



deti

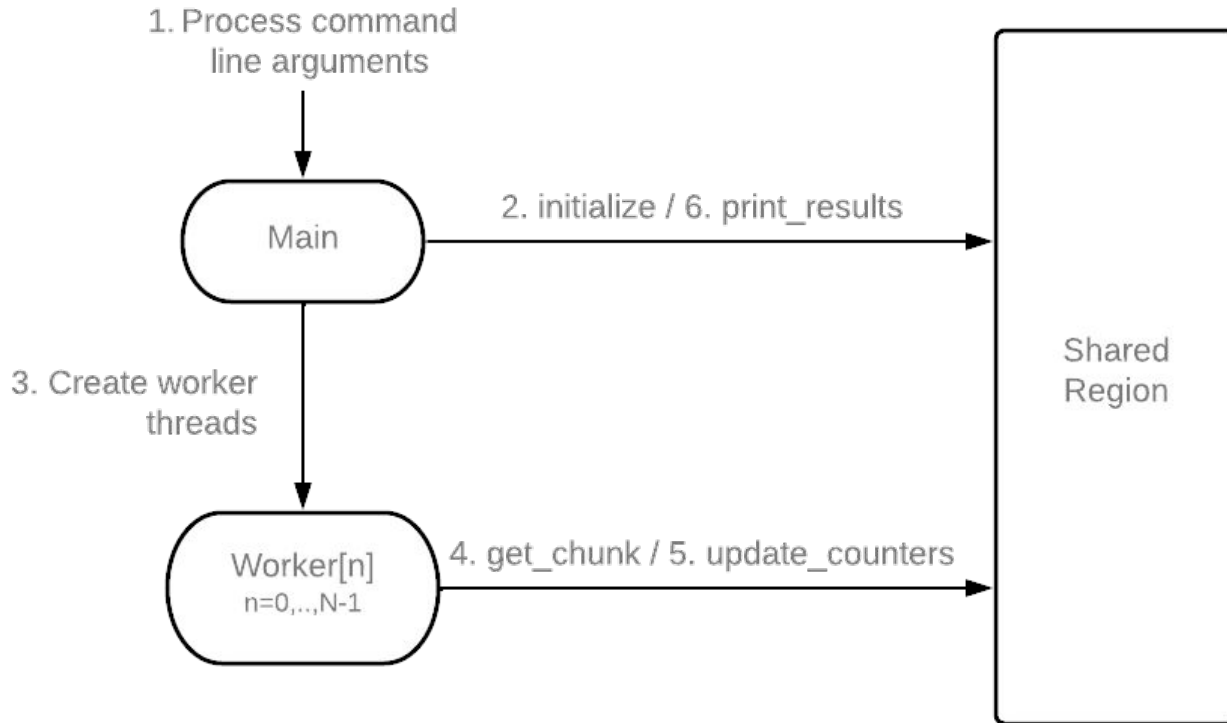
universidade de aveiro  
departamento de electrónica,  
telecomunicações e informática

# 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

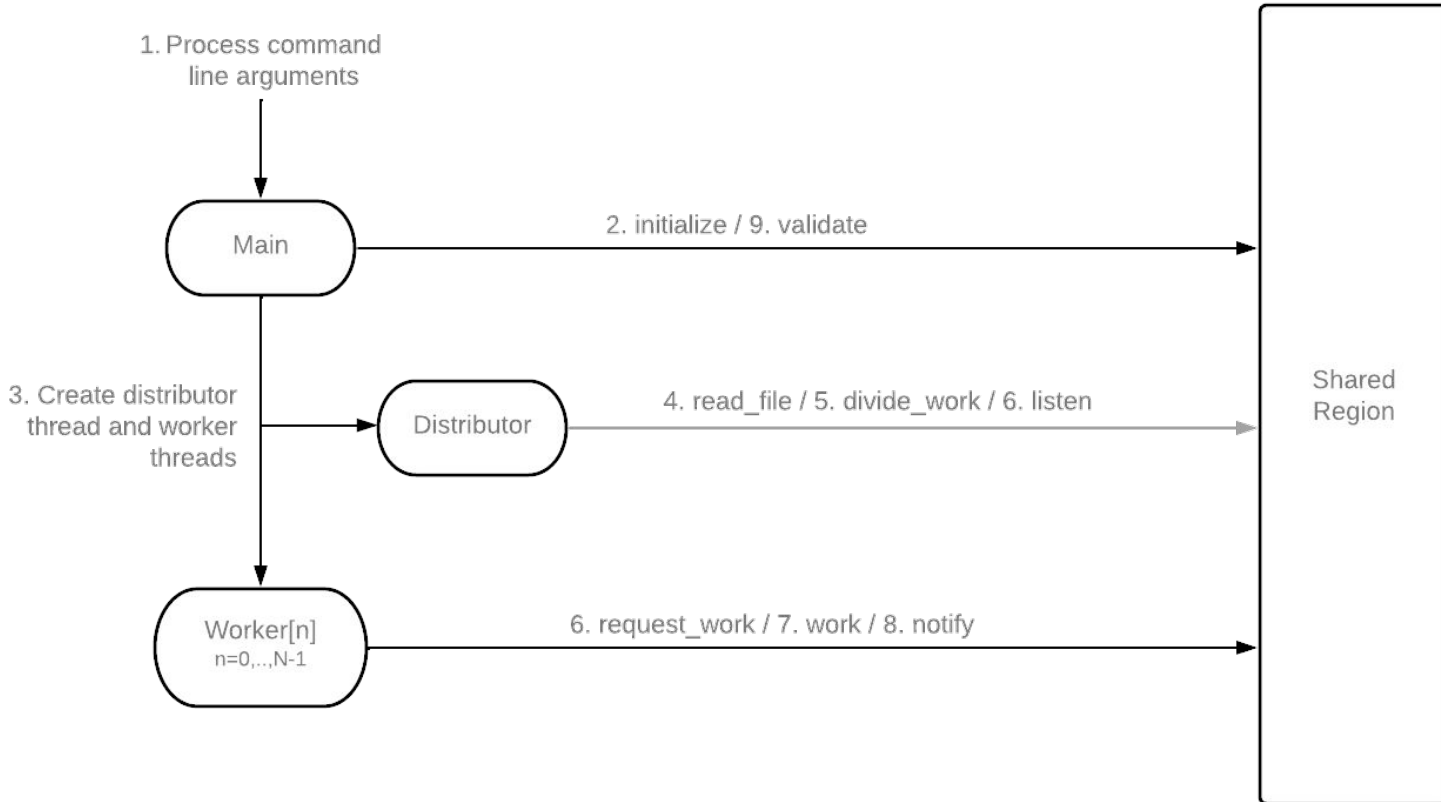
# Text Processing Problem - Results

- Timing Results for processing all files and with a chunk size of 4k bytes

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

Number of workers	Execution Time
1	0.088293s
2	0.071563s
4 (default)	0.059965s
8	0.067840s

# 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

```
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

```
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 processing datSeq256K.bin and datSeq16M.bin

./prog2 dataset/datSeq256K.bin

Number of workers	Execution Time
1	0.206026s
2	0.185438s
4 (default)	0.193403s
8	0.246061s

./prog2 dataset/datSeq16M.bin

Number of workers	Execution Time
1	19.723633s
2	18.968134s
4 (default)	20.360376s
8	25.842932s



# 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.