Report on exercise #4

Matteo Corain S256654

Laboratory #4 – System and device programming – A.Y. 2018-19

The solution for this exercise is proposed, with limited modifications, both for the sequential and the concurrent versions of the program. In both cases, the main function performs the following actions:

- Checks the number of parameters passed via the command line and parses an integer value from the first one (via the `atoi()` function), storing it in the global variable `k`;
- Dynamically allocates vectors `v`, `v1` and `v2` and the matrix `mat` via `malloc()` operations (each containing elements of type `double`), checking the correctness of the operation; in the case of the concurrent program, a vector of thread identifiers is also allocated (type `pthread_t`), storing the TIDs of the created threads in order to be possible to join them afterwards;
- Fills the vectors and the matrix with random values in the interval [-0.5, 0.5]; this is obtained by calling the `rand()` function (whose random seed has been previously initialized via `srand()` to the current time), casting its return value to `double`, dividing it by `RAND_MAX` (so that a real number in [0, 1] is obtained) and finally subtracting 0.5 to the result to shift the interval;
- Prints the vectors and the matrix, by using the `print_vec()` and `print_mat()` functions;
- Performs the product operations, as described below;
- Frees the allocated dynamic memory by means of `free()` operations.

In order to perform the product operations, both the solutions make use of a function, called `scalar_product()`, which, given two vectors of doubles `v1` and `v2` with length `n`, returns the result of the operation $v1 * v2^T$. This is obtained by iteratively multiplying the corresponding elements of the two vectors, cumulating the results in the variable `res` (of type `double` as well) which is finally returned.

In the sequential version of the program, this function is iteratively called on the different rows of the matrix `mat` and the vector `v2`, in order to produce, one by one, the components of the result vector `v`. After the vector `v` has been completely populated, the program uses the function again to compute the result of the product between `v1` and `v`, then it prints the result.

In the concurrent version of the program, calls to `scalar_product()` are wrapped in the `scalar_prod_thr()` function, which is executed by every thread created and later joined by the main. This function receives, by value, an index casted to a pointer, which identifies the row of the matrix to be considered for the operation; the type of this index is set to `long`, in order to avoid problems of casting when compiling the program on a machine running a 64 bits operating system (using 8 bytes pointers instead of 4 bytes ones). After the execution of the product operation, additionally, each thread enters in a critical section protected by a `pthread_mutex_t` called `mutex`, initialized by the main thread, in which:

- It decrements the value of the global variable `left`, which counts the number of threads which have not yet finished their operations, initialized by the main thread to the value of `k`;
- It checks whether the value of `left` is zero (all threads have terminated their operations) and, in that case, it performs the scalar product between `v1` and `v` and prints the result.