TP 8 Systèmes concurrents et distribués

Exercice 1 : Utilisez une/des sémaphore(s) équitable(s) pour implémenter une sémaphore FIFO. C'est-à-dire une nouvelle classe FIFOsemaphore avec les deux méthodes PFIFO() et VFIFO() qui ont la même sémantique que les méthodes P() et V() excepté que les threads sont 'servis' dans l'ordre d'arrivée. Vous pouvez utiliser un compteur pour gérer l'ordre d'arrivée et ne vous souciez pas d'éventuels dépassement de capacité.

Exercice 2 : Montrez que l'implémentation du tampon fini des pages 148-150 du cours est correcte. Montrez que le nombre d'exécutions complètes de la méthode deposit() (#deposit) est toujours plus grand que le nombre d'exécutions complètes de la méthode fetch() (#fetch), i.e. il n'y a pas d'undeflow suivant la terminologie de la page 138. En formule

$$\#deposit - \#fetch >= 0.$$

Montrez de même qu'il n'y a pas d'overflow (c.f. page 139), c'est-à-dire

$$\#deposit - \#fetch <= N.$$

Une idée est d'utiliser des invariants.

Attention : Dans l'implémentation du cours des méthodes *deposit()* et *fetch()* les accès aux variables *buffer, inBuf, outBuf,* doivent être protégés par un verrou (modifiez le code).

Exercice 3:

Proposez une solution du problème des lecteurs-rédacteurs qui met en œuvre les mécanismes Java (synchronized-wait/notify).

Proposez une solution avec priorités aux rédacteurs. Il y a plusieurs manières d'envisager cette priorité, définissez bien le mécanisme que vous implémentez.

Exercice 4:

Idem que l'exercice 3 mais avec des sémaphores, éventuellement équitables.