title: 基于Linux的kfifo移植到STM32（支持os的互斥访问） author: 杰杰 top: false cover: false toc: true mathjax: false date: 2019-10-06 22:07:14 img: coverImg: password: summary: tags:

- Cortex-M
- STM32
- kfifo categories: STM32

基于Linux的kfifo移植到STM32（支持os的互斥访问）

关于kfifo

kfifo是内核里面的一个First In First Out数据结构，它采用环形循环队列的数据结构来实现；它提供一个无边界的字节流服务，最重要的一点是，它使用并行无锁编程技术，即当它用于只有一个入队线程和一个出队线程的场情时，两个线程可以并发操作，而不需要任何加锁行为，就可以保证kfifo的线程安全。

> 具体什么是环形缓冲区，请看我以前的文章

说明

> 关于kfifo的相关概念我不会介绍，有兴趣可以看他的相关文档，我只将其实现过程移植重写，移植到适用stm32开发板上，并且按照我个人习惯重新命名，**RingBuff**->意为环形缓冲区

RingBuff_t

环形缓冲区的结构体成员变量，具体含义看注释。 buffer: 用于存放数据的缓存 size: buffer空间的大小 in, out: 和buffer一起构成一个循环队列。 in指向buffer中队头，而且out指向buffer中的队尾

```
typedef struct ringbuff
{
        uint8_t *buffer;         /* 数据区域 */
        uint32_t size;      /* 环形缓冲区大小 */
        uint32_t in;        /* 数据入队指针 (in % size) */
        uint32_t out;       /* 数据出队指针 (out % size) */
#if USE_MUTEX
        MUTEX_T *mutex;       /* 支持rtos的互斥 */
#endif
}RingBuff_t ;
```

Create_RingBuff

创建一个环形缓冲区，为了适应后续对缓冲区入队出队的高效操作，环形缓冲区的大小应为2^n字节， 如果不是这个大小，则系统默认裁剪以对应缓冲区字节。 当然还可以优化，不过我目前并未做，思路如下：如果系统支持动态分配内存，则向上对齐，避免浪费内存空间，否则就按照我默认的向下对齐，当内存越大，对齐导致内存泄漏则会越多。对齐采用的函数是`roundup_pow_of_two`。如果系统支持互斥量，那么还将创建一个互斥量用来做互斥访问，防止多线程同时使用导致数据丢失。

```
/************************************************************
  * @brief   Create_RingBuff
  * @param   rb：环形缓冲区句柄
  *          buffer：环形缓冲区的数据区域
  *          size：环形缓冲区的大小，缓冲区大小要为2^n
  * @return  err_t：ERR_OK表示创建成功，其他表示失败
  * @author  jiejie
  * @github  https://github.com/jiejieTop
  * @date    2018-xx-xx
  * @version v1.0
  * @note    用于创建一个环形缓冲区
  ************************************************************/
err_t Create_RingBuff(RingBuff_t* rb,
                      uint8_t *buffer,
                      uint32_t size
                                                              )
{
        if((rb == NULL)||(buffer == NULL)||(size == 0))
        {
                PRINT_ERR("data is null!");
                return ERR_NULL;
        }

        PRINT_DEBUG("ringbuff size is %d!",size);
        /* 缓冲区大小必须为2^n字节,系统会强制转换,
                 否则可能会导致指针访问非法地址。
                 空间大小越大,强转时丢失内存越多 */
        if(size&(size - 1))
        {
                size = roundup_pow_of_two(size);
                PRINT_DEBUG("change ringbuff size is %d!",size);
        }

        rb->buffer = buffer;
        rb->size = size;
        rb->in = rb->out = 0;
#if USE_MUTEX
  /* 创建信号量不成功 */
  if(!create_mutex(rb->mutex))
  {
    PRINT_ERR("create mutex fail!");
    ASSERT(ASSERT_ERR);
    return ERR_NOK;
  }
#endif
        PRINT_DEBUG("create ringBuff ok!");
        return ERR_OK;
}
```

