```c
#include "sl_component_catalog.h"
#include "sl_system_init.h"
#include "app.h"
#if defined(SL_CATALOG_POWER_MANAGER_PRESENT)
#include "sl_power_manager.h"
#endif
#if defined(SL_CATALOG_KERNEL_PRESENT)
#include "sl_system_kernel.h"
#else // SL_CATALOG_KERNEL_PRESENT
#include "sl_system_process_action.h"
#endif // SL_CATALOG_KERNEL_PRESENT

#include "em_device.h"
#include "em_chip.h"

#include "FreeRTOS.h"
#include "task.h"
#include "semphr.h"

/***************************************************************************//**
 * Extern Includes
 ******************************************************************************/
extern void task_A(), task_B(), task_C(), task_D();

SemaphoreHandle_t semA = NULL;
SemaphoreHandle_t semB = NULL;
SemaphoreHandle_t semC = NULL;
SemaphoreHandle_t semD = NULL;

int tick_count = 0;

void TaskA(void *params)
{
  (void) params; // suppress warning
  for(;;)
  {
    if(xSemaphoreTake(semA,portMAX_DELAY))
    {
      task_A(); // perform actual task
    }
  }
}

void TaskB(void *params)
{
  (void) params; // suppress warning
  for(;;)
  {
    if(xSemaphoreTake(semB,portMAX_DELAY))
    {
      task_B(); // perform actual task
    }
  }
}

void TaskC(void *params)
{
```



```c
  (void) params; // suppress warning
  for(;;)
  {
    if(xSemaphoreTake(semC,portMAX_DELAY))
    {
      task_C(); // perform actual task
    }
  }
}

void TaskD(void *params)
{
  (void) params; // suppress warning
  for(;;)
  {
    if(xSemaphoreTake(semD,portMAX_DELAY))
    {
      task_D(); // perform actual task
    }
  }
}

void vApplicationTickHook(void)
{
  //used to increase the tick count when an ISR is done
  if((tick_count % 30) == 0)
  {
      xSemaphoreGiveFromISR(semA, NULL);
  }
  if((tick_count % 40) == 0)
  {
      xSemaphoreGiveFromISR(semB, NULL);
  }
  if((tick_count % 60) == 0)
  {
      xSemaphoreGiveFromISR(semC, NULL);
  }
  if((tick_count % 109) == 0)
  {
      xSemaphoreGiveFromISR(semD, NULL);
  }
  tick_count++;
}

int main(void)
{
  // Vendor function to work around bugs in some versions of the hardware
    CHIP_Init();

    // Create a semaphore
    semA = xSemaphoreCreateBinary();
    semB = xSemaphoreCreateBinary();
    semC = xSemaphoreCreateBinary();
    semD = xSemaphoreCreateBinary();

    //use xTaskCreate(function name, "string name", configMINIMAL_STACK_SIZE, NULL, PRIORITY, NULL);
```



```c

    xTaskCreate(TaskA,
                "TaskA",
                configMINIMAL_STACK_SIZE,
                NULL,
                4,
                NULL);

    xTaskCreate(TaskB,
                "TaskB",
                configMINIMAL_STACK_SIZE,
                NULL,
                3,
                NULL);

    xTaskCreate(TaskC,
                "TaskC",
                configMINIMAL_STACK_SIZE,
                NULL,
                2,
                NULL);

    xTaskCreate(TaskD,
                "TaskD",
                configMINIMAL_STACK_SIZE,
                NULL,
                1,
                NULL);

    vTaskStartScheduler();
    //vApplicationTickHook();

  while (1) {

  }
}

```