Sistemas de Operação

(Ano letivo de 2016/2017)

Guiões das aulas práticas

Quiz #IPC/02 About IPC/Shared memory and semaphores

Summary

Understanding and dealing with concurrency using shared memory.
Using semaphores to control access to a shared data structure, by different processes.

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 processes. The data shared is a single pair of integer value, which are incremented by the different processes. Each process increments both variables, taking some time (delay) between one and the other. Three different operations are possible on the variables: set, get and increment their values.*

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

- *If N processes increment both variables M times each, why are the final values different from $N \times M$?*
- *Why can the two variables have different values?*
- *Why are the final value different between executions?*
- *Macros* `SET_TIME`,`GET_TIME`, `INC_TIME` *and* `OTHER_TIME` *represent the times taken by the manipulating operations and by other work. Change their values and understand what happens. 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.*

(f) Reimplement the safe version using the POSIX version of semaphores (`man sem_overview`*).*

Question 2 *Implementing producer-consumer application, using a shared FIFO and semaphores.*

(a) Using an approach similar to the one used in the previous exercice, re-implement the unsafe version of the producer-consumer application of the previous lesson, using shared memory and concurrent processes.

(b) Using semaphores implement a safe. non busy waiting version.
