FreeRtos 获取任务运行时状态以及实现方式分析

原创 gdut_llkkyy 于 2022-03-10 21:19:26 发布 2261 收藏 3 版权

分类专栏： RTOS 文章标签： 单片机 stm32 嵌入式硬件 rtos

一、获取全部任务的状态

1.1 系统API函数

```
void vTaskList( char * pcWriteBuffer )
```

1.2 函数功能

返回所有任务的：

- 任务名
- 任务当前运行状态
- 任务优先级
- 任务剩余的最大栈空间
- 任务ID。

1.3 函数参数

传入一个缓冲区地址。这个缓冲区要足够的长。因为vTaskList，只传入一个缓冲区地址信息，而没有传入缓冲区长度。在函数内部，没办法做缓冲区越界处理。如果传入的缓冲区空间太小，会引起缓冲区越界。缓冲区所需的大小跟系统的任务个数有关。可以先分配一个较大的缓冲区，例如500字节，调用 `vTaskList(buf)` 后，通过 `strlen（buf)` 求出实际所需的大小，再调整。

1.4 函数使用

将缓冲区传入vTaskList，vTaskList 里面把全部的任务状态格式化之后，填入缓冲区。然后再打印缓冲区。

```
char buf[300] = {0};
void task_debug_entry(void *parameter)
{
    task_debug_init();

    while (1)
    {
        os_delay_ms(6000);
        vTaskList(buf);
        app_log("\r\n taskName   status    pri   leftStack     taskNum \r\n");
        app_log("%s",buf);
        app_log("len:%d\n",strlen(buf));

    }
}
```

串口打印输出

```
 taskName   status    pri   leftStack     taskNum
DebugTask       X       8       174     8
IDLE            R       0       112     3
```

二、获取任务的cpu占用率

2.1 系统api 函数

```
void vTaskGetRunTimeStats( char *pcWriteBuffer )
```

2.2 函数功能

返回所有任务的：

- 任务名
- 任务运行时间：从任务第一次被调度，到当前。
- 任务的cpu 占用率。

2.3 函数参数

参照上面 `vTaskList`

2.4 计算cpu 使用率的原理和实现方式

2.4.1 原理

计算每一个任务每次执行的时间，并把这个时间累加。
**任务cpu 占用率 = 任务运行时间的累加和 / 系统运行时间。 **

系统源码，`task.c` 文件的 `vTaskGetRunTimeStats` 函数，对应的实现如下：

```
ulStatsAsPercentage = pxTaskStatusArray[ x ].ulRunTimeCounter / ulTotalTime;
```

2.4.2 实现方式

任务时间计算器

我们需要一个独立于所有任务的计数器，用来精确计算每个任务的执行时间。由于系统调度的默认时间单位是1ms，有些简单的任务，还没执行够1ms，就阻塞，被切换了。所以，我们的计数器的时间单位，要远小于1ms。通常设置为系统调度时间的10-20 分之一。这里，我们设置为50us。

记录每个任务的执行的时间

在任务开始执行之前，记录下当前时间T0（ `ulTaskSwitchedInTime` ）。在任务被换出前，记录下当前时间T1（ `ulTotalRunTime` ）。当前任务运行的总的时间，就是T1 - T0。
在系统源码中， 任务上下文切换函数 `vTaskSwitchContext` 实现了这个处理。

```
2989 void vTaskSwitchContext( void )
2990 {
2991         if( uxSchedulerSuspended != ( UBaseType_t ) pdFALSE )
2992         {
2993                 /* The scheduler is currently suspended - do not allow a context
2994                 switch. */
2995                 xYieldPending = pdTRUE;
2996         }
2997         else
2998         {
2999                 xYieldPending = pdFALSE;
3000                 traceTASK_SWITCHED_OUT();
3001
3002                 #if ( configGENERATE_RUN_TIME_STATS == 1 )
3003                 {
3004                         #ifdef portALT_GET_RUN_TIME_COUNTER_VALUE
3005                                 portALT_GET_RUN_TIME_COUNTER_VALUE( ulTotalRunTime );
3006                         #else
3007                                 ulTotalRunTime = portGET_RUN_TIME_COUNTER_VALUE();
3008                         #endif
3009
3010                         /* Add the amount of time the task has been running to the
3011                         accumulated time so far.  The time the task started running was
3012                         stored in ulTaskSwitchedInTime.  Note that there is no overflow
3013                         protection here so count values are only valid until the timer
3014                         overflows.  The guard against negative values is to protect
3015                         against suspect run time stat counter implementations - which
3016                         are provided by the application, not the kernel. */
3017                         if( ulTotalRunTime > ulTaskSwitchedInTime )
3018                         {
3019                                 pxCurrentTCB->ulRunTimeCounter += ( ulTotalRunTime - ulTaskSwitchedInTime );
3020                         }
3021                         else
3022                         {
3023                                 mtCOVERAGE_TEST_MARKER();
3024                         }
3025                         ulTaskSwitchedInTime = ulTotalRunTime;
3026                 }
```

CSDN @gdut_llkkyy

2.5 在stm32 中的接口移植

增加全局的任务时间计算器

利用一个独立的定时器，产生一个周期为50us 的中断处理，在中断处理里，对任务时间进行累加。

```
long CPU_RunTime = 0;
void HAL_TIM_PeriodElapsedCallback(TIM_HandleTypeDef *htim)
{
    if ( htim == &htim16)
    {
      CPU_RunTime++;
    }
}
```

在系统接口中，返回 CPU_RunTime

```
extern long CPU_RunTime;
__weak unsigned long getRunTimeCounterValue(void)
{
  return CPU_RunTime;
}
```

源码接口

在 FreeRTOSConfig.h 的 portGET_RUN_TIME_COUNTER_VALUE 为系统源码的使用接口。

```
#define portGET_RUN_TIME_COUNTER_VALUE getRunTimeCounterValue
```

2.5 应用层调用

```
void task_debug_entry(void *parameter)
{
    task_debug_init();
```

```
    while (1)
    {
        os_delay_ms(6000);
        vTaskGetRunTimeStats(buf);
        app_log("\r\n taskName    runCount    usedRate\r\n");
        app_log("%s",buf);
        app_log("len:%d\n",strlen(buf));
    }
}
```

```
 taskName     runCount     usedRate
DebugTask       1548            <1%
IDLE            1165694         97%
```