Operating Systems
Spring 2018

S05

AssistantArgs* args = arguments;

Professor : Philippe Cudré-Mauroux
Assistant : Ines Arous

Submitted by Sylvain Julmy

Exercice 2

a)

The problem appears when we try to add multiple consummer. We have to add multiple waiting bit in order to manage multiple processes.
We can also image a world where the wakeup bit is set to 1 and the signal is not lost. Then, when the process would have to go to sleep, it would not and produce a buffer overflow when creating to much item.

b)

It would be a mutual exclusion problem with the count variable. We admit that the producer wakeup the consummer and then try to increment the variable *count* by 1. Normally, the result would be $1 + 1 = 2$ for the count variable. Instead, the consummer process would compute the expression $1 - 1 = 0$ when decrementing the *count* variable and then would overwrite the result, so the *count* variable would have value 0 instead of 1 or 2 if no race condition appears.

Exercice 3

The code for exercise 3 is available in appendix.

Exercice 4

At first, it is not always easy to express synchronization in term of predicate proposition. We rather want to express synchronization in term of index in the program (the position before executing the next statement). For example, we would have to express position in the program using boolean variable that are assigned to *true* when a certain position is reach.
Additionally, the implementation of `WAITUNTIL` itself require to check many time the condition. So there is a loop on the condition like the following :

```
while(cond);
```

Exercice 5

a)

The mutex m_2 is playing the role of signal between function `sem_down` and `sem_up`. When we lock m_2, the thread goes to sleep and unlocking m_2 wakeup the thread.
The mutex m_3 garanteed that the value of *counter* can't be modified by another thread. When the program is going inside `sem_down`, the value of *counter* has to be -1 because it is the condition for the program to go inside the if of `sem_up`.
If we remove m_3, we could have an increment of *counter* between the unlock of m_1 and the lock of m_2, then $counter = 0$ and m_2 will stay blocked because we can't go inside the if of `sem_up`.

b)

We can remove the mutexes m_2 and m_3 and the controls structures inside `sem_up` and `sem_down`.