roundup_pow_of_two

```
/************************************************************
  * @brief   roundup_pow_of_two
  * @param   size：传递进来的数据长度
  * @return  size：返回处理之后的数据长度
  * @author  jiejie
  * @github  https://github.com/jiejieTop
  * @date    2018-xx-xx
  * @version v1.0
  * @note    用于处理数据，使数据长度必须为 2^n
        *                                          如果不是，则转换，丢弃多余部分，如
        *                                          roundup_pow_of_two(66) -> 返回 64
  ************************************************************/
static unsigned long roundup_pow_of_two(unsigned long x)
{
        return (1 << (fls(x-1)-1));                          //向下对齐
  //return (1UL << fls(x - 1));                 //向上对齐，用动态内存可用使用
}
```

Delete_RingBuff

删除一个环形缓冲区，删除之后，缓冲区真正存储地址是不会被改变的（目前我是使用自定义数组做缓冲区的），但是删除之后，就无法对缓冲区进行读写操作。并且如果支持os的话，创建的互斥量会被删除。

```
/************************************************************
  * @brief   Delete_RingBuff
  * @param   rb：环形缓冲区句柄
  * @return  err_t：ERR_OK表示成功，其他表示失败
  * @author  jiejie
  * @github  https://github.com/jiejieTop
  * @date    2018-xx-xx
  * @version v1.0
  * @note    删除一个环形缓冲区
  ************************************************************/
err_t Delete_RingBuff(RingBuff_t *rb)
{
        if(rb == NULL)
        {
                PRINT_ERR("ringbuff is null!");
                return ERR_NULL;
        }

        rb->buffer = NULL;
        rb->size = 0;
        rb->in = rb->out = 0;
#if USE_MUTEX
  if(!deleta_mutex(rb->mutex))
  {
    PRINT_DEBUG("deleta mutex is fail!");
    return ERR_NOK;
  }
```

```
#endif
        return ERR_OK;
}
```

Write_RingBuff

向环形缓冲区写入指定数据，支持线程互斥访问。用户想要写入缓冲区的数据长度不一定是真正入队的长度，在完成的时候还要看看返回值是否与用户需要的长度一致~ 这个函数很有意思，也是比较高效的入队操作，将指定区域的数据拷贝到指定的缓冲区中，过程看注释即可

```
/**************************************************************
  * @brief   Write_RingBuff
  * @param   rb:环形缓冲区句柄
  * @param   wbuff:写入的数据起始地址
  * @param   len:写入数据的长度(字节)
  * @return  len:实际写入数据的长度(字节)
  * @author  jiejie
  * @github  https://github.com/jiejieTop
  * @date    2018-xx-xx
  * @version v1.0
  * @note    这个函数会从buff空间拷贝len字节长度的数据到
             rb环形缓冲区中的空闲空间。
  **************************************************************/
uint32_t Write_RingBuff(RingBuff_t *rb,
                        uint8_t *wbuff,
                        uint32_t len)
{
  uint32_t l;
#if USE_MUTEX
  /* 请求互斥量，成功才能进行ringbuff的访问 */
  if(!request_mutex(rb->mutex))
  {
    PRINT_DEBUG("request mutex fail!");
    return 0;
  }
  else  /* 获取互斥量成功 */
  {
#endif
    len = min(len, rb->size - rb->in + rb->out);

    /* 第一部分的拷贝:从环形缓冲区写入数据直至缓冲区最后一个地址 */
    l = min(len, rb->size - (rb->in & (rb->size - 1)));
    memcpy(rb->buffer + (rb->in & (rb->size - 1)), wbuff, l);

    /* 如果溢出则在缓冲区头写入剩余的部分
       如果没溢出这句代码相当于无效 */
    memcpy(rb->buffer, wbuff + l, len - l);

    rb->in += len;

    PRINT_DEBUG("write ringBuff len is %d!",len);
```

```
#if USE_MUTEX
  }
  /* 释放互斥量 */
  release_mutex(rb->mutex);
#endif
  return len;
}
```

