LLOS 使用文档

目录

1. LLOS 介绍 2
2. 版权声明 2
3. 代码移植 2
    3.1 目录结构 2
    3.2 移植 OS 2
4. 示例代码框架介绍 3
5. 修订日志 4

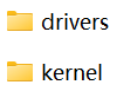
1. LLOS 介绍

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，以事件驱动和状态机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、设备驱动框架和指令解析功能。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、SSD1306 屏幕驱动、低阻塞的 DS18B20 驱动、CRC、环形 FIFO 等模块。其它模块还在不断地开发当中。

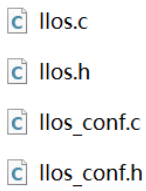
2. 版权声明

3. 代码移植

3.1 目录结构



LLOS/kernel 路径下存放了 OS 的内核，其中有如下四个文件



对于内核代码，开发人员只需要配置 llos_conf.h 和 llos_conf.c 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 路径中存放了其它驱动模块。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 路径下的四个文件是必须的，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。

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
 *      system_reset_hook: 系统复位函数地址
 *      userDelayMs: 延时ms函数地址
 *      userDelayUs: 延时us函数地址
 ============================================================================*/
typedef void                (*ll_hwTimerInit_hook_t)(void);
typedef void                (*ll_system_reset_hook_t)(void);
typedef void                (*ll_userDelay_hook_t)(uint32_t time);
void LLOS_Init(ll_hwTimerInit_hook_t hwTimerInit, ll_system_reset_hook_t system_reset_hook, ll_userDelay_hook_t userDelayMs, ll_userDelay_hook_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！具体的使用方法可以参考

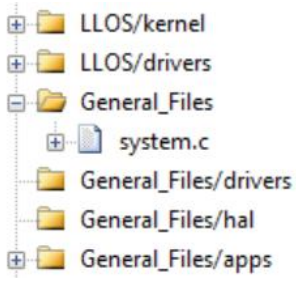
1） 头文件注释及源代码；

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；

5. 修订日志

注：版本号第一位更改表示架构级修改，第二位表示 API 不兼容修改，第三位 bug 修改或优化。

- 2025/02/25
  LLOS kernel 以及 drivers 由静态分配内存改为内存池分配内存。
- 2025/03/15
  按键驱动初始化 API 参数顺序调整。
  LED 驱动 API 调整，初始化方式调整，增加极性选择功能（可以选择各个 LED 高电平有效或低电平有效）。
- 2025/06/17 -> V2.1.0
  DS18B20 代码调整；
  LOG 打印宏增加 LL_前缀；
  增加一些 API；
- 2025/08/25 -> V2.1.2
  优化调度核心代码，减少任务内存占用情况；
- 2025/08/27 -> V2.1.3
  优化调度核心代码，减少任务内存占用情况，每个任务相较于 V2.1.0 节省 32 字节的内存占用以及一些 ROM 的占用；V2.1.2 有严重 Bug 请不要使用！