信号量解决哲学家就餐问题

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

- 了解什么是哲学家就餐问题
- 学习信号量的互斥功能
- 学习信号量的同步功能
- 学习使用信号量来解决生产者消费者问题

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

1965 年,Dijkstra 提出并解决了一个他称之为哲学家就餐的同步问题。从那时起，每个发明新的同步原语的人都希望通过解决哲学家就餐问题来展示其同步原语的精妙之处。这个问题可以简单地描述如下：五个哲学家围坐在一张圆桌周围，每个哲学家面前都有一盘通心粉。由于通心粉很滑，所以需要两把叉子才能夹住。相邻两个盘子之间放有一把叉子，餐桌如图 1 示。

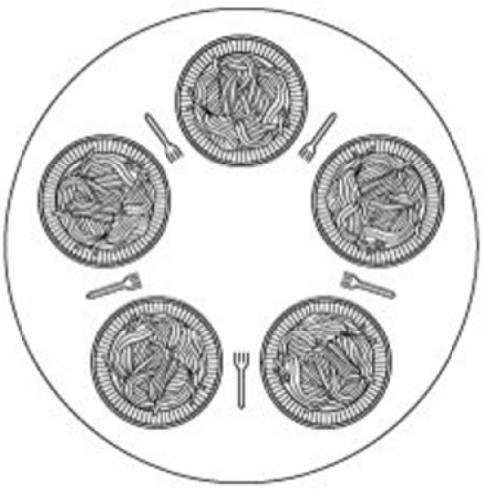

图 1

哲学家的生活中有两种交替活动时段：即吃饭和思考（这只是一种抽象，即对哲学家而言其他活动都无关紧要）。当一个哲学家觉得饿了时，他就试图分两次去取其左边和右边的叉子，每次拿一把，但不分次序。如果成功地得到了两把叉子，就开始吃饭，吃完后放下叉子继续思考。关键问题是：能为每一个哲学家写一段描述其行为的程序，且决不会死锁吗？

声明：以上文字和图片出自《Operating Systems Design and Implementation》 By Andrew S. Tanenbaum。

该实验是对哲学家就餐问题的模拟实现。

实验设计

源程序说明

本实验对应 kernel_sem_phd_dinner。

系统依赖

在 rtconfig.h 中需要开启

- #define RT_USING_SEMAPHORE
  此项必选，选择此项后才可以使用信号量相关 API。
- #define RT_USING_HEAP
  此项可选，开启此项可以创建动态信号量，如果使用静态信号量，则此项不是必要的
- #define RT_USING_CONSOLE
  此项必须，本实验使用 rt_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 中定义全局变量

本程序为《Operating Systems Design and Implementation》中的哲学家就餐问题的就解决方案。

如果读者想要对这个问题进行深入的理解请参考此书 2.3.1 节。

每位哲学家一共有三种状态，分别为思考(THINKING)，饥饿(HUNGRY)，和进餐（EATING）。当哲学家从思考中醒来则进入到饥饿状态，他会试图获取餐叉。当获取到两把叉子，则进入进餐状态（EATING）

使用一个数组 phd_state[N]，来跟踪每位哲学家的状态，在本实验中，N 为 5。

一个哲学家只有当两个邻居都没有进餐是才允许进入到进餐状态。哲学家 i 的两个邻居由 LEFT_PHD(i)和 RIGHT_PHD(i)。即若 i 为 2,则 LEFT_PHD 为 1，RIGHT_PHD 为 3。