Read_RingBuff

读取缓冲区数据到指定区域，用户指定读取长度，用户想要读取的长度不一定是真正读取的长度，在读取完成的时候还要看看返回值是否与用户需要的长度一致~也支持多线程互斥访问。 也是缓冲区出队的高效操作。过程看代码注释即可

```
/**********************************************************
  * @brief   Read_RingBuff
  * @param   rb:环形缓冲区句柄
  * @param   wbuff:读取数据保存的起始地址
  * @param   len:想要读取数据的长度(字节)
  * @return  len:实际读取数据的长度(字节)
  * @author  jiejie
  * @github  https://github.com/jiejieTop
  * @date    2018-xx-xx
  * @version v1.0
  * @note    这个函数会从rb环形缓冲区中的数据区域拷贝len字节
             长度的数据到rbuff空间。
  **********************************************************/
uint32_t Read_RingBuff(RingBuff_t *rb,
                       uint8_t *rbuff,
                       uint32_t len)
{
  uint32_t l;
#if USE_MUTEX
  /* 请求互斥量，成功才能进行ringbuff的访问 */
  if(!request_mutex(rb->mutex))
  {
    PRINT_DEBUG("request mutex fail!");
    return 0;
  }
  else
  {
#endif
    len = min(len, rb->in - rb->out);

    /* 第一部分的拷贝:从环形缓冲区读取数据直至缓冲区最后一个 */
    l = min(len, rb->size - (rb->out & (rb->size - 1)));
    memcpy(rbuff, rb->buffer + (rb->out & (rb->size - 1)), l);

    /* 如果溢出则在缓冲区头读取剩余的部分
       如果没溢出这句代码相当于无效 */
    memcpy(rbuff + l, rb->buffer, len - l);
```

```
    rb->out += len;

    PRINT_DEBUG("read ringBuff len is %d!",len);
#if USE_MUTEX
  }
  /* 释放互斥量 */
  release_mutex(rb->mutex);
#endif
  return len;
}
```

获取缓冲区信息

这些就比较简单了，看看缓冲区可读可写的数据有多少

```
/************************************************************
  * @brief   CanRead_RingBuff
        * @param   rb:环形缓冲区句柄
        * @return  uint32:可读数据长度 0 / len
  * @author  jiejie
  * @github  https://github.com/jiejieTop
  * @date    2018-xx-xx
  * @version v1.0
  * @note    可读数据长度
  ************************************************************/
uint32_t CanRead_RingBuff(RingBuff_t *rb)
{
        if(NULL == rb)
        {
                PRINT_ERR("ringbuff is null!");
                return 0;
        }
        if(rb->in == rb->out)
                return 0;

        if(rb->in > rb->out)
                return (rb->in - rb->out);

        return (rb->size - (rb->out - rb->in));
}

/************************************************************
  * @brief   CanRead_RingBuff
        * @param   rb:环形缓冲区句柄
        * @return  uint32:可写数据长度 0 / len
  * @author  jiejie
  * @github  https://github.com/jiejieTop
  * @date    2018-xx-xx
  * @version v1.0
  * @note    可写数据长度
```

```
  ***********************************************************/
uint32_t CanWrite_RingBuff(RingBuff_t *rb)
{
        if(NULL == rb)
        {
                PRINT_ERR("ringbuff is null!");
                return 0;
        }

        return (rb->size - CanRead_RingBuff(rb));
}
```

