Concurrence Using Semaphores. TASKS 1 & 2

Antonio Vieira Rubio - 2DAM (SEMI)

Ejercicio 1

Sean tres Threads Proc1, Proc2 y Proc3 con sus respectivas secciones de código expresadas más abajo. Los Threads mencionados, precisan sincronizarse y deben cumplir las siguientes reglas:

Inicializar semáforos:		
Hilo 1	Hilo 2	Hilo 3
C0()	C0()	C0()
SC()	SC()	SC()
C1()	C2()	C3()

1) Las secciones C0 pueden Executesrse concurrentemente, pero no más de dos al mismo tiempo.

2) Hasta haber terminado todas las C0, SC no empieza.
Estas secciones SC son consideradas críticas por cuanto actualizan variables compartidas.

3) Las secciones propias C1, C2 y C3 deben Executesrse precisamente en ese orden.

Utiliza los semáforos (con la notación propuesta por Dijkstra, P y V), nombrandolos como S0, S1, ... conforme vaya necesitando hacer uso de ellos y sin olvidar asignarles valores de inicialización, por garantizar que los diferentes procesos y secciones cumplen con todas las reglas dadas.

Setting up:

S0(2)
Mutex(1)
S1(0)
S2(0)
S3(0)
S4(0)
S5(0)

C0	Thread 1	Thread 2	Thread 3
Action	S0.Expends(1)	S0.Expends(1)	Waits (S0=0)
Status S0	S0 = 1	S0 = 0	-
Action	Executes C0	Executes C0	-
Action	S0.Produces(1)	-	S0.Expends(1)
Status S0	S0 = 1	-	S0 = 0
Action	-	S0.Produces(1)	Executes C0
Action	S1.Produces(1)	S2.Produces(1)	S3.Produces(1)

- **S0(2):**

  This semaphore controls access to section C0, allowing a maximum of two threads to execute it simultaneously. We initialize it at 2 because it permits up to two threads to enter C0 at the same time.

- **S1(0), S2(0), S3(0):**

  These semaphores ensure that the SC sections do not execute until all threads have completed their respective C0 sections. By initializing them at 0, you block the execution of SC until they are incremented through the "Produce" operations. In other words, the SC sections cannot start until the threads have finished their C0 sections and "release" the permissions of these semaphores.

SC	Thread 1	Thread 2	Thread 3
Action	S1.Expends(1)	S2.Expends(1)	S3.Expends(1)
Action	Mutex.Expends(1)	Waits (Mutex=0)	Waits (Mutex=0)
Status Mutex	Mutex = 0	-	-
Action	Executes SC	-	-
Action	Mutex.Produces(1)	-	-
Status Mutex	Mutex = 1	Mutex.Expends(1	Waits (Mutex=0)
Action	-	Executes SC	-
Action	-	Mutex.Produces(1)	-
Status Mutex	-	Mutex = 1	-
Action	-	-	Mutex.Expends(1)
Status Mutex	-	-	Mutex = 0

Action	-	-	Executes SC
Action	-	-	Mutex.Produces(1
Status Mutex	-	-	Mutex = 1

- **Mutex(1):**

  The Mutex semaphore ensures mutual exclusion in the critical section SC, allowing only one thread at a time to access this section. By initializing it at 1, you allow only one thread to enter SC, and any other thread will have to wait until the first one has finished and releases the semaphore.

C	Thread 1 C1	Thread 2 C2	Thread 3 C3
Status C1	Executes C1	-	-
Action	S4.Produces(1)	S4.Expends(1)	-
Status C2	-	Executes C2	-
Action	-	S5.Produces(1)	S5.Expends(1)
Status C3	-	-	Executes C3

- **S4(0), S5(0):**

  These semaphores are used to control the execution order of sections C1, C2, and C3, so that each section depends on the completion of the previous one. By initializing them at 0, you ensure that C2 does not start until C1 has finished, and the same applies to C3.

Ejercicio 2

Sean tres hilos Fil1, Fil2 y Fil3 con sus respectivas secciones de código expresadas más abajo. Los hilos mencionados, precisan sincronizarse y deben cumplir las siguientes reglas:

Inicializar semáforos:		
Hilo 1	Hilo 2	Hilo 3
C1()	C1()	C2()
C3()	C3()	C3()
C4()	C4()	C4()

1) La sección C2 no debe ser ejecutada hasta que las dos secciones C1 hayan terminado.
2) Las tres secciones C3 no pueden ser ejecutadas al mismo tiempo en paralelo.
3) Las secciones C4, son críticas y deben ejecutarse en exclusión mutua.

Utilizando semáforos (con la notación propuesta por Dijkstra), S1, S2, ... conforme vaya necesitándose, y sin olvidar asignarles valores de inicialización, complete el código, a fin de garantizar que los hilos hacen que el proceso funcione con las reglas dadas.

Setting up:

S0(2)
S1(1)
S2(1)
S3(1)
S4(1)
S5(1)
S6(1)
S7(1)
Mutex(1)

	Thread 1 C1	Thread 2 C1	Thread 3 C2
Action	S0.Expends(1)	S0.Expends(1)	Waits (S0=0)
Status C1	S0 = 1	S0 = 0	-
Action	Executes C1	Executes C1	-
Action	-	-	S1.Expends(1)
Status C2	-	-	S1 = 0
Action	-	-	Executes C2

C3	Thread 1	Thread 2	Thread 3
Action	S2.Expends(1)	Waits (S2=0)	Waits (S3=0)
Status S2	S2 = 0	-	-
Action	Executes C3	-	-
Action	-	S3.Expends(1)	Waits (S3=0)
Status S3		S3 = 0	-
Action	-	Executes C3	
Action	-	-	S4.Expends(1)
Status S4		-	S4 = 0
Action	-	-	Executes C3

C4	Thread 1	Thread 2	Thread 3
Action	S5.Expends(1)	S6.Expends(1)	S7.Expends(1)
Action	Mutex.Expends(1)	Waits (Mutex=0)	Waits (Mutex=0)
Status Mutex	Mutex = 0	-	-
Action	Executes C4	-	-
Action	Mutex.Produces(1)	-	-
Status Mutex	Mutex = 1	Mutex.Expends(1)	Waits (Mutex=0)
Action	-	Executes C4	-
Action	-	Mutex.Produces(1)	-
Status Mutex	-	Mutex = 1	-
Action	-	-	Mutex.Expends(1)
Status Mutex	-	-	Mutex = 0
Action	-	-	Executes C4
Action	-	-	Mutex.Produces(1
Status Mutex	-	-	Mutex = 1