Sistemas de Operação

(Ano letivo de 2018/2019)

Guiões das aulas práticas

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

Summary

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

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

(a) Directory `incrementer` *provides an example of a simple data structure 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 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.*

(b) 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 between executions?*
- *Macros* `UPDATE_TIME` *and* `OTHER_TIME` *represent the times taken by the update operation and by other work. Change the value of* `OTHER_TIME` *to 1000 and verify if it affects the final value. Why?*

(c) Module `inc_mod_unsafe` *implements the unsafe version of the operations. Analyse the code and try to understand why it is unsafe.*

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

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

(e) Module `inc_mod_safe` *implements the safe version of the operations. Analyse the code and try to understand why it is safe.*

Question 2 *Implementing a monitor of the Lampson / Redell type.*

(a) Directory `prodcon` *provides an example of a simple producer-consumer application. 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 a value produced. For the purpose of easily identify race conditions, the two least significant decimal digits of every value is the id of its producer. Thus the number of producers are limited to 100.*

There are 3 different implementations for the fifo: `fifo_unsafe`*,* `fifo_safe`*, and* `fifo_condsafe`*.*

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

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

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

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

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

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

(f) Generate the cond-safe *version (*`make prodcon_condsafe`*), execute it and analyse the results.*

(g) Look at the code of the cond-safe *version,* `fifo_condsafe`*, analyse it, and try to understand why it is safe and busy waiting free.*

- *Try to understand how conditions variables are used.*

Question 3 *Designing and implementing a simple client-server application*

(a) Consider a simple client-server system, with a single server and two or more clients. The server consumes requests and produces responses to the requests. The clients produce requests and consume the corresponding responses.

A solution to this system can be implemented using a pool (array) of item slots and a fifo of items' ids. A client sends a request to the server, choosing an empty slot, puting its request there, inserting the slot's id in the fifo, and waiting for the response. The server retrieves requests (actually slot ids) from the fifo, process them, put the responses in the slot, and notify the client.

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

- *the server must block while the fifo is empty;*
- *the clinets must block while the fifo is full;*
- *the clients must block while the response is not in the slot.*

Finally, consider that the purpose of the server is to convert a sentence (string) to upper case.

(b) Using the condsafe *implementation of the fifo, used in the previous exercice, as a guide line, design and implement a solution to the data structure and its manipulation functions. Consider the following possible functions:*

```
HANDLE sendRequest(ITEM);
ITEM getResponse(HANDLE);

HANDLE retrieveRequest();
ITEM getRequest(HANDLE);
setResponse(HANDLE, ITEM);
```

(c) Implement the server thread.

(d) Implement the client thread, adding a random delay between sending the request and receiving and displaying the response.

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