System and Device Programming

Lab 04 Exercises – barriers, memory map and threads

Exercise 1

Write a C program using Pthreads to sort the content of a **binary** file including a sequence of random integer numbers, passed as an argument of the command line.
Map the file as a vector in memory.
Implement a threaded quicksort program where the recursive calls to **`quicksort`** are replaced by threads activations, i.e. sorting is done, in parallel, in different regions of the file.
If the difference between the **`right`** and **`left`** indexes is less than a value **`size`**, given as an argument of the command line, sorting is performed by the standard **`quicksort`** algorithm.

This is a sequential recursive implementation of the quicksort algorithm.

```
void quicksort (int v[], int left, int right) {
  int i, j, x, tmp;
  if (left >= right)   return;
  x = v[left];
  i = left - 1;
  j = right + 1;
  while (i < j) {
    while (v[--j] > x);
    while (v[++i] < x);
    if (i < j)
       swap (i,j);
  }
  quicksort (v, left, j);
  quicksort (v, j + 1, right);
}

void swap(int i, int j){
  int tmp;

  tmp = v[i];
  v[i] = v[j];
  v[j] = tmp;
}
```

Exercise 2

Implement a sequential program in C that takes a single argument **`k`** from the command line. The program creates two vectors (**`v1`** and **`v2`**) of dimension **`k`**, and a matrix (**`mat`**) of dimension **`k`**x**`k`**, which are filled with random numbers in the range **`[-0.5 0.5]`**, then it performs the product **$v1^T$ * mat * v2**, and print the result. This is an example for **`k=5`**:

```
v1^T = [-0.0613   -0.1184   0.2655    0.2952   -0.3131]
mat=[  -0.3424   -0.3581    0.1557    0.2577    0.2060
        0.4706   -0.0782   -0.4643    0.2431   -0.4682
        0.4572    0.4157    0.3491   -0.1078   -0.2231
       -0.0146    0.2922    0.4340    0.1555   -0.4538
        0.3003    0.4595    0.1787   -0.3288   -0.4029]
v2^T = [-0.3235   0.1948   -0.1829    0.4502   -0.4656]

Result: 0.0194
```

Perform the product operation in two steps: **`v = mat * v2`**, which produces a new vector **`v`**, and **`result = v1`**$^{\mathbf{T}}$ **`* v`**

Then, write a concurrent program using threads that performs the same task. The main thread creates the vectors, the matrix, and **`k`** threads. Then, it waits the termination of the other threads.

Each thread **`i`** performs the product of the **`i-th`** row vector of **`mat`** and **`v2,`** which produces the **`i-th`** element of vector **`v`**.

One of the created threads, the **last** one terminating its product operation, performs the final operation **`result= v1`**$^{\mathbf{T}}$ **`* v`**, and prints the result.