Threads POSIX **SCOMP-PL5**

1. Consider the following code:

```
1  int main() {
2      pid_t pid = fork();
3
4      if(pid == 0) {
5         fork();
6         pthread_t thread_id;
7         pthread_create(&thread_id, NULL, thread_func, NULL);
8         pthread_join(thread_id, NULL);
9      }
10     fork();
11     …
12 }
```

   Excluding the main process and thread:
   a) How many processes are created? Justify your answer.
   b) How many threads are created? Justify your answer.

2. Consider an array, in which each array element is a structure. The fields of such structure are: number, name and grade. The array has a capacity for 5 elements. Implement a function to receive as parameter one element of the array and print it. The function must print all the fields of the structure. Assumptions:
   - Create 5 threads;
   - Each thread must start in the created function;
   - Each thread will only deal with an element of the array.

3. Implement a program that given a number, searches that number in an array of integers (filled up with non-duplicated values). The size of the array is 1000. Assumptions:
   - The search must be carried out by 10 threads;
   - Each thread searches 100 positions of the array;
   - The thread that finds the requested number must:
     - Print out the respective array position;
     - Return as "exit code" a pointer to the thread number (1, 2, 3, 4, 5, 6, 7, 8, 9 or 10);
   - The other threads, that do not find such a number, must return as "exit code" a NULL pointer;
   - The main thread must wait for all created threads and print out the thread number that found such number (returned as "exit code").

4. Implement a program to multiply two matrices. Assumptions:
   - The size of each matrix is 8x8;
   - Create two threads to fill the two matrices;
   - Create 8 more threads for solving the problem (to multiply the matrices);
   - The main thread must wait for all threads to calculate and should print the calculated matrix.

5. Implement a program to provide statistics of the balance of all customers of a bank. Consider that the balance values of all customers are in an array. The size of the array is 1000. Assumptions:
   - The program should create 3 threads;
   - The first thread will search for the lowest balance and write it to a global variable;
   - The second thread should search for the highest balance and write it to a global variable;
   - The third thread should calculate the average balance and write it to a global variable;
   - The main thread, using the global variables, should print out the results written by the created threads.

6. Implement a program that outputs prime numbers. Your program should ask the user for an unsigned integer number. The program will then create another thread that outputs all the prime numbers that are less than or equal to the number entered by the user.

7. "Totoloto" is a lottery game. Actually, it encompasses two separate lotteries:
   - First is a pick-5 from 49 numbers (for our purpose only this game will be relevant);
   - Second is a pick-1 from 13 numbers.

   To play the game, the player must choose 5 numbers from 1 to 49 and 1 number from 1 to 13 by marking the numbered squares on a play slip. Then take the play slip to a lottery retailer (or agent). The retailer enters the selection in the on-line terminal, which produces a game ticket.

   Implement a program that outputs the statistical values related to the first lottery (1 to 49). That is, it should print out the number of times each number was drawn.

   Consider a database composed by 8000 "keys" (a "key" is a set of 5 numbers drawn on each lottery game session).

   Assumptions:
   - The program should create 16 threads;
   - Each thread performs a partial accounting of 500 “keys” (8000/16);
   - The main thread should print out the statistical values.

   Suggestion: Use an array of mutexes to ensure the data coherence.

8. Implement a program to perform a set of calculations. Consider an array of random integers called `data` and another array called `result` were results of the calculations must be saved in. The size of each array is 1000. Assumptions:
   - The program should create 5 threads;
   - Each thread performs a partial calculation of 300 values (1000/5);
     - `result[i] = data[i] * 10 + 2`;
     - These calculations must be performed in parallel;
   - Each thread prints out its partial calculations in the correct order. That is, the first thread must print out values from `result[0]` to `result[199]`, the second thread must print out values from `result[200]` to `result[399]` and so on.

9. Implement a railway simulator. The railway network’s infrastructure is composed by four city train stations ("City A", "City B", "City C" and "City D") and the connections are as showed in Figure 1.

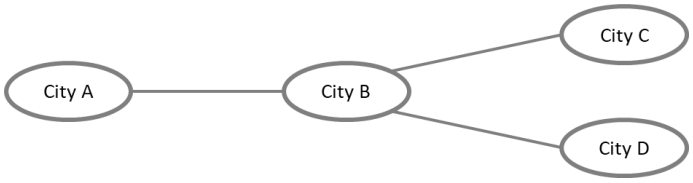

Figure 1: The railway network's infrastructure

   Assumptions:
   - Use threads to simulate the trains;
   - The trains are numbered;
   - The duration of each trip could be simulated using the sleep function (`pthread_sleep`);
   - Each connection is a single-track. So, only one train can use it at a given time instant;
   - Whenever a train use a track, it must print out its number, origin and destination;
   - Whenever a train finishes its trip it should print out the trip’s duration.

10. A retail market study has a set of data related to 5 products sold in 3 hypermarkets. Such data is stored in an array, called `Vec`. The size of the array is 10000. Each element of the array is a triple `{id_h, id_p, p}`, where:
    - `id_h`: hypermarket identifier;
    - `id_p`: product identifier;
    - `p`: product price.