附带

这里的代码我是用于测试的，随便写的

```
        RingBuff_t ringbuff_handle;

        uint8_t rb[64];
        uint8_t res[64];
        Create_RingBuff(&ringbuff_handle,
                                                                rb,
                                                                sizeof(rb));
                        Write_RingBuff(&ringbuff_handle,
                    res,
                    datapack.data_length);

                        PRINT_DEBUG("CanRead_RingBuff =
%d!",CanRead_RingBuff(&ringbuff_handle));
                        PRINT_DEBUG("CanWrite_RingBuff =
%d!",CanWrite_RingBuff(&ringbuff_handle));

                        Read_RingBuff(&ringbuff_handle,
                    res,
                    datapack.data_length);
```

支持多个os的互斥量操作

此处模仿了文件系统的互斥操作

```
#if USE_MUTEX
#define  MUTEX_TIMEOUT   1000     /* 超时时间 */
#define  MUTEX_T         mutex_t  /* 互斥量控制块 */
#endif

/********************************** mutex
***********************************************/
#if USE_MUTEX
/*************************************************************
```

```
  * @brief   create_mutex
  * @param   mutex:创建信号量句柄
  * @return  创建成功为1，0为不成功。
  * @author  jiejie
  * @github  https://github.com/jiejieTop
  * @date    2018-xx-xx
  * @version v1.0
  * @note    创建一个互斥量,用户在os中互斥使用ringbuff，
  *          支持的os有rtt、win32、ucos、FreeRTOS、LiteOS
  ************************************************************/
static err_t create_mutex(MUTEX_T *mutex)
{
  err_t ret = 0;

//      *mutex = rt_mutex_create("test_mux",RT_IPC_FLAG_PRIO); /* rtt */
//      ret = (err_t)(*mutex != RT_NULL);

//      *mutex = CreateMutex(NULL, FALSE, NULL);                /* Win32 */
//      ret = (err_t)(*mutex != INVALID_HANDLE_VALUE);

//      *mutex = OSMutexCreate(0, &err);                /* uC/OS-II */
//      ret = (err_t)(err == OS_NO_ERR);

//      *mutex = xSemaphoreCreateMutex();       /* FreeRTOS */
//      ret = (err_t)(*mutex != NULL);

//  ret = LOS_MuxCreate(&mutex);  /* LiteOS */
//      ret = (err_t)(ret != LOS_OK);
  return ret;
}
/************************************************************
  * @brief   deleta_mutex
  * @param   mutex:互斥量句柄
  * @return  NULL
  * @author  jiejie
  * @github  https://github.com/jiejieTop
  * @date    2018-xx-xx
  * @version v1.0
  * @note    删除一个互斥量，支持的os有rtt、win32、ucos、FreeRTOS、LiteOS
  ************************************************************/
static err_t deleta_mutex(MUTEX_T *mutex)
{
        err_t ret;

//      ret = rt_mutex_delete(mutex);   /* rtt */

//      ret = CloseHandle(mutex);       /* Win32 */

//      OSMutexDel(mutex, OS_DEL_ALWAYS, &err); /* uC/OS-II */
//      ret = (err_t)(err == OS_NO_ERR);

//  vSemaphoreDelete(mutex);            /* FreeRTOS */
//      ret = 1;
```

```
//  ret = LOS_MuxDelete(&mutex);  /* LiteOS */
//      ret = (err_t)(ret != LOS_OK);

        return ret;
}
/***********************************************************
  * @brief   request_mutex
  * @param   mutex:互斥量句柄
  * @return  NULL
  * @author  jiejie
  * @github  https://github.com/jiejieTop
  * @date    2018-xx-xx
  * @version v1.0
  * @note    请求一个互斥量，得到互斥量的线程才允许进行访问缓冲区
  *          支持的os有rtt、win32、ucos、FreeRTOS、LiteOS
  ***********************************************************/
static err_t request_mutex(MUTEX_T *mutex)
{
        err_t ret;

//      ret = (err_t)(rt_mutex_take(mutex, MUTEX_TIMEOUT) == RT_EOK);/* rtt */

//      ret = (err_t)(WaitForSingleObject(mutex, MUTEX_TIMEOUT) == WAIT_OBJECT_0);
/* Win32 */

//      OSMutexPend(mutex, MUTEX_TIMEOUT, &err));               /* uC/OS-II */
//      ret = (err_t)(err == OS_NO_ERR);

//      ret = (err_t)(xSemaphoreTake(mutex, MUTEX_TIMEOUT) == pdTRUE);  /*
FreeRTOS */

//  ret = (err_t)(LOS_MuxPend(mutex,MUTEX_TIMEOUT) == LOS_OK);                  /*
LiteOS */

        return ret;
}
/***********************************************************
  * @brief   release_mutex
  * @param   mutex:互斥量句柄
  * @return  NULL
  * @author  jiejie
  * @github  https://github.com/jiejieTop
  * @date    2018-xx-xx
  * @version v1.0
  * @note    释放互斥量，当线程使用完资源必须释放互斥量
  *          支持的os有rtt、win32、ucos、FreeRTOS、LiteOS
  ***********************************************************/
static void release_mutex(MUTEX_T *mutex)
{
//      rt_mutex_release(mutex);/* rtt */

//      ReleaseMutex(mutex);            /* Win32 */

//      OSMutexPost(mutex);             /* uC/OS-II */
```

```
//      xSemaphoreGive(mutex);  /* FreeRTOS */

//  LOS_MuxPost(mutex);   /* LiteOS */
}
#endif
/********************************* mutex
************************************************/
```

