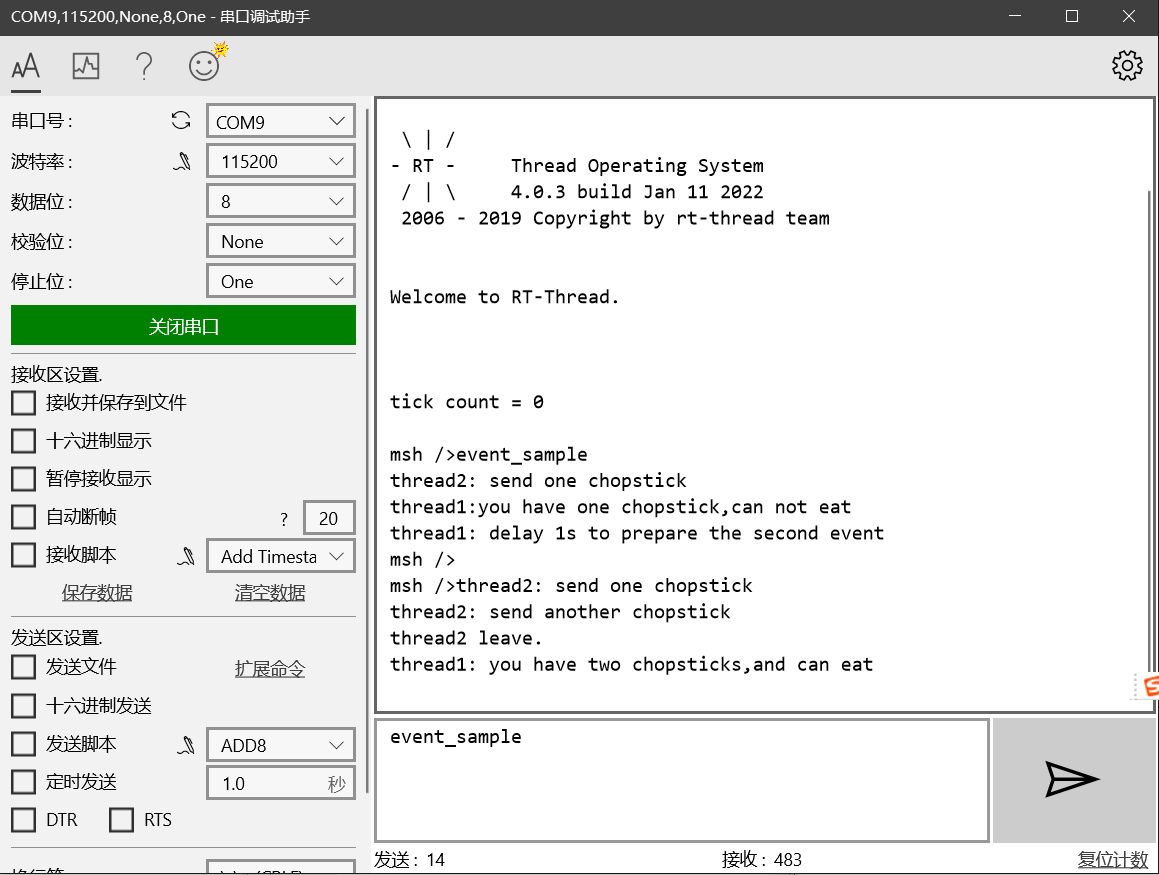
## 2.5 RT\_Thread操作系统开发

### 设计方案

通过进程和事件等操作，模拟哲学家进餐问题。

### 实现方法

输入event\_sample



不同的进程代表不同的哲学家，为他们分配不同的优先级，哲学家需要筷子才能够进餐，通过事件模拟哲学家获得筷子的情况，可以很好的实现哲学家进餐问题，避免死锁。

### 代码实现

#include <rtthread.h>

#define THREAD\_PRIORITY 9

#define THREAD\_TIMESLICE 5

#define EVENT\_FLAG3 (1 << 3)

#define EVENT\_FLAG5 (1 << 5)

#define EVENT\_FLAG6 (1 << 6)

#define EVENT\_FLAG7 (1 << 7)

/\* 事件控制块 \*/

static struct rt\_event event;

ALIGN(RT\_ALIGN\_SIZE)

static char thread1\_stack[1024];

static struct rt\_thread thread1;

/\* 线程1入口函数 \*/

static void thread1\_recv\_event(void \*param)

{

rt\_uint32\_t e;

/\* 第一次接收事件，事件3或事件5任意一个可以触发线程1，接收完后清除事件标志 \*/

if (rt\_event\_recv(&event, (EVENT\_FLAG3 | EVENT\_FLAG5),

RT\_EVENT\_FLAG\_OR | RT\_EVENT\_FLAG\_CLEAR,

RT\_WAITING\_FOREVER, &e) == RT\_EOK)

{

rt\_kprintf("thread1:you have one chopstick,can not eat\n", e);

}

rt\_kprintf("thread1: delay 1s to prepare the second event\n");

rt\_thread\_mdelay(1000);

/\* 第二次接收事件，事件3和事件5均发生时才可以触发线程1，接收完后清除事件标志 \*/

if (rt\_event\_recv(&event, (EVENT\_FLAG3 | EVENT\_FLAG5),

RT\_EVENT\_FLAG\_AND | RT\_EVENT\_FLAG\_CLEAR,

RT\_WAITING\_FOREVER, &e) == RT\_EOK)

{

rt\_kprintf("thread1: you have two chopsticks,and can eat\n", e);

}

/\* 第三次接收事件，事件6和事件7均发生时才可以触发线程1，接收完后清除事件标志 \*/

if (rt\_event\_recv(&event, (EVENT\_FLAG6 | EVENT\_FLAG7),

RT\_EVENT\_FLAG\_AND | RT\_EVENT\_FLAG\_CLEAR,

RT\_WAITING\_FOREVER, &e) == RT\_EOK)

