Sistemas de Operação / Fundamentos de Sistemas Operativos

(Ano letivo de 2022-2023)

## Guiões das aulas práticas

Summary

Understanding and dealing with concurrency using threads.

Using mutexes and condition variables to control access to a shared data structure, using the `pthread` library.

Previous note

In the code provided, the `pthread` library is not used directly. Instead, equivalent functions provided by the `thread.{h,cpp}` library are used. The functions in this library deal internally with error situations, either aborting execution or throwing exceptions, thus releasing the programmer of doing so. This library will be available during the practical exams.

Question 1 *Implementing a bounded-buffer application, using a shared FIFO and mutexes and condition variables.*
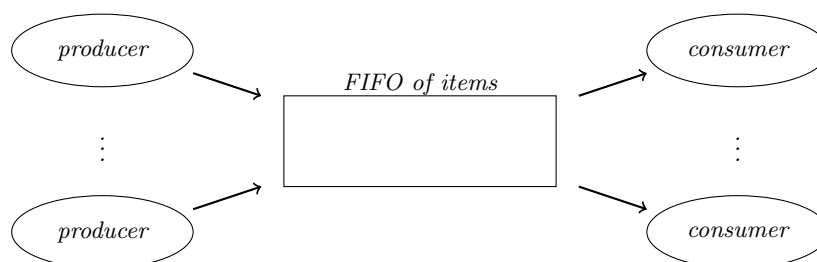
Directory `bounded_buffer` provides an example of a simple producer-consumer application, where interaction is accomplished through a buffer with limited capacity. The application relies on a FIFO to store the items of information generated by the producers, that can be afterwards retrieved by the consumers.

Each item of information is composed of a pair of integer values, one representing the id of the producer and the other the value produced. For the purpose of easily identify race conditions, the two least significant decimal digits of every value contains the id of its producer. Thus the id appears twice in each item of information. The number of producers is limited to 100.

There are 2 different implementations for the fifo: `fifo_unsafe` and `fifo_safe`.

(a) *Generate the* unsafe *version (`make bounded_buffer_unsafe`), execute it and analyse the results. Race conditions appear in red color.*

(b) *Look at the code of the* unsafe *version, `fifo_unsafe.cpp`, analyse it, and try to understand why it is unsafe.*

- *Try to understand how the shared FIFO is created.*
- *Try to understand why race conditions can appear.*
- *What should be done to solve the problem?*

(c) *Generate the* safe *version (`make bounded_buffer_safe`), execute it and analyse the results.*

(d) *Look at the code of the* safe *version, `fifo_safe`, and analyse it.*

- *Try to understand how mutexes and condition variables are used to avoid both race conditions and busy waiting.*

Question 2 *Designing and implementing a simple client-server application*

The figure below represents a simplified representation of a client-server concurrent system based on shared memory. The supporting (shared) data structure consists of a pool of N buffers of communication, individually identified by a number (between 0 and N−1), and two fifos, one of ids of buffers available and one of ids of buffers with pending orders. The same buffer is used for a client to place a request and the server to place the response to that request.
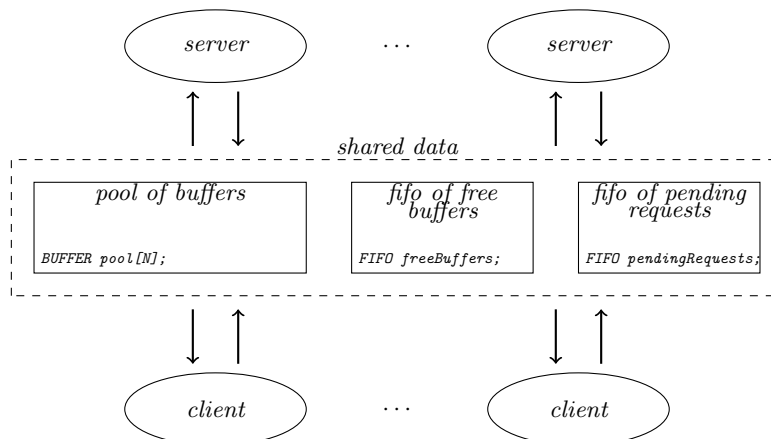
On the client side, interaction with the server takes place according to the following pseudo-code:

```
id = getFreeBuffer();          /* take a buffer out of fifo of free buffers */
putRequestData(data, id);      /* put request data on buffer */
addNewPendingRequest(id);      /* add buffer to fifo of pending requests */
waitForResponse(id);           /* wait (blocked) until a response is available */
resp = getResponseData(id);    /* take response out of buffer */
releaseBuffer(id);             /* buffer is free, so add it to fifo of free buffers */
```

On the server side, the interaction is described by the pseudo-code:

```
id = getPendingRequest();       /* take a buffer out of fifo of pending requests */
req = getRequestData(id);       /* take the request */
resp = produceResponse(req);    /* produce a response */
putResponseData(resp, id);      /* put response data on buffer */
signalResponseIsAvailable(id);  /* so client is waked up */
```

This is a double producer-consumer system, requiring three types of synchronization points:

- *the server must block while the fifo of pending requests is empty;*

- *a client must block while the fifo of free buffers is empty;*

- *a client must block while the response to its request is not available in the buffer.*

Note that in the last case there is a synchronization point per buffer. Note also that, as long as the fifos' capacities are at least the pool capacity, there is no need for a fifo full synchronization point.

Finally, consider that the purpose of the server is to process a sentence (string) to compute some statistics, specifically the number of characters, the number of digits and the number of letters.

(a) *Using the* safe *implementation of the fifo, used in the previous exercice, as a guideline, design and implement a solution to the data structure and its manipulation functions. Consider, for example, the following two main functions:*

```
void callService(ServiceRequest & req, ServiceResponse & res);
void processService();
```

The former is called by a client when it wants to be served; the latter is called by the server, in a cyclic way.

(b) *Implement the server thread, assuming that there will only be one.*

(c) *Implement the client thread.*

(d) *Does your solution work if there are more than one server?*