Threads POSIX

Orlando Sousa, António Barros, Luis Lino Ferreira, Luís Miguel Pinho,
André Andrade, Carlos Gonçalves, Jorge Pinto Leite, Nuno Morgado,
David Freitas, Armando Nobre

April 2023

1. Consider the following code:

```c
int main() {
    pid_t pid = fork();

    if(pid == 0) {
        fork();
        pthread_t thread_id;
        pthread_create(&thread_id, NULL, thread_func, NULL);
        pthread_join(thread_id, NULL);
    }
    fork();
    ...
```

   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 which stores up to 5 elements of a given structure type. The structure stores data for number, name and grade. Implement a function that receives as parameter one element of the array and prints all its fields. The program should:

   - Create 5 threads;
   - For each thread, the implemented function should be executed;
   - On each thread only one array element is its argument.

3. Implement a program that for a given a number, an array filled with non-duplicated values is used to find the number. The array stores 1000 values. The program should:

   - Carry out the search using 10 threads;
   - For each thread only 100 positions of the array are searched;
   - The thread that finds the requested number should:
     - 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);
   - Threads that do not find the 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 the number (returned as "exit code").

4. Implement a program to multiplies two matrices. The program should:

   - Use two matrices, each with 8x8 dimensions;
   - Create two (2) threads to fill the two matrices;
   - Create eight (8) more threads to solve the problem (perform the matrices multiplication);

- The main thread should wait for all threads to end and then print out the calculated matrix, which should be stored on a global variable.

5. Implement a program that generates statistics of all bank customers balances. All balance values are stored in an array of 1000 values. The program should:
    - Create 3 threads;
    - The first thread searches for the lowest balance and stores it in a global variable;
    - The second thread searches for the highest balance and stores it in a global variable;
    - The third thread computes the average balance and stores it in a global variable;
    - The main thread, using the global variables, prints out all the results stored by each threads.

6. Implement a program that outputs prime numbers. It should begin by asking the user for a highest positive value. Then a created thread will outputs all the prime numbers that are less than or equal to the number entered by the user.

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

    To play the game, the gambler must choose 5 numbers from 1 to 49 and 1 number from 1 to 13 by marking the numbered squares on a bet slip. Then the bet slip is registered by a lottery retailer (or agent) that enters the selection in the on-line terminal, to produce a game ticket.

    A draw will occur in which a "key" is drawn. This "key" is composed by 5 numbers from 1 to 49 as well as 1 number from 1 to 13.

    Implement a program that outputs statistical information concerning the first lottery (pick-5 from 49 numbers). The expected result is the amount of times each one of the 49 number were drawn.

    Consider a database composed by 8000 "keys", being each "key" a set of 5 numbers concerning the first lottery.

    The program should:
    - Create 16 threads;
    - For each thread perform a partial account of 500 "keys" (8000/16);
    - In the main thread 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 the **data** array with random integers and the **result** array were results of the calculations will be stored. Each array has a size of 1000. The program should:
    - Create 5 threads;
    - For each thread perform a partial calculation of 200 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, but only after all values are calculated. That is, the first thread should 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.

    The program should:
    - Use threads to simulate the trains;
    - Number each train;

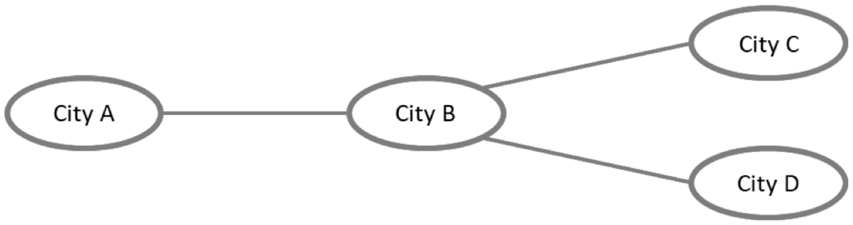

Figure 1: The railway network's infrastructure

- Simulate each trip using sleep function (pthread_sleep);
- Guarantee each connection is a single-track. So, only one train can use it at a given time instant;
- Print the train number, origin an destination when a track is being used;
- Print the trip's duration when the train finishes it .

