014224	0.000202
2	0.073090	0.009678	0.000093
4 (default)	0.092923	0.032570	0.001061
8	0.072264	0.010129	0.000102

Sort Integers Problem - Implementation



Sort Integers Problem - Implementation

- We created **3 structures** to help us solving the problem.

```
struct SubSequence {
  unsigned int *subsequence;
  unsigned int size;
  bool is_sorted;
  bool is_being_processed;
};
```

```
struct File {
  char *filename;
  FILE *file;
  int size;
  unsigned int *sequence;
  struct SubSequence **all_subsequences;
  int all_subsequences_length;
};
```

```
struct Task {
    int worker_id;
    char *type;
    int index_sequence1;
    int index_sequence2;
    bool is_busy;
};
```

Sort Integers Problem - Implementation

```c
static void *worker (void *worker_id) {
  unsigned int id = *((unsigned int *)worker_id); // worker id
  // // printf(">> Starting worker %d thread\n", id);
  bool requested = false;   // flag para não estar sempre a fazer pedidos de request (apenas faz um pedido e espera)

  while(true) {

    if (!(tasks + id)->is_busy) {
      // send a request to distributor and wait for a work assignment
      if (!requested) {
        request_work(id);
        requested = true;
      }

    } else {

      if ( strcmp((tasks + id)->type, "sort") == 0 ) {
        // get work and sort integers
        sort_sequence(id);

      } else if ( strcmp((tasks + id)->type, "merge") == 0 ) {
        // get work and merge subsequences
        merge_sequences(id);
      }

      // send notification to distributor that the work that has been assigned is completed
      notify(id);

      requested = false;
      (tasks + id)->is_busy = false;
    }
    if (all_work_done) break;
  }
  workers_status[id] = EXIT_SUCCESS;
  pthread_exit(&workers_status[id]);
}
```

workers lifecycle

```c
static void *distribute (void *distributor_id) {
  unsigned int id = *((unsigned int *)distributor_id); // worker id
  // printf(">> Starting distributor thread\n");

  read_file();

  divide_work(n_workers);

  // esperar que um worker peça trabalho
  listen(id, n_workers);

  distributor_status = EXIT_SUCCESS;
  pthread_exit(&distributor_status);
}
```

distributor lifecycle

Sort Integers Problem - Results

- Timing Results for datSeq1M.bin.
- We execute the next command 5 times and we calculate the results that are in the table.

./prog2 dataset/datSeq1M.bin

Number of workers	Mean Execution Time	Standard deviation (σ)	Variation (σ2)
1	0.936073	0.019091	0.000364
2	0.943078	0.039129	0.001531
4 (default)	1.008060	0.019225	0.000369
8	1.401512	0.056005	0.003136

Conclusion

After reviewing the achieved results, we came to the following conclusions:

- All worker threads are synchronized, no race conditions have been compromised.
- The use of Structs helped us a lot implementing the multithread application.
- Oddly, for the second program, the execution time increases with the increased number of worker threads.