Университет ИТМО

Лабораторная работа № 4

по дисциплине «Встроенные системы»

Студент:
P3410 Нгу Фыонг Ань
Преподаватель:

Санкт-Петербург
2020 г.

Цель работы – освоение работы с операционной системой реального времени FreeRTOS.

ПОРЯДОК ВЫПОЛНЕНИЯ ЛАБОРАТОРНОЙ РАБОТЫ

1. Изучить теоретическую часть.
2. Изменить проект ЛР 3, добавив FreeRTOS с помощью CubeMX (вкладка Middleware).
3. Изучить структуру нового проекта.
4. Скомпилировать проект без изменений в исходном коде и запустить в ITMO.cLAB.
5. Определить, почему в таком случае система cLAB возвращает ошибку (связано с трассировочным логом) и решить данную проблему.
6. Разработать алгоритм программы, реализующей действия согласно варианту из таблицы 1.
7. Написать программное обеспечение в соответствии с разработанным алгоритмом.
8. Собрать проект в Debug режиме.
9. Загрузить сгенерированный бинарный файл на ITMO.cLAB для проверки правильности выполнения задания.
10. По итогам работы написать и защитить отчет. Отчет должен содержать описание теоретической и практической частей, а также основной код программы. В процессе защиты работы требуется демонстрация реализованного функционала, поэтому .bin файл для загрузки на ITMO.cLAB должен быть готов к началу сдачи.

№ Варианта	Задание
2*	В одной задаче необходимо с периодом 1 сек. генерировать значение uint32_t X, которое будет являться параметром для третьей задачи. В другой задаче необходимо генерировать с тем же периодом значени uint32_t Y, которое также будет являться параметром для третьей задачи. В третьей задаче считается следующая функция: Z = X - Y, причем параметры в эту задачу передаются через очереди. В трассировочный график выводить сгенерированные значения и вычисленные значени функции.

*Для генерации случайных чисел:

1. в CubeMX активировать RNG (вкладка Security);
2. #include "rng.h";
3. uint32_t random_value = HAL_RNG_GetRandomNumber(&hrng);

1. Теория

1. Принцип работы FreeRTOS

FreeRTOS – многозадачная операционная система реального времени для встраиваемых систем. За переключение между задачами отвечает диспетчер, работающий по прерыванию от таймера. Задача выглядит как обычная функция Си, которая выполняет некоторое действие в бесконечном цикле.

2. Назначение и применение FreeRTOS

Ядро FreeRTOS - это операционная система реального времени, поддерживающая множество архитектур. Он идеально подходит для создания приложений для встроенных микроконтроллеров. Это обеспечивает:

- Планировщик многозадачности.
- Множественные варианты распределения памяти (включая возможность создания полностью статически распределенных систем).
- Примитивы межзадачной координации, включая уведомления о задачах, очереди сообщений, несколько типов семафоров, а также буферы потоков и сообщений.

Ядро FreeRTOS никогда не выполняет недетерминированные операции, такие как просмотр связанного списка внутри критического раздела или прерывания. Ядро FreeRTOS включает эффективную реализацию программного таймера, которая не использует процессорное время, если только таймер не требует обслуживания. Заблокированные задачи не требуют трудоемкого периодического обслуживания. Прямые уведомления о задачах позволяют быстро сигнализировать о задачах, практически без накладных расходов на оперативную память. Их можно использовать в большинстве сценариев передачи сигналов между задачами и прерываний.

3. Преимущества и недостатки FreeRTOS

Преимущества:

- многозадачность;
- временная база для измерения интервалов времени;
- обмен данными между задачами через очередь и синхронизацию.

Недостатки:

- увеличение потребной памяти программ для реализации ядра;
- большее количество памяти для хранения стека каждой задачи, семафоров, очередей, мьютексов и других объектов ядра системы;
- задержки при переключении между задачами на сохранение контекста.

4. Примитивы синхронизации

Примитивы синхронизации - это простые программные механизмы, предназначенные для поддержки синхронизации потоков или процессов. Мьютекс, событие, условные переменные и семафоры - все это примитивы синхронизации.

5. Кооперативную и вытесняющую многозадачность

Кооперативная многозадачность — тип многозадачности, при котором следующая задача выполняется только после того, как текущая задача явно объявит себя готовой отдать процессорное время другим задачам.

Вытесняющая многозадачность — это вид многозадачности, при которой операционная система может временно прервать текущий процесс без какой-либо помощи с его стороны.

