


# *Computação em Larga Escala*

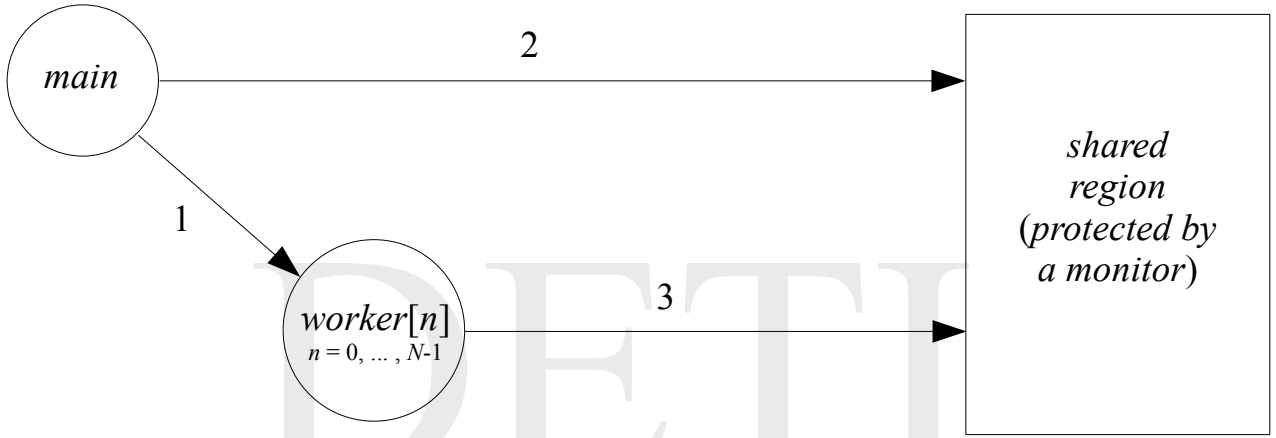
*Assignment 1 – Algorithmic analysis*

António Rui Borges

# *Summary*

- *Text processing in Portuguese*
  - *Decomposition*
  - *Roles assigned to the different intervening entities*
  - *Characterization of the shared region*
- *Sorting a sequence of integers*
  - *Decomposition*
  - *Roles assigned to the different intervening entities*
  - *Characterization of the shared region*

# *Text processing in Portuguese - 1*






**Operations**

1 - create the *worker* threads, wait for them to terminate

2 - store the list of the names of the text files that are to be processed, print the results of the processing

3 - get a chunk of text and process it, save partial results

# *Text processing in Portuguese - 2*

Role of the *main thread*

- to get the text file names by processing the command line and storing them in the shared region
- to create the *worker* threads and wait for their termination
- to print the results of the processing.

Role of the *worker threads*

- while there is work to be carried out
  - to request chunks of text of some text file
  - to process them
  - to return the partial results.

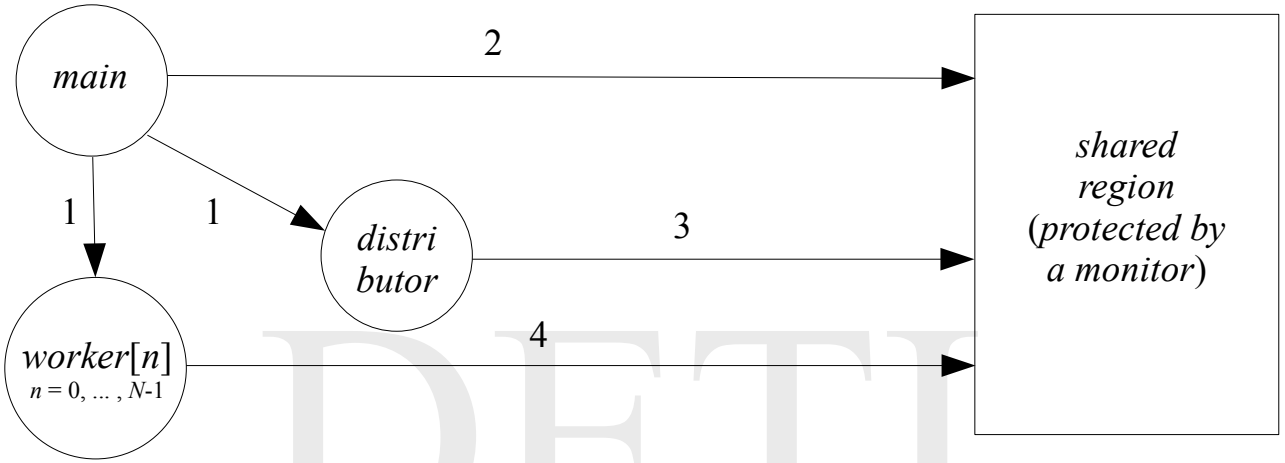
# *Text processing in Portuguese - 3*

All operations carried out in the shared region should be performed in mutual exclusion.

Internal data structures should be provided so that

- the text files are to be processed in succession
- the *worker* threads, when they request a chunk of text, need not to know the text file it belongs to
- the chunks of text should only contain portions with complete words
- the chunks of text should have a size of approximately 4K or 8K bytes.

# *Sorting a sequence of integers - 1*






**Operations**

1 - create the *distributor* and the *worker* threads, wait for them to terminate

2 - store the name of the binary file 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

# *Sorting a sequence of integers - 2*

Role of the *main thread*

- to get the binary file name by processing the command line and storing it in the shared region
- to create the distributor and the *worker* threads and wait for their termination
- to check if the sequence is properly sorted.

Role of the *distributor thread*

- to read the sequence of integers from the binary file
- to distribute sub-sequences of it to the *worker* threads.

Role of the *worker threads*

- while there is work to be carried out
  - to request a sub-sequence of the sequence of integers
  - to sort it
  - to let the *distributor* thread know the work is done.

# *Sorting a sequence of integers - 3*

All operations carried out in the shared region should be performed in mutual exclusion.

There is two synchronization points for the *distributor* thread: in the first, it waits for work requests by the *worker* threads; in the second, it waits for a notification that the work that was previously assigned is completed.

There is one synchronization point for the *worker* threads, where they wait for a work assignment.

Internal data structures should be provided so that

- the sorting procedure can be carried out is several iterations steps whose granularity depends on the number of worker threads that were created.

# *Routine for time measurement*

```
/**
 *  \brief Get the process time that has elapsed since last call of this time.
 *
 *  \return process elapsed time
 */

static double get_delta_time(void)
{
  static struct timespec t0, t1;

  t0 = t1;
  if (clock_gettime (CLOCK_MONOTONIC, &t1) != 0)
     { perror ("clock_gettime");
       exit(1);
     }
  return (double) (t1.tv_sec - t0.tv_sec) +
                  1.0e-9 * (double) (t1.tv_nsec - t0.tv_nsec);
}
```