Synchronisation with semaphores

Orlando Sousa, António Barros, Luis Lino Ferreira, André Andrade, Carlos Gonsalves, Jorge Pinto Lei

May 10, 2022

Instructions

All exercises must use semaphores for the synchronization between processes.

For all exercises you must present three items:

1. Which pattern are being used in a specific exercise. Consult the slides from the T or TP classes about synchronization.
2. The pseudo-code of the algorithm used on the resolution.
3. The implementation of the algorithm in C. Adequately formatted and commented.

1 Part 1

1. Implement a program that creates 8 new processes and of fulfills all the following requirements.
   1. Each process reads 200 integer numbers from a text file named "Numbers.txt", and writes those numbers to the file named "Output.txt".
   2. At any time only one child should be able to read/write from/into the files (one process operating in each of the files).
   3. At the end, the father process should print the content of the file "Output.txt".
2. Change the last program in a way that every child process reads from the file "Numbers.txt" by order of its process number (from the first to the eighth process). Use 8 semaphores to synchronize between the child processes.
3. Implement a program that behaves as follows:
   (a) Lock a shared memory area.
   (b) Add, to the shared memory, a text line with the following information: "I'm the Father - with PID X", where X should be replaced by the PID of the process.
   (c) Wait N seconds (from 1 to 5 seconds).
   (d) Unlock the shared memory area.

   Test the solution by executing several concurrent instances of your code and make sure that when the limit of 50 strings is reached no more text can be added.

   Note: Consider that you have a shared memory area with 50 strings, each string having 80 characters.
4. Change the last question by adding the following features:
   (a) Add an option to "remove" (free) one string from the shared memory;
   (b) Your program should wait at most 12 seconds to access the shared memory, warning the user if that is not possible.

   To test lunch in parallel several programs from exercise 3.

5. Implement a program that creates a new process and fulfills all the following requirements.
   (a) The child process writes on the screen the message "I'm the child";
   (b) The father process writes on the screen the message "I'm the father";
   (c) The child process should be (always) the first to write the message on screen.

   Use semaphores to synchronize their actions.

6. Change the previous program in order to write 15 messages, alternating between father and child.

7. Consider a program that spawns three children processes. Each child process must behave as follows:

Child process 1	**Child process 2**	**Child process 3**
printf("Sistemas ");	printf("de ");	printf("Computadores -");
printf("a ");	printf("melhor ");	printf("disciplina! ");

   Use the minimum number of semaphores required, so the result printed in the screen is: *"Sistemas de Computadores – a melhor disciplina!"*.

   Note: use *fflush(stdout);* after every *printf()* so the output is not buffered.

8. Implement a program that creates a new process. The father and the child should indefinitely write the letters 'S' and 'C', respectively:

Father process	**Child process**

while TRUE do:
    print "S"
	while TRUE do:
	    print "C"

   Implement a policy based on semaphores such that at any moment the number of 'S' or 'C' differs by at most one. The solution should allow strings such as: "SCCSSCCSCS".

   Note: Use *fflush(stdout);* after every *printf()* so the text is not buffered.

9. Implement a program that creates a new process. Use semaphores to synchronize their actions. Consider the two following processes:

Process 1	**Process 2**
sleep(some random time);	sleep(some random time);
buy_chips();	buy_beer();
eat_and_drink();	eat_and_drink();

   (a) P1 and P2 can only execute *eat_and_drink()*, after both have acquired the chips and the beer.
   (b) Modify the program to add 8 more processes which will also execute (randomly) *buy_chips()* or *buy_beer()*. The 10 processes can only execute *eat_and_drink()* when all other processes finished acquiring the chips and the beer.

2 Part 2

10. Implement a program that creates a new process. Use semaphores to synchronize their actions. A company wants to develop an application which allows to consult and insert personal data records (Number, Name, Address), for that it needs to develop the following 3 programs:

   - **Consult** : This application allows you to consult a user's personal data. To do this, you must ask the user for his or her identification number.
   - **Insert** : Allows you to enter a user's data in the shared memory area.

- **Consult_All** : Lists all data in the shared memory area. You should only display the records that contain valid data.

Note 1: The application should support several *consults* and *inserts* in parallel.

Note 2: Consider that there is a shared memory region that has the capacity to store 100 records.

11. Assume a train has 200 passengers capacity and has three doors. Use semaphores to develop a solution that ensures:

    - the train will never be overloaded;
    - the passengers will not collide with each other when getting on or getting off the train.

      For the solution assume:

      1. Each passenger is a process (e.g. lunch 500 passengers), 250 leaving the train and 250 entering the train).
      2. Each door is also a process that allows one passenger to leave or to enter, at a time. This operation requires e.g. 20 ms.
      3. The output of your code should show the operation of the train.

12. Implement a program that simulates the selling of tickets. You should simulate the customers (several processes), using semaphores to synchronize the access to a shared memory area for data exchange. Consider also the following:

    - Only the seller has access to the tickets, only he knows the number of the next ticket.
    - The clients should be served by their arrival order, blocking until getting their ticket.
    - The client should print the number of the ticket.
    - The behavior of *client* and *seller* should be implemented in separate programs.
    - Assume that each client needs a random number of seconds (1-10) to be served.
    - You should execute several clients concurrently, demonstrating the operation of the algorithm implemented.

