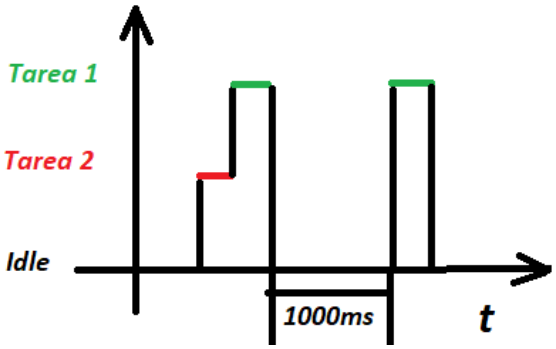
4)_ Si se podría, pero la zona critica que estoy protegiendo tiene que tardar muy poco tiempo para no aumentar la latencia de todas las tareas ya que como el planificador es apropiativo la interrupción del timer que hace que este cambie de tarea no se realizaría evitando que cuando está enviando el dato por el puerto no se interrumpa y ejecutaría el recurso compartido de manera atómica.

No sería útil usar semáforos ya que la zona critica que es el puerto serie se esta ejecutando de manera atómica.

5)_ Tarea 2 toma el semáforo p_sem_a ***prende Led verde*** y hace el cambio de contexto debido a que se bloquea con vTaskDelay pasa ejecutar la Tarea 1 que es de menor prioridad y esta toma el semáforo p_sem_b ***prende el Led amarillo*** y cuando toma el semáforo p_sem_a se bloquea debido a que lo tomo la Tarea2 pero este bloqueo dura 1000ms, entonces cuando retome la Tarea 2 retome va intentar tomar al semáforo p_sem_b que va a estar ocupado por la tarea1 entonces se va a bloquear por 1000ms, como el semáforo p_sem_b se tomo 50ms antes debido al tiempo de vTaskDelay primero va a continuar la tarea 1 donde había dejado liberando p_sem_b apaga el Led libera libera el semaforo y justo alcanza a ejecutar de nuevo el bucle tomando el semáforo p_sem_b dentro de los 50ms continua la tarea 2 que queda bloqueada aun y se da el interbloqueo.




```
/*==================[inclusions]=============================================*/

#include "board.h"

#include "FreeRTOS.h"

#include "task.h"

#include "semphr.h"

/*==================[macros and definitions]=================================*/

#define PRIO_TAREA1 1

#define PRIO_TAREA2 2

#define TAM_PILA 150

#define mainDELAY_LOOP_COUNT        ( 0xffffff )

SemaphoreHandle_t p_sem_a;

SemaphoreHandle_t p_sem_b;
```

```
/*==================[internal data declaration]==============================*/

/*==================[internal functions declaration]=========================*/

/*==================[internal data definition]===============================*/

/*==================[external data definition]===============================*/

/*==================[internal functions definition]==========================*/

static void vTarea1(void *pvParameters)
{
  const char *pcTaskName = "Tarea 1 menos prioritaria\r\n";
  const TickType_t xDelay1000ms = pdMS_TO_TICKS( 1000UL );
  printf( pcTaskName );

   for( ;; ) {
    if((xSemaphoreTake( p_sem_b, xDelay1000ms)) == pdPASS)
     {
       printf("Tomo el semaforo p_sem_b Tarea 2")
     }
     Board_LED_Set(3,TRUE); //Enciende "LED 1" ( amarillo )
     if(xSemaphoreTake( p_sem_a, xDelay1000ms) == pdPASS)
     {
        printf("Tomo el semaforo p_sem_a Tarea 2")
     }
     /* El puerto serie está libre */
     printf( pcTaskName );
     xSemaphoreGive(p_sem_b);
     Board_LED_Set(3,FALSE); //Apaga "LED 1" ( amarillo )
     xSemaphoreGive(p_sem_a);
   }
}
```

```
static void vTarea2(void *pvParameters)
{
  const char *pcTaskName = "Tarea 2 mas prioritaria \r\n";
  const TickType_t xDelay50ms = pdMS_TO_TICKS( 50UL );
  const TickType_t xDelay200ms = pdMS_TO_TICKS( 200UL );
  const TickType_t xDelay1000ms = pdMS_TO_TICKS( 1000UL );
  printf( pcTaskName );

  for( ;; ) {
    if(xSemaphoreTake( p_sem_a, xDelay1000ms) == pdPASS)
    {
      printf("Tomo el semaforo p_sem_a Tarea 1")
    }
    Board_LED_Set(5,TRUE); //Enciende "LED 3" ( verde )
    vTaskDelay( xDelay50ms ); //para que SIEMPRE haya Deadlock ;-)
    if(xSemaphoreTake( p_sem_b, xDelay1000ms) == pdPASS)
    {
       printf("Tomo el semaforo p_sem_b Tarea 1")
    }
    //if ((xSemaphoreTake( p_sem_b, xDelay200ms)) == pdPASS) {
    /* El puerto serie está libre */
    printf( pcTaskName );
    xSemaphoreGive(p_sem_a);
    Board_LED_Set(5,FALSE); //Apaga "LED 3" ( verde )
    xSemaphoreGive(p_sem_b);
    /*}
    else {
     xSemaphoreGive(p_sem_a);
     vTaskDelay( xDelay50ms ); //para que SIEMPRE haya Deadlock ;-)
     }*/

  }
}
```

```
/*==================[external functions definition]==========================*/


int main(void)

{

   //Se inicializa HW

         /* Se crean las tareas */

         p_sem_a = xSemaphoreCreateMutex ();  //se inicializa por defecto en 1

         p_sem_b = xSemaphoreCreateMutex ();  //se inicializa por defecto en 1


         xTaskCreate(vTarea1, (const char *)"Tarea1", TAM_PILA, NULL, PRIO_TAREA1, NULL );

         xTaskCreate(vTarea2, (const char *)"Tarea2", TAM_PILA, NULL, PRIO_TAREA2, NULL );


         vTaskStartScheduler(); /* y por último se arranca el planificador . */

   //Nunca llegara a ese lazo  .... espero

    for( ;; );

    return 0;


}


/*==================[end of file]============================================*/
```

6)_ Inhabilitar las interrupciones:

- ✓ ***Desventajas***: si la zona critica tarda mucho tiempo en ejecutarse aumenta la latencia tanto de foreground como baground.
- ✓ ***Ventaja***: si la zona critica tarda poco tiempo es un método rápido debido a que se realiza con pocas instrucciones de máquina.

Inhabilitar las conmutaciones de tareas:
Se realiza con vTaskSuspendAll

- ✓ ***Desventajas***: sólo afectará a los tiempos de respuesta de las tareas de primer plano, que normalmente no son tan estrictos como los de las rutinas de interrupción es decir aumenta la latencia en badground
- ✓ ***Ventaja***: la tarea se ejecuta hasta que termine y no cambia su contexto por otras tareas de mayor prioridad por lo tanto no tenemos problemas de concurrencia.

Semáforos:

- ✓ ***Desventajas***: al ser un mecanismo más complejo, su gestión por parte del sistema operativo presenta una mayor carga al sistema
- ✓ ***Ventaja***: sólo afecta a las tareas que comparten el recurso