{

rt\_kprintf("thread1: you have two chopsticks,and can eat\n", e);

}

rt\_kprintf("thread1 leave.\n");//执行完两个IF后，直接printf退出

}

ALIGN(RT\_ALIGN\_SIZE)

static char thread2\_stack[1024];

static struct rt\_thread thread2;

/\* 线程2入口 \*/

static void thread2\_send\_event(void \*param)

{

rt\_kprintf("thread2: send one chopstick\n");

rt\_event\_send(&event, EVENT\_FLAG3);

rt\_thread\_mdelay(200);

rt\_kprintf("thread2: send one chopstick\n");

rt\_event\_send(&event, EVENT\_FLAG5);

rt\_thread\_mdelay(200);

rt\_kprintf("thread2: send another chopstick\n");

rt\_event\_send(&event, EVENT\_FLAG3);

rt\_kprintf("thread2 leave.\n");

}

int event\_sample(void)

{

rt\_err\_t result;

/\* 初始化事件对象 \*/

result = rt\_event\_init(&event, "event", RT\_IPC\_FLAG\_FIFO);//名字是"event"，是优先级等待方式

if (result != RT\_EOK)

{

rt\_kprintf("init event failed.\n");

return -1;

}

rt\_thread\_init(&thread1,

"thread1",

thread1\_recv\_event,

RT\_NULL,

&thread1\_stack[0],

sizeof(thread1\_stack),

THREAD\_PRIORITY - 1, THREAD\_TIMESLICE);

rt\_thread\_startup(&thread1);

rt\_thread\_init(&thread2,

"thread2",

thread2\_send\_event,

RT\_NULL,

&thread2\_stack[0],

sizeof(thread2\_stack),

THREAD\_PRIORITY, THREAD\_TIMESLICE);

rt\_thread\_startup(&thread2);

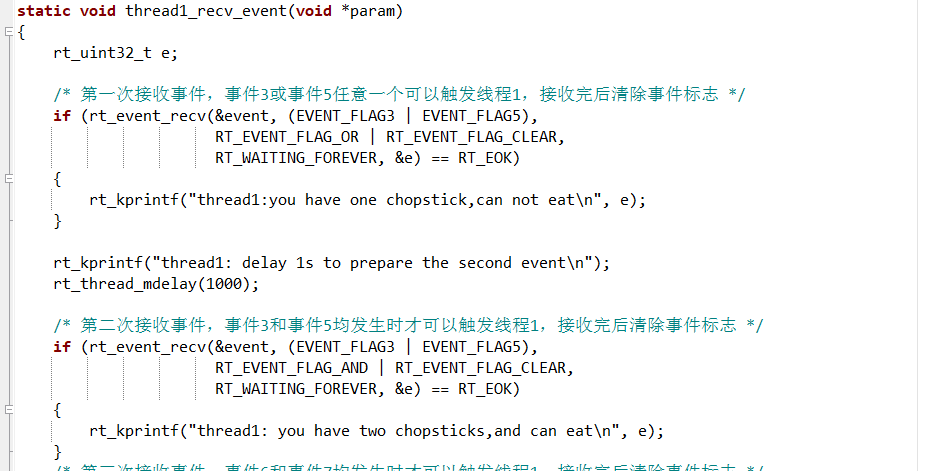
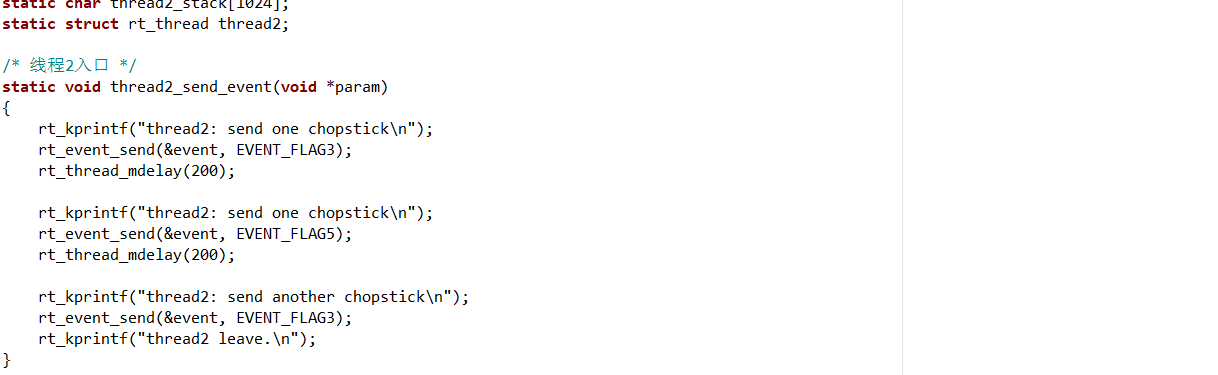
return 0;

}

/\* 导出到 msh 命令列表中 \*/

MSH\_CMD\_EXPORT(event\_sample, event sample);

## 3.4 RT\_Thread操作系统开发的调试

通过利用事件，信号量和线程，实现了哲学家进餐的问题。

遇到的问题和解决办法：哲学家进餐问题需要充分考虑死锁的问题，当每人都拥有一根筷子时，那么所有的人都无法吃饭，所以我通过设置线程的优先级，来控制谁有资格能优先拿到，同时利用信号量模拟释放筷子和拿到筷子的操作，利用事件来模拟筷子的发送和并判断筷子数量是否满足吃饭要求，在书写过程中，由于线程优先级相同导致死锁问题，最终通过修改哲学家进餐的优先级解决了死锁问题，完成了哲学家进餐问题。