13. Implement a program that starts by creating two new processes. Each of the child processes will have the role of producer, while the father will act as a consumer. Between them there is a circular buffer capable of handling 10 integers on a shared memory area. Each of the producers writes increasing values into the buffer and prints them into screen, which should be printed by all consumers. Assume that only 30 values are exchanged and the following behaviour for the producer and consumers:

    1. a producer blocks if the buffer is full;
    2. a producer only writes after guaranteeing mutual exclusion;
    3. a consumer blocks while there is no new data to read.

14. Implement the following two programs:

    (a) **Reader**, reads a string from a shared memory area:

      i. The readers do not change the shared memory area, therefore several readers can access the shared memory area, in parallel.
      ii. Each reader should sleeps for 1s, print the string read from shared memory and the number of readers at the time.
      iii. Readers can only access shared memory when there are no writers.

    (b) **Writer**, writes on the shared memory area:

      i. Writes its PID and current time.
      ii. Writers have priority over readers.
      iii. Only one writer can access the shared memory area at the same time.
      iv. Each writer prints the number of writers and also the number of readers.

Execute several readers and writers at the same time to test your software.

15. Consider a show room facility and a set of visitors, who randomly enter a room. Each visitor is simulated by an infinite loop in which he enters a room and leaves at the end of the show. Each show in a room starts and ends at defined time instants; the visitors can only enter and leave the room at those strict times. Each room supports 5 visitors; once the room is full, further visitors have to wait. For simulation purposes consider that:

    1. you have 25 visitors, 4 exhibition rooms and one lobby;
    2. the duration of each show is 5 seconds;
    3. all shows start and end at the same time, with 1 second delay between the end of a show and the start of the next;
    4. visitors wait in a lobby until they are allowed in randomly in a room.

    Print the necessary message to clearly show the status of the show room. Visitors and rooms are represented by different processes.

16. Consider a bridge with only one lane that permits traffic in both East-West directions. A car can only enter the bridge if the bridge is not occupied by cars going in the opposite way. If that is the case, the car must wait until the bridge allows traffic in the desired direction.

    Assume that a car takes 10 seconds to cross the bridge. The bridge only current allows direction changes whenever the last car, in one direction, finishes its crossing.

    Also assume that cars are processes, randomly created by another program, each of them going in a random direction. The bridge represents the critical section of those processes.

17. Consider a social club with a maximum capacity of 300 clients. There are three types of clients: **VIP**, **Special** and **Normal**. While the full capacity is not reached, clients enter the club according to their arrival order. Whenever the club is full, the entrance of clients is conditioned to the exit of other clients. In this case, priority should be given to waiting VIP, then Special, and only then Normal clients, by this order.

    Print the necessary messages to clearly show the status of the club. The clients show be modeled as processes each of them with its type and the time during which it stays at the club.

18. From an initial vector of 10000 integers randomly chosen in the range 1-5000, it is intended to fill a final vector of 5000 positions with the products of each pair of numbers of the initial vector (n1*n2, n3*n4, . . . , n9999*n10000, n10000) and determine the largest product found.

    To resolve the problem, design a C program with the following requirements:

    (a) Two child processes (P1 and P2) must be created that fill the final vector according to the proposed problem.

    (b) Each child process must process half of the vector, but alternately, i.e., P1 calculates n1*n2 and writes the result in the final vector, then P2 calculates n3*n4 and writes the result in the final vector, and so on until all products have been calculated. Note the sequence should always be P1,P2, P1, P2..., until all values are calculated. Properly synchronize access to vectors through traffic lights.

    (c) Create a third child process (P3) that, running in parallel with the two processes mentioned above, should print a warning message whenever a new larger product is found. This process should not use active standby.

    (d) The final vector and value of the largest product should be printed by the parent process after being sure that P1 and P2 have finished processing.

    Note: In your solution you should take into account the correct use of processes, shared memory and traffic lights, ensuring that the division of work between the processes is able to take advantage of all existing processors on the machine.

19. Unit X has developed a new product, which is intended to be monitored by a set of 5 sensors. Each of the sensors reads 6 values and ends. In each of the measurements, and depending on the value obtained, the sensor can be placed in an alarm state. You were asked to develop a solution that creates 6 processes: 1 "Controller" process and 5 "Sensor" processes.

    The "Controller" process shall be responsible for:

    1. Create and initialize a shared memory region where the measurement results of each sensor will be saved;
    2. Create the semaphores needed to implement the solution;
    3. Print on the screen the result of the execution of each sensor, whenever values are received from the sensors;
    4. Print on the screen the result of changing the number of sensors in alarm, whenever this occurs;
    5. Finish at the end of receiving 6 measurements of each sensor, as well as remove the shared memory region and the created semaphores.

    Each "Sensor" process must:

    1. Generate a random value between 0 and 100, thus simulating the result of a measurement;
    2. Save this value in your respective area of the shared memory region;
    3. Pass a sensor to "alarm" state, thus increasing the number of sensors in this state, if the measurement result exceeds 50 (consider that you should not trigger the same sensor in alarm, in duplicate);
    4. Decrement the number of alarm sensors if the result of the current measurement and the previous measurement is less than 50;
    5. Wait 1 second and repeat the operation again (a total of 6 times);

    The 5 "Sensor" processes must be performed simultaneously, ensuring correct access to the shared memory region through the use of traffic lights and without using active standby. You should also define the shared data structure that you consider most appropriate for troubleshooting.