Sistemas de Operação / Fundamentos de Sistemas Operativos

(Ano letivo de 2020/2021)

Guiões das aulas práticas

Quiz #IPC/01 — Threads, mutexes, and condition variables

Summary

Understanding and dealing with concurrency using threads.
Programming using the `pthread` library.

Previous note

In the code provided, the `pthread` library is not used directly. Instead, it uses the equivalent functions provided by the `thread.{h,cpp}` library. This library will be available during the practical exams and allows to ignore the error conditions, as they are treated internally.

Question 1 *Understanding race conditions in the access to a shared data structure.*

Directory `incrementer` provides an example of a simple concurrent program, used to illustrate race conditions in the access to shared data by several concurrent threads. The data shared is a single integer value, which is incremented a number of times by the different threads. Each thread makes a local copy of the shared value, takes some time (delay) to simulate a complex operation on the value and copies the incremented value back to the shared variable. Three different operations are possible on the variable: set, get and increment its value.

(a) *Generate the* unsafe *version (`make incrementer_unsafe`), execute it and analyse the results.*

- *If N threads increment the variable M times each, why is the final value different from $N \times M$?*
- *Why is the final value different in different executions?*
- *Macros `UPDATE_TIME` and `OTHER_TIME` represent the times taken by the update operation and by some other work. Change the value of `OTHER_TIME` and verify if it affects the final value. Why?*

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

- *What should be done to solve the problem?*

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

(d) *Look at the code of the safe version. `inc_mod_safe`, analyse it, and try to understand why it is unsafe.*

- *Do you understand the role of the mutex?*

Question 2 *Implementing a bounded-buffer application using a monitor of the Lampson/Redell type.*

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 3 different implementations for the fifo: `fifo_unsafe`*,* `fifo_bwsafe`*, 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.*

   - *What should be done to solve the race conditions?*

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

(d) *Look at the code of the* bwsafe *version,* `fifo_bwsafe.cpp`*, analyse it, and try to understand why it is safe.*

   - *However, there is still a problem:* busy waiting*. Can you identify it?*

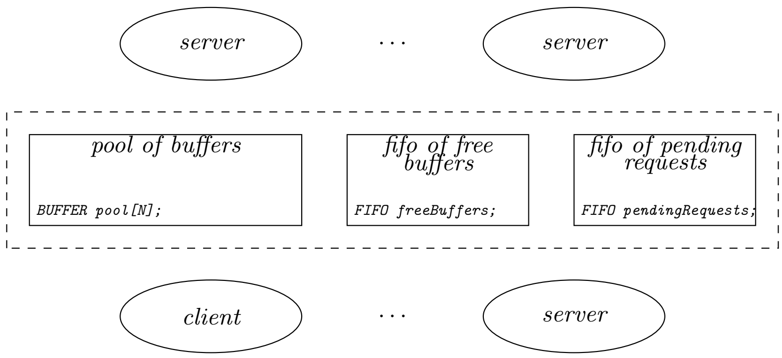
(e) *Generate the* safe *version (*`make bounded_buffer_safe`*), execute it and analyse the results.*

(f) *Look at the code of the* safe *version,* `fifo_safe.cpp`*, analyse it, and try to understand why it is safe and busy waiting free.*

   - *Absence of busy waiting is obtained by the use of condition variables. Try to understand how they are used.*

Question 3 *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?*