debug.h

最后送一份debug的简便操作源码，因为前文很多时候会调用 `PRINT_ERR` `PRINT_DEBUG`

```
#ifndef _DEBUG_H
#define _DEBUG_H
/************************************************************
  * @brief   debug.h
  * @author  jiejie
  * @github  https://github.com/jiejieTop
  * @date    2018-xx-xx
  * @version v1.0
  * @note    此文件用于打印日志信息
  ************************************************************/
/**
* @name Debug print
* @{
*/
#define PRINT_DEBUG_ENABLE              1               /* 打印调试信息 */
#define PRINT_ERR_ENABLE                        1       /* 打印错误信息 */
#define PRINT_INFO_ENABLE                       0               /* 打印个人信息 */


#if PRINT_DEBUG_ENABLE
#define PRINT_DEBUG(fmt, args...)        do{(printf("\n[DEBUG] >> "), printf(fmt,
##args));}while(0)
#else
#define PRINT_DEBUG(fmt, args...)
#endif

#if PRINT_ERR_ENABLE
#define PRINT_ERR(fmt, args...)          do{(printf("\n[ERR] >> "), printf(fmt,
##args));}while(0)
#else
#define PRINT_ERR(fmt, args...)
#endif

#if PRINT_INFO_ENABLE
#define PRINT_INFO(fmt, args...)         do{(printf("\n[INFO] >> "), printf(fmt,
##args));}while(0)
#else
#define PRINT_INFO(fmt, args...)
```

```
#endif

/**@} */

//针对不同的编译器调用不同的stdint.h文件
#if defined(__ICCARM__) || defined(__CC_ARM) || defined(__GNUC__)
    #include <stdint.h>
#endif

/* 断言 Assert */
#define AssertCalled(char,int)  printf("\nError:%s,%d\r\n",char,int)
#define ASSERT(x)   if((x)==0)  AssertCalled(__FILE__,__LINE__)

typedef enum
{
        ASSERT_ERR = 0,                                                 /*
错误 */
        ASSERT_SUCCESS = !ASSERT_ERR    /* 正确 */
} Assert_ErrorStatus;

typedef enum
{
        FALSE = 0,              /* 假 */
        TRUE = !FALSE   /* 真 */
}ResultStatus;

#endif /* __DEBUG_H */
```

喜欢就关注我吧！

相关代码可以在公众号后台获取。