Universidad de los Andes
Ingeniería de Sistemas y Computación

ISIS 2203 Infraestructura Computacional

Taller de manejo de semáforos

1. El objetivo de este punto es desarrollar los semáforos en Java. Para esto, desarrolle una clase `semáforo` cuyos métodos implementen las operaciones de los semáforos. Puede usar todos los métodos privados que quiera, pero solo puede haber tres métodos públicos: P, V y un constructor para inicializar el semáforo. Necesitará usar métodos sincronizados. Nota: Este es un ejercicio de semáforos, por lo cual solo en la clase `semáforo` puede haber métodos sincronizados; pero los restantes puntos solo se pueden sincronizar por medio de semáforos
2. Patrones de sincronización con semáforos (del libro "The Little Book of Semaphores", de Allen B. Downey). Implemente las siguientes acciones usando semáforos, y escriba programas para probar que funcionan efectivamente.
   - *Exclusión mutua*. Se tiene un thread A con una instrucción a y un thread B con una instrucción b, y se quiere que a no pueda ejecutar al tiempo con b. Para esto, escriba dos métodos mA y mB (no sincronizados) que realicen alguna acción sobre una variable compartida (por ejemplo, uno incrementa y el otro decrementa la variable); el thread A llama a mA y el thread B llama a mB continuamente. Sincronice mA y mB usando semáforos para que A y B no puedan usar la variable al mismo tiempo.
   - *Multiplex*. Generalización de la exclusión mutua: n threads pueden estar simultáneamente en la sección crítica. Para esto, escriba un método m al cual todos los thread llaman continuamente. Sincronice m usando semáforos para que dos threads no puedan estar en el método al mismo tiempo.
   - *Señalamiento*. Se tiene un thread A con una instrucción a y un thread B con una instrucción b, y se quiere que a ejecute antes que b. Para esto, escriba dos métodos mA y mB que realicen alguna acción sobre una variable compartida; el thread A llama a mA y el thread B llama a mB. Sincronice mA y mB usando semáforos para que mB no pueda ejecutar a menos que mA ya haya ejecutado.
   - *Rendez-vous*. Se tiene un thread A con unas instrucciones $x_A;y_A$ y un thread B con unas instrucciones $x_B;y_B$, y se quiere que x_A ejecute antes que y_B, y que x_B ejecute antes que y_A. Es decir, las instrucciones x_A y x_B (las dos), deben ejecutar antes de $y_A;y_B$.
   - *Barrera*. Se tienen `n` threads y se quiere hacer una sincronización de barrera.
3. Usando la clase `semáforo`, implemente los lectores-redactores.

Nota: para probar sus programas puede hacer que en las secciones críticas impriman el estado del programa; también puede usar la función sleep para hacer que los threads se demoren en las secciones críticas (así incrementamos la probabilidad de que haya colisiones); igualmente puede usar aserciones.