邮箱基本使用

RealTouch 评估板 RT-Thread 入门文档

版本号：1.0.0

日期：2012/8/12

修订记录

日期	作者	修订历史
2012/8/12	prife	创建文档

实验目的

- 了解邮箱的基本使用
- 熟练使用邮箱相关 API 实现多个线程间通信

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

实验设计

本实验的主要设计目的是帮助读者快速了解邮箱相关 API，包括静态邮箱初始化/脱离，发送/接受邮件。本实验中使用静态邮箱控制块，动态邮箱的使用就交给读者测试。

本实验中创建一个邮箱，两个线程，线程 2 以一定周期发送邮箱，线程 1 以一定周期从邮箱中取出邮件。当线程 2 中发送 20 封邮件之后，发送一封特殊的邮件通知其他线程，自己已经运行结束。线程 1 取出邮件后，检查邮件是否是特殊邮件，如果是，则线程 1 也退出。

源程序说明

本实验对应 kernel_mailbox_basic。

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邮箱，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_MAILBOX

此项必须，开启此项后才可以使用邮箱相关 API。

- #define RT_USING_CONSOLE
  此项必须，本实验使用 rt_kpriintf 向串口打印按键信息，因此需要开启此项

主程序说明

在 applications/application.c 中定义静态邮箱控制块、存放邮件的缓冲区，以及一些字符串用来作为邮件。如下所示

定义全局变量代码

```
/* 邮箱控制块 */
static struct rt_mailbox mb;
/* 用于放邮件的内存池 */
static char mb_pool[128];

static char mb_str1[] = "I'm a mail!";
static char mb_str2[] = "this is another mail!";
static char mb_str3[] = "over";
```

在 applications/application.c 中的 int rt_application_init()函数中，初始化邮箱。

初始化邮箱代码

```
    rt_err_t result;

    /* 初始化一个 mailbox */
    result = rt_mb_init(&mb,
        "mbt",           /* 名称是 mbt */
        &mb_pool[0],      /* 邮箱用到的内存池是 mb_pool */
        sizeof(mb_pool)/4, /* 邮箱中的邮件数目，因为一封邮件占 4 字节 */
        RT_IPC_FLAG_FIFO); /* 采用 FIFO 方式进行线程等待 */
    if (result != RT_EOK)
    {
        rt_kprintf("init mailbox failed.\n");
        return -1;
}
```

在 int rt_application_init()初始化名为”thread1”的 thread1 的静态线程，如下所示。

初始化线程 1 代码

```
    rt_thread_init(&thread1,
                   "thread1",
                   thread1_entry,
                   RT_NULL,
                   &thread1_stack[0],
                   sizeof(thread1_stack),10,5);
    rt_thread_startup(&thread1);
```

其线程入口函数如下所示，线程 1 以 10 个 tick 的间隔不停地从邮箱中取出邮件，并打印每每封邮件的内容，并对每封邮件进行检查。当检测邮件为特殊的邮件（mb_str3）这表明这是线程 2 发送的最后一封邮件，则线程 1 不再循环接收邮件，从 while 循环中调出，线程函数运行结束。

线程 1 代码

```
 ALIGN(RT_ALIGN_SIZE)        //设置下一句线程栈数组为对齐地址
 static char thread1_stack[1024];  //设置线程堆栈为 1024Bytes
 struct rt_thread thread1;       //定义静态线程数据结构
 /* 线程 1 入口 */
 static void thread1_entry(void* parameter)
 {
    char* str;

    while (1)
    {
       rt_kprintf("thread1: try to recv a mail\n");

       /* 从邮箱中收取邮件 */
       if (rt_mb_recv(&mb, (rt_uint32_t*)&str, RT_WAITING_FOREVER)
== RT_EOK)
       {
          rt_kprintf("thread1: get a mail from mailbox, the
content:%s\n", str);
          if (str == mb_str3)
             break;

          /* 延时 10 个 OS Tick */
```

```
            rt_thread_delay(10);
        }
    }
    /* 执行邮箱对象脱离 */
    rt_mb_detach(&mb);
}
```

在 int rt_application_init()初始化名为”thread2”的 thread2 的静态线程，如下所示。

初始化线程 2 代码

```
    rt_thread_init(&thread2,
                   "thread2",
                   thread2_entry,
                   RT_NULL,
                   &thread2_stack[0],
                   sizeof(thread2_stack),10,5);
rt_thread_startup(&thread2);
```