    Implement a program to compute which is the hypermarket where the sum of price of 5 products is the lowest. The program should create 6 threads (T1, T2, T3, T4, T5 and T6). The computation is split in 3 parts: *filtering*, *computing* and *presenting*.
    In the *filtering* part are used 3 threads (T1, T2 and T3). These threads read data from `Vec` and write it in a specific array according to the hypermarket identifier (`id_h`). That is, all data related to hypermarket 1, is stored in an array called `Vec1`, all data related to hypermarket 2, is stored in an array called `Vec2` and, finally, all data related to hypermarket 3, is stored in an array called `Vec3` (see Figure 2). When these three threads finish they work they signalize such event.
    The *computing* part starts when filtering threads (T1, T2 and T3) finish their work. When they signalized the end of filtering part, threads T4, T5 and T6 start executing. Each thread is dedicated to only one array's hypermarket (T4 uses data from Vec1, T5 uses data from Vec2 and T6 uses data from Vec3). Each thread should compute the average price of each product. Then, it must sum the average price of 5 products. This is the cost of the 5 products per hypermarket. Next, the lowest cost value and the respective hypermarket identifier must be stored in a global variable.
    The *presenting* part is performed by the Main thread that prints out the cost value and hypermarket identifier. Note that, the Main thread does not perform any computation, it only prints out information.

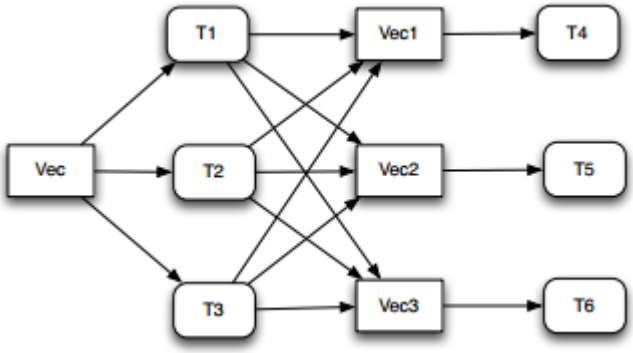

Figure 2: Program architectural structure

11. Implement a program to compute the results of the SCOMP assessment exam. The assessment exam is composed by 3 groups of questions. Each group is classified in the range 0 to 100. The final SCOMP mark (notaFinal) is computed as follows:

$$notaFinal = (notaG1 + notaG2 + notaG3) / 3$$

Where notaG1, notaG2 and notaG3 are the classification of the first, second and third groups, respectively. Consider the following data structure to store student's assessment exam data:

```
typedef struct {
    int number; //student number
    int notaG1;
    int notaG2;
    int notaG3;
    int notaFinal;
}Prova;
```

The program must create 5 threads (T1,T2,T3,T4 and T5). Thread T1 generates 300 Provas. Whenever it inserts one Prova into array ArrayProva, it signalizes threads T2 and T3 about that event. Therefore, one of such threads perform the calculation of notaFinal field and if notaFinal value:

- is higher or equal to 50, it signalizes thread T4 that increments the pos variable (that counts the number of positive exams).
- is smaller than 50, it signalizes thread T5 that increments the neg variable (that counts the number of negative exams).

In the end, the Main thread prints out the percentage of positive and negative exams. Figure 3 shows the program architectural structure.

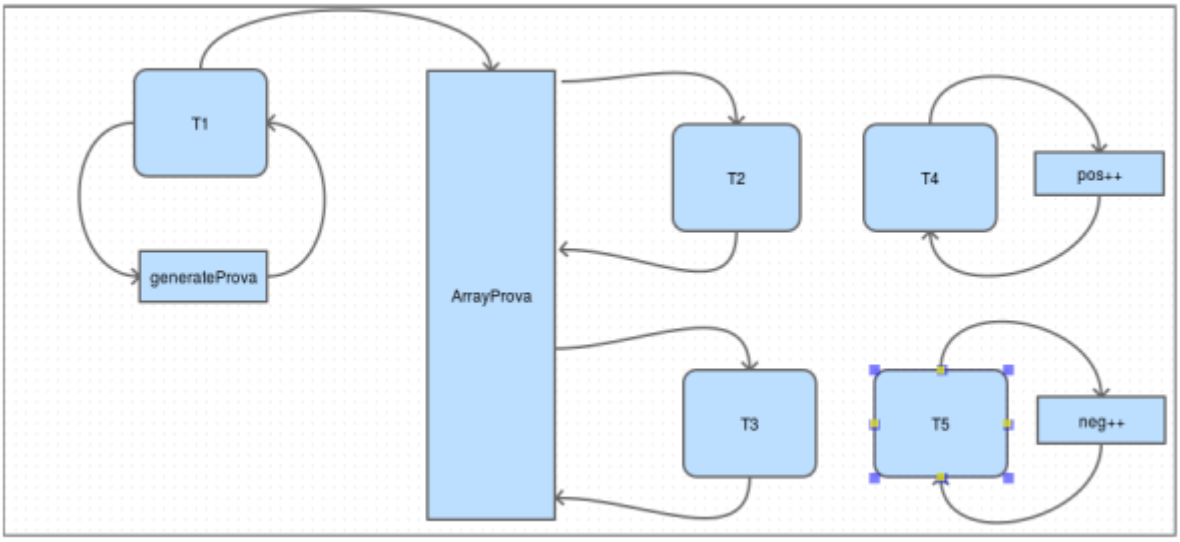

Figure 3: Program architectural structure