PL4 - Shared Memory

Luís Nogueira Luís Miguel Pinho António Barros
Paulo Ferreira David Freitas Orlando Sousa
André Andrade Carlos Gonçalves

{lmn,lmp,amb,pdf,dcf,oms,lao,cag}@isep.ipp.pt

Sistemas de Computadores
2023/2024

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

    - The writer must create a shared memory area, read the data from the keyboard, and write them into 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. Active (busy) waiting must be used.

2. Implement a program that creates a shared memory area to store two integers and initializes those integers with the values 10000 and 500, respectively. Then, it creates a new process. The parent and child processes must perform the following operations 1000000 times:

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

    The parent process only writes the final values on the screen.

    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 into the shared memory.
   - The reader should read the 10 values, calculate and print the average.

   Note: Ensure that read operations occur only if there is data available.

4. Using active waiting, modify the solution from the previous exercise to enable the exchange of five sequences of ten numbers between processes.

5. Implement a program that creates an array `int v[1000]` and a shared memory area for an array `int max[10]`. The array `v[]` should be initialized with random values.

   Then, 10 children should be created to determine the greatest value in `v[]` in a concurrent/parallel fashion.

   - Each child should search the greatest value in 1/10 of `v[]` and put it in `max[i]` (where *i* is the child's index)
   - The parent process waits for all children to terminate and calculates the greatest value in `max[]`

   All processes should disconnect and close the shared memory area. The parent should remove the shared memory area before ending.

6. Implement a program that allows the cooperative creation of a message by a set of processes.

   The parent process must:

   - Create a shared memory area to store a string of 100 characters and the necessary synchronization variables;
   - Create *N* new processes;
   - Wait for the *N* child processes to write the complete message;
   - Print the message;
   - Delete the shared memory area.

   Each child process must:

   - Wait for its turn;
   - Ask the user which word to append to the message;
   - Append the word to the current message;

   Example: the parent process creates 5 new processes. The first one writes "We", the second one "like", the third one "SCOMP", the fourth one "very", and the fifth one "much".

   Note: the message should be built in the same string (assume that it has enough space to store the message to be sent to the parent).

7. Implement a program that facilitates the exchange of data related to a student between two processes (containing the student's number, name, and grades for a set of classes). The data to be exchanged are represented in the following student struct.

```
#define STR_SIZE 50
#define NR_COURSES 10

typedef struct{
    int number;
    char name[STR_SIZE];
    int courses[NR_COURSES];
}student_t;
```

   The parent process must:

   - Create a shared memory area for data exchange;
   - Determine the necessity of adding one or more variables to synchronize the writing and reading of data operations;
   - Create two new processes;
   - Populate the shared memory area with user-entered information;
   - Wait until both child processes have finished;
   - Delete the shared memory area.

   The first child process must:

   - Wait for the student data;
   - Calculate and print the highest and lowest grades;

   The second child process should:

   - Wait for the student data;
   - Calculate and print the average grade.

8. Modify the previous exercise to allow the exchange of information from a class, consisting of 20 students, one at a time. Pay attention to the necessary synchronization of reads and writes of the shared data.

9. Implement 3 programs named "Interface", "Encrypt", and "Display" that communicate through a single shared memory area.

   - The "Interface" program prompts the user for a message, writes it to the shared memory area, and waits for it to be read by the "Display" program before prompting the user for the next message.
   - The "Encrypt" program encrypts the message by generating a random integer value between 1 and 5 and incrementing this value to the ASCII code of each character in the message.

- The "Display" program waits for the existence of a new message, decrypts it, writes it to the screen, and deletes it from the shared memory area.

10. Implement a program that creates two new processes: one as the producer and the other as the consumer. Between them, a shared memory area with a circular buffer to store 10 integers and the necessary synchronization variables should be created.

    - The producer inserts increasing values into the buffer, which should be printed by the consumer.
    - The consumer can read elements from the buffer if the number of elements is greater than 0.

    Assume that 30 values are exchanged between them.