Sistemas de Operação

(Ano letivo de 2017/2018)

Guiões das aulas práticas

Quiz #IPC/01 About IPC/Threads

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 aplication. The aplication 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 is the id of the producer. Thus the number of producers are limited to 100.*

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

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

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

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

(e) *Module* `fifo_safe` *implements the safe, but busy waiting, version of the operations. Analyse the code and identify why busy waiting is presented.*

(f) *Use conditions variables to obtain a safe, non busy waiting version.*
