PL6 - Threads

Luís Nogueira, Orlando Sousa

{lmn, oms}@isep.ipp.pt

Sistemas de Computadores
2024/2025

Note: All exercises must passive synchronization mechanisms between threads. The use of active (busy) waiting is discouraged.

1. Consider that in the main function, two strings are defined, where the first contains your first name and the second contains your last name. Create a function that takes a string as a parameter and writes that string to the screen. Implement a program where each string is presented by a different thread.

2. Develop a program to output prime numbers. The program operates as follows:
   - Prompt the user to input the highest positive value.
   - Create a thread responsible for outputting all prime numbers less than or equal to the value entered by the user.

3. Consider an array designed to hold up to 5 elements of a specific structure type. This structure includes fields for number, name, and grade data. Your task is to implement a function that accepts one element of the array as a parameter and prints all its fields. The program specifications include:
   - Creation of 5 threads;
   - Execution of the implemented function for each thread;
   - Passing only one array element as an argument to each thread.

4. Develop a program that searches for a given number within an array of integers, where the array is populated with non-duplicate values and has a size of 1000. Here are the assumptions:
   - The search operation is conducted using 5 threads.
   - Each thread is responsible for searching through 200 positions of the array.
   - Upon finding the target number, each thread must:

- Print the respective array position where the number was found.
- Return a pointer to the thread number (1, 2, 3, 4, or 5) as its "exit code".

- Threads that do not locate the target number should return a NULL pointer as their "exit code".
- The main thread is tasked with waiting for all created threads and then printing out the thread number that successfully found the target number (returned as "exit code").

5. Develop a program to generate statistics on the balances of all bank customers. All balance values are stored in an array of 1000 elements. You should create three threads:
   - The first thread is responsible for finding the lowest balance and storing 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 utilizes the global variables to print out all the results obtained by each thread.

6. Develop a program that spawns 5 new threads with the following specifications:
   - Each thread is tasked with writing 200 numbers into a text file.
   - Only one thread should have write access to the file at any given time.
   - Upon completion, the main thread should display the contents of the file.

7. Consider the following two threads:

T1	T2
`buy_chips()`	`buy_beer()`
`eat_and_drink()`	`eat_and_drink()`

   Ensure that neither T1 nor T2 eat or drink until both the beer and chips are acquired.

8. Modify the previous program to:
   - Add 4 more threads that will also execute (randomly) `buy_chips()` or `buy_beer()`;
   - The 6 threads can only execute `eat_and_drink()` when all other threads have finished acquiring the chips and the beer.

   Be careful to produce adequate output that clearly shows the execution of the threads.

9. Develop a program to perform a set of calculations. The program operates as follows:
   - Utilize two arrays: **data** containing random integers and **result** where the calculation results will be stored. Both arrays have a size of 1000.
   - Create 5 threads. Each thread is responsible for performing a partial calculation on 200 values (1000/5):
     - Calculate $result[i] = data[i] \times 10 + 2$; for each value in its assigned range.
     - Perform these calculations in parallel.
   - Once all values are calculated, each thread prints out its partial calculations in the correct order:
     - The first thread prints values from $result[0]$ to $result[199]$.
     - The second thread prints values from $result[200]$ to $result[399]$.
     - And so on.

10. Implement a railway simulator. The railway network's infrastructure comprises four train stations ("Cidade A", "Cidade B", "Cidade C", and "Cidade D") with connections as shown in Figure 1.

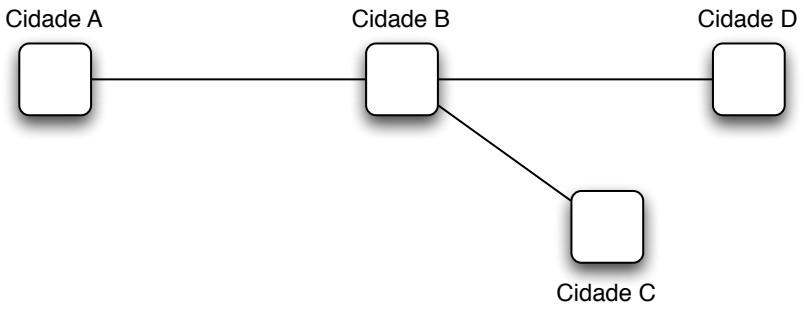

Figure 1: The railway network's infrastructure

Assumptions:
   - Use threads to simulate trains.
   - Each train is assigned a number.
   - The duration of each trip can be simulated using the sleep function.
   - Each connection is single-track, allowing only one train to use it at a time.
   - When a train occupies a track, it must print out its number, origin, and destination.
   - Upon completing a trip, a train should print out the trip's duration.

11. The data concerning the flight path of three airplanes is stored in a *vec* array, where each element has the following information: $\{id, x, y\}$, where id is the identifier of the airplane, and x and y are the coordinates in the 2D plane.

Develop an application where the information contained in the *vec* array is filtered and placed into three arrays *vec*1, *vec*2, and *vec*3. Each of these arrays should contain only the coordinates of airplane 1, airplane 2, and airplane 3, respectively.

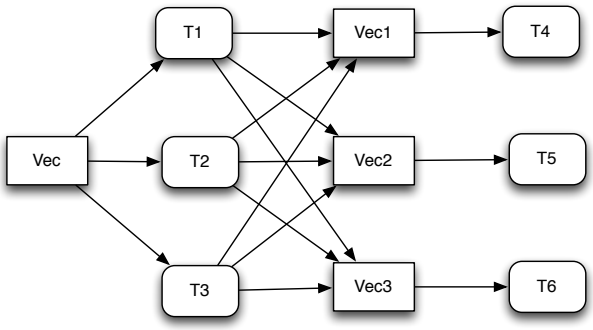

Figure 2: Program architectural structure

For this operation, use three threads (T1, T2, and T3). As soon as threads T1, T2, and T3 finish their work, three threads (one for each vector, T4, T5, and T6) calculate the distance traveled by each airplane and print it on the screen (see Figure 2). To calculate the distance between two points, you can use the following formula:

$$dist = \sqrt{(x1 - x2)^2 - (y1 - y2)^2}$$

Note: all six threads should execute concurrently.