10. A retail market study collected a set of data related to 5 products sold in 3 hypermarkets. The data collected is stored in an array, called **Vec** with a size of 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 hypermarket has the lowest sum for the 5 products. The program should:

    - Create 6 threads (T1, T2, T3, T4, T5 and T6);
    - Split the computation into 3 stages: **filtering**, **computing** and **presentation**;
    - For the **filtering** stage it should:
      - Use 3 threads (T1, T2 and T3);
      - For each thread read data from **Vec** and write on a specific array associated with the hypermarket identifier (id_h);
      - Associate an hypermarket identifier (id_h) with a specific array, i. e., for the first identifier the **Vec1** array should be used, **Vec2** array for the second identifier and **Vec3** array for the third hypermarket identifier (see Figure 2);
      - Signalize that all 3 threads ended the **filtering** stage.
    - For the **computing** stage:
      - It should start after all **filtering** threads (T1, T2, T3) have finished;
      - Upon being signaled the end of the **filtering** stage, threads T4, T5 and T6 start its execution;
      - Each thread processes only the data of one hypermarket (identifier), i.e. T4 for **Vec1**, T5 for **Vec2** and T6 for **Vec3**;
      - To determine the cost of the 5 products, the threads should:
        * Compute the average price of each product;
        * Sum all 5 average prices.
      - A global variable should be updated with the lowest cost as well as the hypermarket identifier.

    The **presentation** stage is performed on the main thread that prints out values for both cost and hypermarket identifier. Note that in this thread no computation is executed as it only presents the information.

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 a scale of 0 to 100. The final SCOMP grade (finalGrade) is computed as follows: $finalGrade = (g1Result + g2Result + g3Result)/3$ where g1Result, g2Result and g3Result are the classification for the first, second and third groups, respectively. Consider the following data structure to store student's assessment exam data:

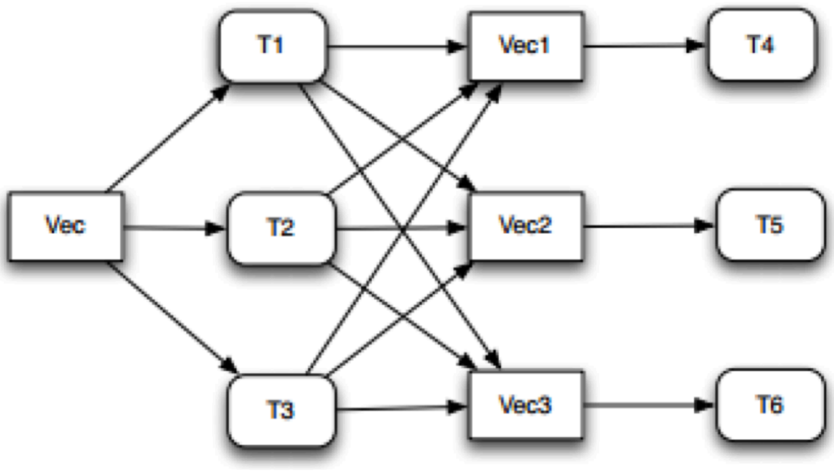

Figure 2: Program architectural structure

```c
typedef struct {
    int number;
    int g1Result;
    int g2Result;
    int g3Result;
    int finalGrade;
}Exam;
```

The program should:

- Create 5 threads (T1, T2, T3, T4 and T5);
- T1 thread generates 300 exams (Provas). Whenever one exam is inserted into ArrayProva array, threads T2 and T3 receive a notification for such an event;
- Upon receiving the notification one of these threads compute the final grade and if value:
  - Is higher or equal to 50, it signalizes thread T4 which increments the pos variable, counting the number of positive grade;
  - is smaller than 50, it signalizes thread T5 that increments the neg variable, counting the number of negative grades.
- In the end, on main thread print out the percentage of positive and negative grades. Figure 3 shows the program architectural structure.

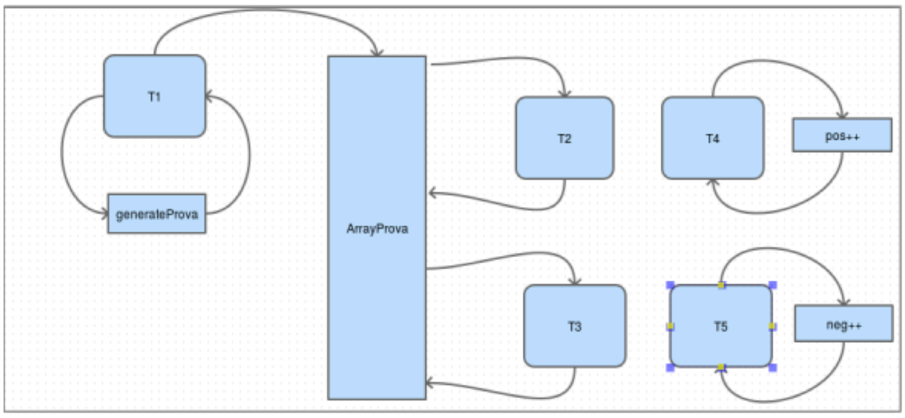

Figure 3: Program architectural structure