该程序使用了一个信号量数组，每个信号量对应一位哲学家，这样在所需的叉子被占用时，想进餐的哲学家就被阻塞。

代码如下所示。

哲学家就餐问题代码

```
#define THREAD_STACK_SIZE 1024  /* 定义线程堆栈的大小 */
#define N 5                    /* 定义哲学家的数目 5 */
struct rt_semaphore sem[N];     /* 每位哲学家一个信号量 */
struct rt_semaphore sem_lock;   /* 定义二值信号量实现临界区互斥 */

enum _phd_state {              /* 定义使用枚举类型表示哲学家状态*/
THINKING = 0,
HUNGRY,
EATING,
} phd_state[N];                /* 定义哲学家状态数组 */

const char * status_string[N] =
{
   "thinking",
   "hungry",
   "eating",
};

#define LEFT_PHD(i)   ((i+N-1)%N) /* 哲学家 i 左边的哲学家 */
#define RIGHT_PHD(i)  ((i+1)%N)   /* 哲学家 i 右边的哲学家 */

#define LEFT_PHD(i)   ((i+N-1)%N) /* 哲学家 i 左边的哲学家 */
#define RIGHT_PHD(i)  ((i+1)%N)   /* 哲学家 i 右边的哲学家 */

/* 哲学家线程 */
void phd_thread_entry(void* parameter)
{
   int i;
```

```
    i = (int)parameter;
    rt_kprintf("phd %i starts...\n", i);
    while(1)
    {
        /* thinking */
        rt_thread_delay(RT_TICK_PER_SECOND);
        rt_kprintf("phd %d is %s\n", i, status_string[phd_state[i]]);

        /* take forks */
        take_forks(i);

        /* eating */
        rt_kprintf("phd %d is %s\n", i, status_string[phd_state[i]]);
        rt_thread_delay(RT_TICK_PER_SECOND*2);

        /* put forks */
        put_forks(i);
    }
}

void take_forks(int i)
{
    /* 进入临界区*/
    rt_sem_take(&sem_lock, RT_WAITING_FOREVER);
    phd_state[i] = HUNGRY;
    test(i);
    /* 退出临界区*/
    rt_sem_release(&sem_lock);

    /* 如果不处于 EATING 状态则阻塞哲学家 */
    rt_sem_take(&sem[i], RT_WAITING_FOREVER);
}

void put_forks(int i)
{
    /* 进入临界区*/
    rt_sem_take(&sem_lock, RT_WAITING_FOREVER);
    phd_state[i] = THINKING;
```

```
    test(LEFT_PHD(i));
    test(RIGHT_PHD(i));
    /* 退出临界区*/
    rt_sem_release(&sem_lock);
}

void test(int i)
{
    if (phd_state[i] == HUNGRY &&
        phd_state[LEFT_PHD(i)] != EATING &&
        phd_state[RIGHT_PHD(i)] != EATING)
    {
        phd_state[i] = EATING;

        /* 可以得到叉子，故发布信号量 */
        rt_sem_release(&sem[i]);
    }
}
```

在 applications/application.c 中，初始化信号量数组，以及创建 5 个哲学家线程。注意，程序中哲学家状态数组的操作为临界区，需要互斥，这里使用二值信号量 sem_lock 实现。5 个哲学家线程具有相同的优先级和时间片。

初始化线程代码

```
int rt_application_init()
{
    int i;
    rt_thread_t tid;
    rt_thread_t init_thread;
    rt_err_t result;

    /* 初始化信号量 */
    result = rt_sem_init(&sem_lock , "lock", 1, RT_IPC_FLAG_FIFO);
    if (result != RT_EOK)
        goto _error;
    for (i=0; i<5; i++)
```

```
    {
        result = rt_sem_init(&sem[i] , "sem", 0, RT_IPC_FLAG_FIFO);
        if (result != RT_EOK)
            goto _error;
}
…..
    for (i=0; i<5; i++)
    {
        tid = rt_thread_create(
                    "phd",
                    phd_thread_entry,
                    (void *)i,
                    THREAD_STACK_SIZE, 10, RT_TICK_PER_SECOND*3);
        if(tid != RT_NULL)
            rt_thread_startup(tid);
}
```

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：

串口输出

```
\ | /
- RT -     Thread Operating System
 / | \     1.1.0 build Aug  7 2012
 2006 - 2012 Copyright by rt-thread team
phd 0 starts...
phd 1 starts...
phd 2 starts...
phd 3 starts...
phd 4 starts...
phd 0 is thinking
phd 0 is eating
phd 1 is thinking
phd 2 is thinking
```

```
phd 2 is eating
phd 3 is thinking
phd 4 is thinking
phd 4 is eating
phd 1 is eating
phd 0 is thinking
phd 2 is thinking
phd 3 is eating
phd 0 is eating
phd 4 is thinking
phd 1 is thinking
phd 2 is eating
phd 4 is eating
phd 3 is thinking
phd 0 is thinking
phd 1 is eating
phd 3 is eating
phd 2 is thinking
phd 4 is thinking
phd 0 is eating
phd 2 is eating
phd 1 is thinking
phd 3 is thinking
phd 4 is eating
phd 1 is eating
phd 0 is thinking
phd 2 is thinking
….
```

结果分析

由于 5 个哲学家线程完全一致，那么在运行一段时间后，他们进餐的概率将大致相同。将日志信息保存成文本文件，然后分别统计下面四条语句的出现次数，如果他们基本相同，则表示算法为正确。

笔者运行一段时间后，统计得到如下结果：

字符串	行数

phd 0 is eating	881
phd 1 is eating	880
phd 2 is eating	880
phd 3 is eating	880
phd 4 is eating	880

小技巧:如果读者安装了 cygwin,那么可以使用如下命令来实现统计,

```
$ grep "phd 3 is eating" test.log  | wc -l
```

test.log 为读者的串口日志信息,此命令的输出即为字符串“phd 3 is eating”在 test.log 中出现的次数。

总结

本实验演示了 RT-Thread 中使用信号量解决哲学家就餐问题，关于这个问题的深入讨论请参阅操作系统相关书籍，在本实验中不予以过多介绍。