12. Consider that there are 3 snails (`caracol_1`, `caracol_2` and `caracol_3`) that have to race in a previously defined path according to Figure 3.

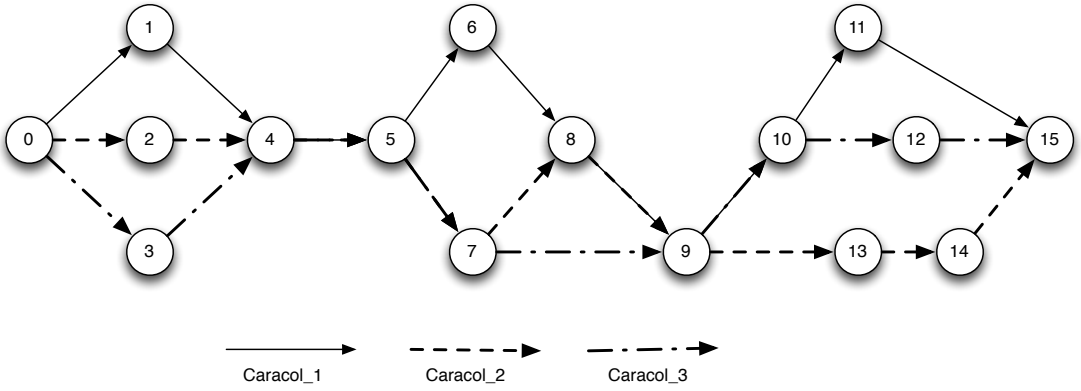

Figure 3: Snail race paths

Each path is composed by several segments (represented by an arrow). Each segment connects two points (represented by a numbered circle). Some segments are dedicated to each path, but there are some segments shared. That is, more than one path use such segments.

The travel time (in seconds) of each segment is defined in Table 1. The starting and finishing points of each segment are in the lines and columns, respectively.

The starting and finishing points interception defines the segment travel time. However, a snail, during the movement, leaves a stringy substance in the path, which double the travel time of the subsequent snail. For instance, the segment between points 4 and 5 has as travel time of 3 seconds. All paths use it, so the first snail to traverse it, will take 3 seconds, the second one 6 seconds and the last one 12 seconds.

Table 1: Segments travel time (seconds)

	0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**
0		5	5	5												
1					4											
2					4											
3					4											
4						3										
5							5	5								
6									6							
7									6	4						
8										4						
9											5			5		
10												5	5			
11															3	
12															3	
13															5	
14																3

Implement a program to simulate a snail race with the follwing assumptions:

- Simulate each snail behaviour with a thread;
- Simulate the travel time with the `sleep` function;
- Each thread must print out its travel path time and stores it in a global variable;
- The main thread prints out the snail race classification.

13. Implement a program to compute the final grade of each student in SCOMP, according to the following fomula:

$$final\_grade = 0.40 * (grade\_S1 + grade\_S2 + grade\_S3)/3 + 0.6 * exam \quad (1)$$

where $grade\_S1$, $grade\_S2$ and $grade\_S3$ are the classification of the first, second and third sprints, respectively, and $exam$ the studen's grade on the exam.

Consider the following data structure to store the student's assessment data:

```
typedef struct {
    int number; //student's number
    float sprint_grades[3];
    float exam;
    float final_grade;
}student_grade;
```

Assume there is an array `student_grade grades[300]` to store the assessment data of all students.

The program must create 5 threads (`T1`,`T2`,`T3`,`T4` and `T5`). Thread T1 randomly generates a grade between 0 and 20 for each component of the grade of each student.

Whenever it generates the partial grades of one student, it signals threads T2 and T3 about that event.

Upon receiving the notification, one of those threads will compute and fill the $final\_grade$ field of that student and if that value:

- Is higher or equal to 10.0, it signalizes thread T4 which increments the *pos* variable, counting the number of positive grades;
- Is smaller than 10.0, it signalizes thread T5 that increments the *neg* variable, counting the number of negative grades.

In the end, the main thread prints the number of positive and negative grades.

14. Develop a program that begins by spawning three new threads. Two of those threads will function as a `producer`, while the other one will serve as a `consumer`.

    These threads interact via a circular buffer, capable of holding 10 integers. Each `producer` inserts incrementing values into the buffer, which the `consumer` subsequently reads and prints.

    It is assumed that only 30 values are exchanged throughout the lifetime of the program. Ensure that:

    - The `producer` thread will be blocked if the buffer reaches its capacity.
    - The `producer` thread will write to the buffer only after ensuring mutual exclusion.
    - The `consumer` threads will be blocked until new data becomes available to read.

15. Develop the following two types of threads:
    - `reader` - responsible for reading a `string` from the heap:
      - Readers refrain from modifying the shared memory area, enabling multiple readers to access it concurrently.
      - Readers are prioritized over writers.
      - Each reader should print the retrieved `string` along with the current count of readers.
    - `writer` - tasked with writing to the shared memory area in the heap:

- Writes its Thread ID (PID) and the current time.
- Only one writer can access the shared memory area at any given time.
- Writers can access the shared memory area only when there are no active readers.
- Each writer prints the count of writers as well as the count of readers.

To verify the effectiveness of the software, execute multiple instances of readers and writers simultaneously.