LLOS 使用文档

目录

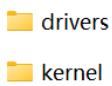
1. LLOS 介绍

LLOS 是一款轻量级的可以在支持 64bit 变量的 MCU 上运行的非实时资源管理系统，以事件驱动的方式实现线程管理，目的是取代裸机编程。系统内核提供简单的线程管理、任务间通信、软定时器、RTC、动态内存管理和设备驱动框架。另外截至目前，已经配备了了好用且功能强大的按键驱动、LED/GPIO 驱动模块、SSD1306 屏幕驱动模块。其它模块还在开发当中。

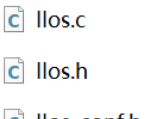
2. 版权声明

3. 代码移植

3.1 目录结构

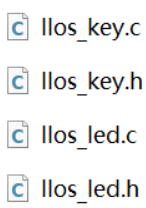


LLOS/kernel 路径下存放了 OS 的内核，其中有如下三个文件



对于内核代码，开发人员只需要配置 llos_conf.h 即可，除非您对代码非常熟悉，否则请不要修改其它文件。

LLOS/drivers 路径中存放了如按键和 LED/GPIO 等的驱动模块。这些模块的使用会在后面的章节进行介绍。



3.2 移植 OS

对于 OS 的运行，kernel 路径下的三个文件是必须的，在您的 IDE 中添加相关文件和头文件路径后，在您的代码中进行以下步骤的操作，在此之前注意将您的 IDE 编码格式设置为 UTF-8，否则可能出现中文注释乱码的情况。

类型声明：

```
typedef volatile uint32_t   ll_IO_t;

typedef uint8_t             ll_err_t;
typedef uint8_t             ll_taskId_t;
typedef uint16_t            ll_taskEvent_t;
typedef uint32_t            ll_tick_t;

typedef enum
{
    ll_reset,
    ll_set = !ll_reset,
}ll_bit_t;
typedef enum
{
    ll_disable,
    ll_enable = !ll_disable,
}ll_newState_t;
typedef enum
{
    ll_success,
    ll_fail = !ll_success,
}ll_result_t;
```

回调函数原型：

```
typedef void                (*ll_hwTimerInit_t)(void);
typedef void                (*ll_userDelay_t)(uint32_t time);
```

1）在初始化时调用 LLOS_Init();

```
/*=============================================================================
 * 函数名: LLOS_Init
 * 描述: 初始化
 * 参数:
 *      hwTimerInit: OS所使用的硬件定时器的初始化回调函数
 *      userDelayMs: 延时ms函数地址
 *      userDelayUs: 延时us函数地址
 =============================================================================*/
void LLOS_Init(ll_hwTimerInit_t hwTimerInit, ll_userDelay_t userDelayMs, ll_userDelay_t userDelayUs);
```

2）在 while(1)中调用调用 LLOS_Loop();

```
/*=============================================================================
 * 函数名: LLOS_Loop
 * 描述: OS处理函数，在死循环执行
 =============================================================================*/
void LLOS_Loop(void);
```

3）在定时器中断函数中调用 LLOS_Tick_Increase(x)为 OS 提供时钟，可以使用 MCU 的滴答定时器等提供时钟；

```
/*=============================================================================================
* 函数名: LLOS_Tick_Increase
* 描述: OS节拍计数，在定时器中断中调用，推荐1ms调用一次
* 参数:
*       ms: 以ms为单位的定时器中断时间
=============================================================================================*/
void LLOS_Tick_Increase(uint8_t ms);
```

例如在使用 HAL 库开发 STM32 时

```
/**
  * @brief This function handles System tick timer.
  */
void SysTick_Handler(void)
{
  /* USER CODE BEGIN SysTick_IRQn 0 */

  /* USER CODE END SysTick_IRQn 0 */
  HAL_IncTick();
  /* USER CODE BEGIN SysTick_IRQn 1 */
    extern void LLOS_Tick_Increase(uint8_t ms);
    LLOS_Tick_Increase(1);
  /* USER CODE END SysTick_IRQn 1 */
}
```

编译通过后，至此祝贺你，您已经完成了对 LLOS 的移植！

3.3 配置 OS

打开 llos_conf.h，对以下参数进行配置

```
/* ============================[kernel]============================ */
#define LL_TASK_NUM          (5)     /* 最大支持255个任务 */
#define LL_TIMER_NUM         (3)     /* 最大支持255个定时器 */
#define LL_ALARM_NUM         (1)     /* 最大支持255个闹钟 */
#define LL_HEAP_SIZE         (4096)  /* 用于ll_malloc的内存大小，要求4字节对齐 */
#define LL_DEV_MAX_NUM       (5)     /* 最大支持255个设备 */
```

数值过大会占用更多的 RAM，为避免资源浪费，请按照需求进行配置。

3.4 LOG 日志输出

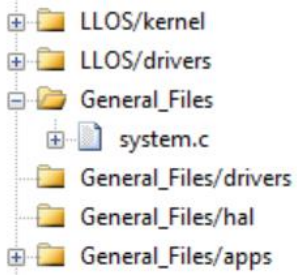
```
#define LOG_LEVEL           (4)     /* 打印日志等级 */

#if(LOG_LEVEL > 3)
#ifndef LOG_I
#define LOG_I(x...)         printf(x)
#endif
#endif

#if(LOG_LEVEL > 2)
#ifndef LOG_D
#define LOG_D(x...)         printf("[LLOS DEBUG] "x)
#endif
#endif

#if(LOG_LEVEL > 1)
#ifndef LOG_W
#define LOG_W(x...)         printf("[LLOS WARNING] "x)
#endif
#endif

#if(LOG_LEVEL > 0)
#ifndef LOG_E
#define LOG_E(x...)         printf("[LLOS ERROR] "x)
#endif
#endif
```

您可以在此修改 LOG 打印等级以及进行重映射。至此，您可以愉快的使用 LLOS 了！具体的使用方法可以参考

1） 头文件注释及源代码；
2） 例程 demo；
3） 哔哩哔哩（小叶子的技术世界）视频教程；

4. 示例代码框架介绍

本项目所有的 demo 例程都按照以下格式进行组织，



1） system.c 用于降低用户代码与 main.c 的耦合，以及存放一些公共代码，如串口驱动，重映射等；
2） drivers 目录用于存放硬件层的驱动程序；
3） hal 目录下的文件用于软硬件的衔接；

4） apps 目录用于存放用户的应用程序；

祝您使用愉快！

5. 修订日志