Shared Memory

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

March 2023

Exercises

1. Implement a solution that sends the number, name, and address of a student between two processes (i.e. two different programs to executed one after the other) not related hierarchically, a writer and a reader.

   - The writer must create a shared memory area, read the data from the keyboard and write them in the shared memory.
   - The reader should read the data from the shared memory and print them on the screen. Note: The reader can only read when there is data. Must use active waiting.

   Note: the writer must be executed before the reader.

2. Implement a program that creates a shared memory area to store two integers and initializes those integers with the values 10000 and 500 and creates a new process. Parent and child must perform the following operations 1.000.000 times:

   - The father will decrease the first value and increase the second value.
   - The child will increase the first value and decrease the second value.

   The father only writes the value on the screen at the end. Review the results. Will these results always be correct? Propose a solution that ensures data consistency.

3. Implement a solution that allows you to share an array of 10 integers between two processes not related hierarchically, a writer and a reader.

   - The writer must create a shared memory area, generate 10 random numbers between 1 and 20 and write them in the shared memory.
   - The reader should read the 10 values, calculate and print the average.

   Note: The reader can only read when there is data. Must use active waiting.

4. Through the use of active waiting adapt the solution from the previous exercise so that it is possible to send and receive five sequences of ten numbers between processes.

5. Implement a program that creates a shared memory area to store an integer, initializes this value to 100, and creates a new process. Parent and child must perform the following operations 1.000.000 times:

   - Increase the value;
   - Decrease the value;

   The father only writes the value on the screen at the end.
   Note: To guarantee data coherence, you must use signals as a synchronization mechanism when accessing shared data.

6. Implement a program to determine the biggest element of an array in a parallel/concurrent environment. The parent process should:

   - Create an array of 1000 integers, initializing it with random values between 0 and 1000;
   - Create a shared memory area to store an array of 10 integers, each containing the local maximum of 100 values;
   - Create 10 new processes;
   - Wait until the 10 child processes finish the search for the local maximum;
   - Determine the global maximum;
   - Eliminate the shared memory area.

   Each child process should:

   - Calculate the largest element in the 100 positions;
   - Write the local maximum found in the corresponding position (0-9) in the array.

7. Implement a program that allows the exchange of data concerning a student between two processes (number, name and grades of a set of classes). The data to be exchanged are represented in the following `struct aluno`.

```
1 #define STR_SIZE 50
2 #define NR_DISC 10
3 struct aluno{
4   int numero;
5   char nome[STR_SIZE];
6   int disciplinas[NR_DISC];
7 };
```

   The parent process should:

   - Create a shared memory area for data exchange. Check the need to add one or more variables to synchronize the writing and reading of data operations;
   - Create two new processes;
   - Fill the shared memory area in accordance with user-entered information;
   - Wait until the two child processes ends.
   - Eliminate the shared memory area.

   The first child process should:

   - Wait for the student data;
   - Calculates the highest and the lowest grade;
   - Print the information on the screen.

   The second child process should:

   - Wait for the student data;
   - Calculates the average grade;
   - Print the information on the screen.

8. Implement a program that optimize the search of words in a set of text files in parallel/concurrent environment. The parent process should:

   - Create an area of shared memory. The memory area must contain, for each child process, the following information:
     - path to the file;

  - word to search;
  - an integer to store the number of occurrences.
- Create 10 new processes;
- Fill the shared memory area with the information for each child process;
- Wait until the child processes finish their search;
- Print the number of occurrences determined by each child;
- Eliminate the shared memory area.

Each child process should:

- Open the text file assigned to it by the parent (can/should be different for each child process);
- Determine the number of occurrences of the word to search;
- Write the number of occurrences in their position in the shared memory area.

9. Implement a program that creates a new process. One will be the producer and the other the consumer. Among them should be created a shared memory area with a circular buffer to store ten integers and the necessary synchronization variables in a shared memory area. The producer puts increasing values in the buffer that should be printed by the consumer. Consider that thirty values are exchanged between them. The consumer can read elements from the buffer if the number of elements is bigger than 0.

10. Solve exercise 9 again, performing synchronization accessing the shared memory region, supported by unnamed pipes. Suggestion: you can use a pipe where you place 10 integers/chars. This pipe will control the producer's access to the shared memory region, which in the case of already having 10 integers in shared memory, will have to wait passively. Use a second pipe to synchronize the consumer's access to the shared memory region. In the case of no values to consume, the consumer process should wait passively.