6. Принцип действия планировщика задач

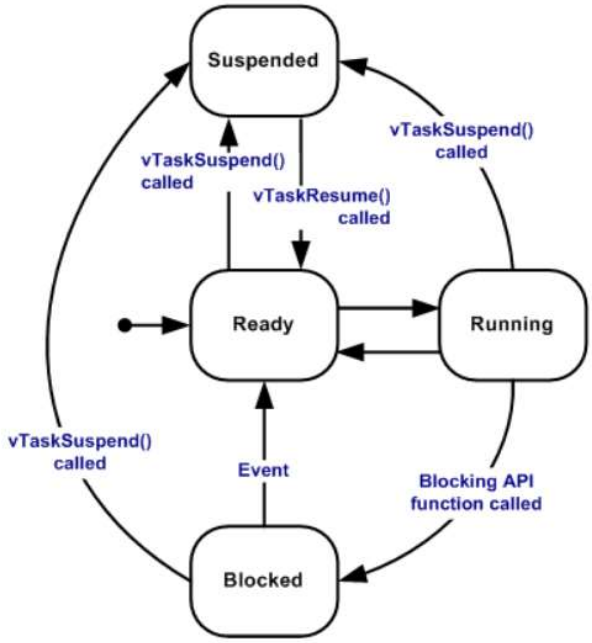
Чтобы обеспечить детерминированное поведение в реальном времени, планировщик задач FreeRTOS позволяет назначать задачам строгие приоритеты. RTOS гарантирует, что задача с наивысшим приоритетом, которая может быть выполнена, получает время

обработки. Это требует разделения времени обработки между задачами с равным приоритетом, если они готовы работать одновременно. FreeRTOS также создает простаивающую задачу, которая выполняется только тогда, когда другие задачи не готовы к запуску.

7. Состояния задачи

A task can exist in one of the following states:

- Running: task is executing. If the processor on which the RTOS is running only has a single core then there can only be one task in the Running state at any given time.
- Ready: tasks are able to execute (they are not in the Blocked or Suspended state) but are not currently executing because a different task of equal or higher priority is already in the Running state.
- Blocked: A task is waiting for either a temporal or external event. For example, if a task calls vTaskDelay() it will block (be placed into the Blocked state) until the delay period has expired – a temporal event. Tasks can also block to wait for queue, semaphore, event group, notification or semaphore event. Tasks in the Blocked state normally have a 'timeout' period, after which the task will be timeout, and be unblocked, even if the event the task was waiting for has not occurred.
  Tasks in the Blocked state do not use any processing time and cannot be selected to enter the Running state.
- Suspended: Like tasks that are in the Blocked state, tasks in the Suspended state cannot be selected to enter the Running state, but tasks in the Suspended state do not have a time out. Instead, tasks only enter or exit the Suspended state when explicitly commanded to do so through the vTaskSuspend() and xTaskResume() API calls respectively.



Valid task state transitions

8. Квант времени работы планировщика

Квант времени работы планировщика (tick) — это жестко фиксированный отрезок времени, в течение которого планировщик не вмеши-вается в выполнение задачи. По истечении кванта времени планировщик получает воз-можность приостановить текущую задачу и возобновить следующую, готовую к выполнению.

9. Приоритет задачи

Каждой задаче назначается приоритет от 0 до (configMAX_PRIORITIES - 1), где configMAX_PRIORITIES определяется в FreeRTOSConfig.h.

Номера с низким приоритетом обозначают задачи с низким приоритетом. Неактивная задача имеет нулевой приоритет (tskIDLE_PRIORITY).

Планировщик FreeRTOS гарантирует, что задачам в состоянии готовности или выполнения всегда будет отдаваться процессорное время (ЦП), а не задачам с более низким приоритетом, которые также находятся в состоянии готовности. Другими словами, задача, переведенная в состояние «Выполняется», всегда является задачей с наивысшим приоритетом, которая может выполняться.

Любое количество задач может иметь одинаковый приоритет. Если configUSE_TIME_SLICING не определен или configUSE_TIME_SLICING установлен в 1, то задачи состояния готовности с равным приоритетом будут разделять доступное время обработки с использованием схемы циклического планирования с временным разделением.

2. Исходный код

In main.c:

```
int main(void)
{
  //init part

  /* Do not remove this code below */
  MX_TRACE_Init();
  SDK_TRACE_Start();
  /* Do not remove this code from above */

  /* USER CODE END 2 */

  /* Call init function for freertos objects (in freertos.c) */
  MX_FREERTOS_Init();
  /* Start scheduler */
  osKernelStart();

  //…
}
```

In freertos.c:

Variables used to store generated value and queue to store data:

```
uint32_t x = 0;
```

```
uint32_t y = 0;
uint32_t received_x = 0;
uint32_t received_y = 0;
xQueueHandle q1;
xQueueHandle q2;
```

Task to stop SDK_TRACE:

```
void StartDefaultTask(void const * argument)
{
  /* USER CODE BEGIN StartDefaultTask */
  /* Infinite loop */
  for(;;)
  {
    osDelay(6000);
    SDK_TRACE_Stop();
  }
  /* USER CODE END StartDefaultTask */
}
```

