1- **deadblok**: Es cuando dos procesos se encuentran a la espera de que otro haga algo para poder avanzar por lo cual ninguno avanza,

Ej-Dos autos quieren pasar por un solo carril y ambos están esperando que el otro pase, quedan bloqueados. Cada auto tiene un recurso (su espacio actual) y necesita el recurso del otro (el espacio libre adelante).

**livelock**: Es cuando ambos procesos quedan en un bucle en el cual cambian continuamente de estado por acciones de otros procesos pero sin llegar a buen puerto (sin ejecutarse correctamente).

Ej-Dos personas quieren pasar por un pasillo angosto, para dejar pasar a la otra se mueven constantemente de izquierda a derecha pero al mismo tiempo, están activos moviendosè pero es un bucle en el cual no consiguen el resultado esperado.

2-

1. Puede haber deadlock ya que P1 y P2 adquieren R1 (lo comparten, está en su límite). P1 solicita R2 → espera porque P3 lo tiene. P3 ahora solicita R1 → pero ya está en uso por P1 y P2, y no puede entrar. Por lo que P1 espera R2 (que tiene P3) y P3 espera R1 (ocupado por P1 y P2). **Ciclo de espera circular** = **deadlock**.

También livelock ya que si los procesos detectan un posible bloqueo e intentan librar recursos y esto lo hacen al mismo tiempo, cambian de estado sin éxito.

1. Menos probable pero si porque r2 sique siendo exclusivo.

En el caso de livelock también, misma explicación.

3- El estado es de deadlock ya que por ejemplo los tres procesos utilizan tres recursos, pero si, acorde lo planteado en el ejercicio, quieren utilizar 2 recursos por proceso (lo que sería 6) no alcanzarían los recursos y hay posibilidad de que queden a la espera de que se liberen, pero si los 3 quedan a la espera hay deadlock.

4- a) El sistema no está libre de deadlock, ya que si p1 está usando r1, p2 r2 y p3 r3, en caso de que p1 solicite r2 por ej. y p2 a r1 o p1 a r3 quedarían en espera para siempre bloqueándose.

b) NO influye ya que si lo usa y cuando termina de usar lo libera no influiría ya que no requiere otro recurso.  
c) Si me aseguro de que está libre de deadlock ya que p3 solo usa r3 y termina, por lo que p1 utilizaría los tres recursos y luego entraría p2.

5-a)

p1 – x-- / x = 9

p2 – x-- / x = 8

p1 – x++ / x = 9

p1 – printf("x is %d",x); / “x is 9”

b)

p1 – x-- / x = 9

p2 – x-- / x = 8

p1 – x++ / x = 9

p2 – x++ / x is 10

p1 - if (x != 10) { printf("x is %d",x) / por una concurrencia entro en el if cuando valìa 9 pero imprime 10 ya que p2 lo incrementò:

“x is 10”

6-a) El problema es que hay condición de carrera en la variable peticiones (es cuando varios procesos leen o escriben al mismo tiempo en concurrencia y puede variar el resultado final pudiendo no ser el esperado) lo que puede causar inconsistencias en el valor final, se requiere aplicar semáforos para gestionar el acceso al recurso.

b- Al mismo tiempo R2 lee que peticiones es 1 y le suma 1 pero R1 tambien lee que es 1 y resta 1 pasando a ser 0 cuando tendría que ser 1 o 2 (según el orden correcto).

R2 lee peticiones =1  
R1 lee peticiones =1 (al mismo tiempo)  
R2 se suma 1 (queda en 2)  
R1 se resta 1 (queda en 0)

R2 guarda el dato en peticiones (peticiones = 2)

R1 guarda el dato en peticiones (peticiones =0) (error, tendría que ser 1 o 2 según el caso)

Esto sucede en condición de carrera.

7- Hice el código nashe

8-Puede caer en deadlock ya que p1 obtiene el sem1, p2 sem2 y luego p1 quiere llamar a sem2 pero esta ocupado con p2 por lo que queda bloqueado, por su parte p2 llama a sem1, que esta ocupado con p1, por lo que se bloquea también.

10-Problema:Se espera el mutex primero, y luego el elementoDisponible. Esto genera un riesgo de bloqueo innecesario: si el buffer está vacío, el consumidor toma el mutex y se bloquea en sem\_wait(elementoDisponible), lo que bloquea al productor al intentar entrar al buffer.