内存环形缓冲区 ringbuffer

RealTouch 评估板 RT-Thread 入门文档

版本号：1.0.0

日期：2012/8/14

修订记录

日期	作者	修订历史
2012/8/14	bloom5	创建文档

实验目的

- 快速了解 ringbuffer 相关背景知识
- 掌握了解 ringbuffer 相关 API

硬件说明

本实验使用 RT-Thread 官方的 Realtouch 开发板作为实验平台。涉及到的硬件主要为

- 串口 3，作为 rt_kprintf 输出，需要连接 JTAG 扩展板

具体请参见《Realtouch 开发板使用手册》

实验原理及程序结构

Ringbuffer 的数据结构

```
struct rt_ringbuffer
{
    rt_uint16_t read_index, write_index;
    rt_uint8_t *buffer_ptr;
    rt_uint16_t buffer_size;
};
```

环形 Buffer 的特点：

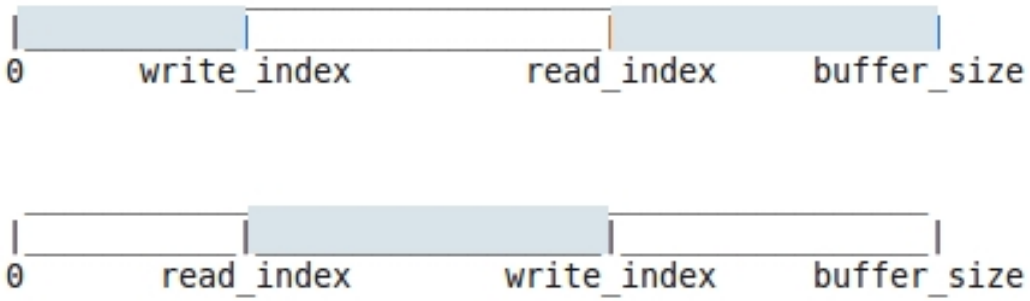
通常包含一个读指针（read_index）和一个写指针（write_index）。读指针指向环形 Buffer 中第一个可读的数据，写指针指向环形 Buffer 中第一个可写的缓冲区。通过移动读指针和写指针就可以实现 Buffer 的数据读取和写入。在通常情况下，环形 Buffer 的读用户仅仅会影响读指针，而写用户也仅仅会影响写指针。

环形 Buffer 的原理：首先在内存里开辟一片区域（大小为 buffer_size），对于写用户，顺序往 Buffer 里写入东西，直到写满为止；对于读用户，顺次从 Buffer 里读出东西，直到读空为止。

有效存储空间与 buffer_size 的区别：有效存储空间是指那些没有存放数据，或者以前存放过但已经处理过的数据，就是可用的空间大小；而 buffer_size 指的是总大小。

通过上面介绍可知，环形 Buffer 在物理上仍然是一块连续的内存 Buffer，只不过其空间会被循环使用而已。示意图如下，根据读写指针的

位置可分为两种情况，其中阴影填充部分为数据/已用空间，空白区域为可用空间，即有效存储空间。



Ringbuffer 示意图

实验设计

本实验的主要设计目的是帮助读者了解环形缓冲区内容，熟悉相关 API 调用方法。请读者注意，本实验本身不具有实际的工程参考价值，只是帮助读者快速了解相关 API 的用法。

源程序说明

本实验对应 1_kernel_ringbuffer

系统依赖

在 rtconfig.h 中需要开启

- #define RT_USING_HEAP

此项可选，开启此项可以创建动态线程和动态信号量，如果使用静态线程和 静态信号量，则此项不是必要的

- #define RT_USING_CONSOLE

此项必须，本实验使用 rt_kprintf 向串口打印按键信息，因此需要开启此项

- #define RT_USING_DEVICE_IPC

此项必须，本实验使用 ringbuffer 相关内容，只有将上述宏打开，编译器才会将 ringbuffer.c 文件编译，当然如果使用的是 keil，可以手动加入。

主程序说明

首先是我们的全局变量：

全局变量

```
static rt_sem_t sem = RT_NULL;

static rt_uint8_t working_buffer[256];
struct rt_ringbuffer rb;

/* 指向线程控制块的指针 */
static rt_thread_t tid1 = RT_NULL;
static rt_thread_t tid2 = RT_NULL;
```

我们为环形缓冲区初始化了 256 个字节，然后操作读写线程。

ringbuffer 初始化

```
rt_ringbuffer_init(&rb,working_buffer,256);
```

线程入口程序

```
static void thread1_entry(void* parameter)
{
    rt_bool_t result;
    rt_uint8_t data_buffer[33];
    rt_uint32_t i = 1;

    while(i--)
    {
        rt_sem_take (sem,RT_WAITING_FOREVER);
        result = rt_ringbuffer_get(&rb,&data_buffer[0],33);
        rt_sem_release(sem);

        rt_kprintf("%s\n",data_buffer);

        rt_thread_delay(5);
    }

}

static void thread2_entry(void *parameter)
{
    rt_bool_t result;
    rt_uint32_t index,setchar,i = 1;
    rt_uint8_t data_buffer[33];

    setchar = 0x21;
    while(i--)
    {
        for (index = 0; index < 32; index++)
        {
```

```
            data_buffer[index] = setchar;
            if (++setchar == 0x7f)
            {
                setchar = 0x21;
            }
        }
        data_buffer[32] = '\0';

        rt_sem_take(sem,RT_WAITING_FOREVER);

        result = rt_ringbuffer_put(&rb,&data_buffer[0],33);
        rt_kprintf("write buffer success!\n");

        rt_sem_release(sem);

        rt_thread_delay(10);
    }
}
```

编译调试及观察输出信息

编译请参见《RT-Thread 配置开发环境指南》完成编译烧录，参考《Realtouch 开发板使用手册》完成硬件连接，连接扩展板上的串口和 jlink。

运行后可以看到如下信息：

```
 \ | /
- RT -     Thread Operating System
 / | \     1.1.0 build Aug 13 2012
 2006 - 2012 Copyright by rt-thread team
write buffer success!
!"#$%&'()*+,-./0123456789:;<=>?@
```

结果分析

本例程主要想要体现的就是环形缓冲区的读写操作，需要两个线程操作才能完成，其中一个线程向缓冲区内写数据，另一个从缓冲区内读取数据。当从缓冲区中读到数据时将它打印出来。

写环形缓冲区

```
rt_ringbuffer_put(&rb,&data_buffer[0],33);
```

读环形缓冲区

```
rt_ringbuffer_get(&rb,&data_buffer[0],33);
```

关于上面两个函数需要注意一点的是，在编程指南上旧版的函数返回值是 bool 型的，而在新的 ringbuffer.c 中定义的函数的返回值均是 size_t。

关于环形缓冲区的进一步操作，可以通过查阅相关资料，进行更深层次的探索。