其线程入口函数如下所示，线程 2 以 20 个 tick 的间隔不停地向邮箱中发送邮件，并使用变量 count 进行计数，奇数次发送数组 mb_str1 首地址作为邮件，偶数次发送数组 mb_str2 首地址作为邮件，累计发送 20 封邮件后，将发送数组 mb_str3 首地址作为邮件发送，这是线程 2 发送的最后一封邮件，线程函数运行结束。

线程 2 代码

```
ALIGN(RT_ALIGN_SIZE)        //设置下一句线程栈数组为对齐地址
static char thread2_stack[1024];  //设置线程堆栈为 1024Bytes
struct rt_thread thread2;       //定义静态线程数据结构
/* 线程 2 入口 */
static void thread2_entry(void* parameter)
{
    rt_uint8_t count;

    count = 0;
    while (count < 10)
    {
        count ++;
```

```
        if (count & 0x1)
        {
            /* 发送 mb_str1 地址到邮箱中 */
            rt_mb_send(&mb, (rt_uint32_t)&mb_str1[0]);
        }
        else
        {
            /* 发送 mb_str2 地址到邮箱中 */
            rt_mb_send(&mb, (rt_uint32_t)&mb_str2[0]);
        }

        /* 延时 20 个 OS Tick */
        rt_thread_delay(20);
    }

    /* 发送邮件告诉线程 1，线程 2 已经运行结束 */
    rt_mb_send(&mb, (rt_uint32_t)&mb_str3[0]);
}
```

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。

运行后可在串口上看到如下信息：

串口输出

```
 \ | /
- RT -     Thread Operating System
 / | \     1.1.0 build Aug  9 2012
 2006 - 2012 Copyright by rt-thread team
thread1: try to recv a mail
thread1: get a mail from mailbox, the content:I'm a mail!
thread1: try to recv a mail
thread1: get a mail from mailbox, the content:this is another mail!
thread1: try to recv a mail
thread1: get a mail from mailbox, the content:I'm a mail!
thread1: try to recv a mail
```

```
thread1: get a mail from mailbox, the content:this is another mail!
thread1: try to recv a mail
thread1: get a mail from mailbox, the content:I'm a mail!
thread1: try to recv a mail
thread1: get a mail from mailbox, the content:this is another mail!
thread1: try to recv a mail
thread1: get a mail from mailbox, the content:I'm a mail!
thread1: try to recv a mail
thread1: get a mail from mailbox, the content:this is another mail!
thread1: try to recv a mail
thread1: get a mail from mailbox, the content:I'm a mail!
thread1: try to recv a mail
thread1: get a mail from mailbox, the content:this is another mail!
thread1: try to recv a mail
thread1: get a mail from mailbox, the content:over
```

结果分析

整个程序运行过程中各个线程的状态变化：

rt_application_init 中创建线程 thread1 和 thread2，两者具有相同的优先级，由于先使用 rt_thread_startup(&thread1)，故线程 1 优先运行，首先执行：

```
rt_kprintf("thread1: try to recv a mail\n");
```

无法确定此语句执行时间大致是多少个 tick，这个跟 RT_TICK_PERSECOND 的设定、系统的主频、以串口驱动的实现有关。不过庆幸的是，这些不确定因素并不影响程序最终的运行结果。

当线程 1 试图从邮箱中获取邮件时，如果邮箱中没有邮件，则线程 1 被挂起直到邮箱中填充了邮件。此时线程 2 就会运行，向邮件中发送邮件，之后休眠 20 个 tick，线程 2 挂起，线程 1 继续调度运行。线程 1 从邮箱中收到邮件后，打印邮件内容，检查邮件是否是最后一封，若是，则线程 1 跳出循环，脱离静态邮箱控制块，线程函数运行结束；若否，则线程 1 挂起 10 个 tick。如果此时线程 2 也在挂起，则内核执行 IDLE 线程。当线程 2 发送 20 封邮件后，发送最后一封邮件，线程函数处理结束。

在 IDLE 线程中将两个线程脱离。

总结

本实验演示了 RT-Thread 中邮箱/邮件作为多线程通信的用法，以静态邮箱控制块为例，动态邮箱的用法类似，只是创建/删除需要使用 rt_mb_create/rt_mb_delete 函数，读者可以使用动态邮箱重复本实验。