Task to generate X and push X to queue q1:

```
void StartTask02(void const * argument)
{
  /* USER CODE BEGIN StartTask02 */
  /* Infinite loop */
  for(;;)
  {
        if (xSemaphoreTake(myBinarySem01Handle, portMAX_DELAY)) {
              if(uxQueueSpacesAvailable(q1) == 0){
                          xQueueReset(q1);
                    }
              x = HAL_RNG_GetRandomNumber(&hrng);
              SDK_TRACE_Print("Value X: %u", x);
              xQueueSend(q1,(void *) &x,portMAX_DELAY);
              osDelay(500);
              xSemaphoreGive(myBinarySem01Handle);
              osDelay(1);
        }
  /* USER CODE END StartTask02 */
  }
}
```

Task to generate Y and push Y to queue q2:

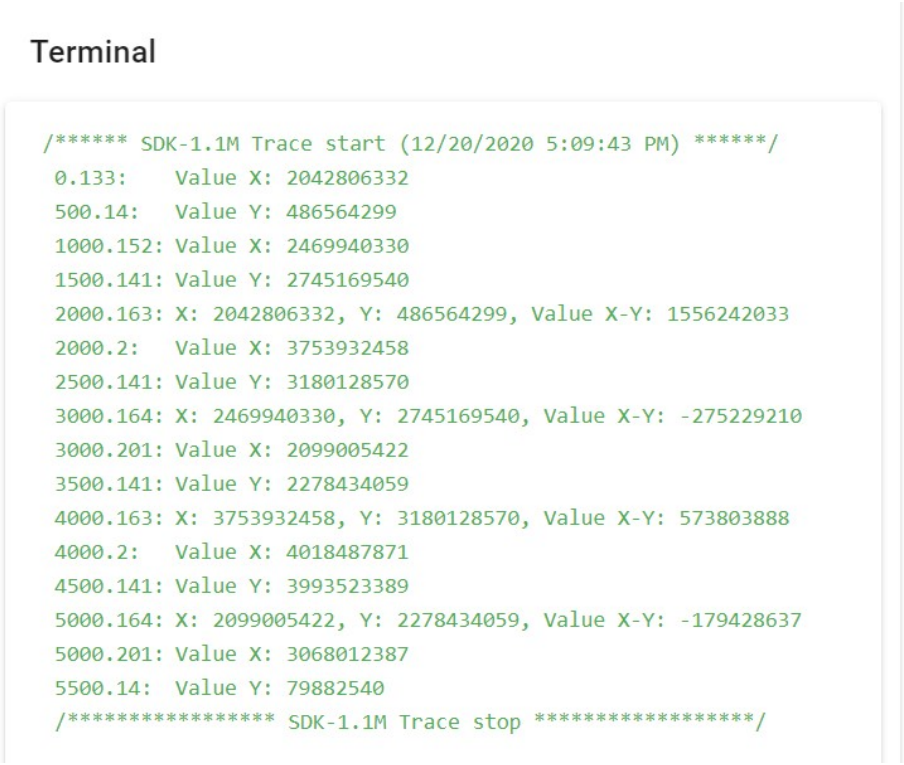
```
void StartTask03(void const * argument)
{
  /* USER CODE BEGIN StartTask03 */
  /* Infinite loop */
  for(;;)
  {
        if (xSemaphoreTake(myBinarySem01Handle, portMAX_DELAY)) {
              if(uxQueueSpacesAvailable(q2) == 0){
                          xQueueReset(q2);
                    }
              y = HAL_RNG_GetRandomNumber(&hrng);
```

```
                    SDK_TRACE_Print("Value Y: %u", y);
                    xQueueSend(q2,(void *) &y,portMAX_DELAY);
                    osDelay(500);
                    xSemaphoreGive(myBinarySem01Handle);
                    osDelay(1);
              }
        }
        /* USER CODE END StartTask03 */
}
```

Task to calculate Z=X-Y:

```
      void StartTask04(void const * argument)
      {
        /* USER CODE BEGIN StartTask04 */
        /* Infinite loop */
        for(;;)
        {
              if (xSemaphoreTake(myBinarySem01Handle, portMAX_DELAY)) {
                    if ((uxQueueMessagesWaiting(q1)>1) &&
      (uxQueueMessagesWaiting(q2)>1)){
                          xQueueReceive(q1, &(received_x), portMAX_DELAY);
                          xQueueReceive(q2, &(received_y), portMAX_DELAY);
                          if (received_x<received_y){
                                SDK_TRACE_Print("X: %u, Y: %u, Value X-Y: -%u",
      received_x, received_y, received_y-received_x);
                          } else
                                SDK_TRACE_Print("X: %u, Y: %u, Value X-Y: %u",
      received_x, received_y, received_x-received_y);
                    }
                    xSemaphoreGive(myBinarySem01Handle);
                    osDelay(1);
                    }
        }
        /* USER CODE END StartTask04 */
}
```

3. Результат



4. Вывод

В ходе выполнения лабораторной работы была изучен принцип работы и применение FreeRTOS и написана программа, реализующая работы с операционной системой реального времени FreeRTOS.