实验四　μC/OS-II 系统原理实验

一、　实验目的

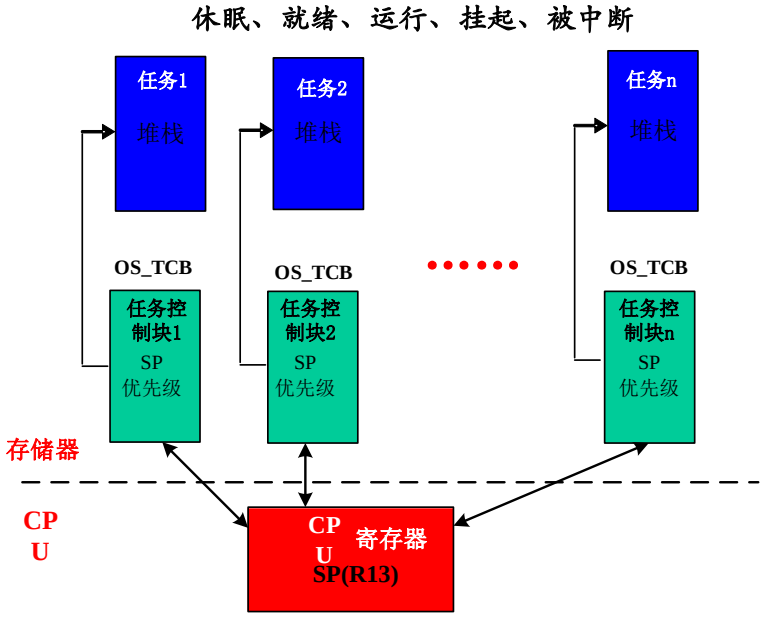
1. 掌理解任务管理的基本原理，了解任务的各个基本状态及其变迁过程；
2. 掌握 μCOS-II 中任务管理的基本方法：创建、启动、挂起、解挂任务；
3. 掌握 μCOS-II 中任务使用信号量的一般原理。

二、　实验内容

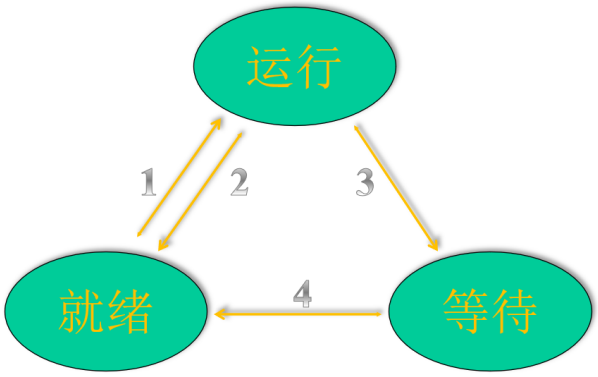
1. 掌设计多个应用任务，验证任务管理函数；
2. 通过实现“哲学家就餐”问题，验证信号量对任务间互斥访问的支持；
3. 应用信号量实现任务间的同步；
4. 设计 7 个任务，并顺序执行。

三、　实验原理

1. 任务管理的基本状态



2. 任务的状态切换



四、 实验步骤

1. 验证示例源码
   - 拷贝整个实验例程源码目录到本地磁盘自己的工作目录下；
   - 使用 μ Vision IDE for ARM 通过 ULINK2 仿真器连接实验板，打开实验例程目录 04-uCOS\2.1_Task_test 子目录下的 ucos2.Uv2 例程，编译链接工程；
   - 将程序下载到实验平台的 NorFlash 中，观察串口输出；
   - 打开实验例程目录\04-uCOS\2.3_Semaphore_test 子目录下的 ucos2.Uv2 例程，编译链接工程；
   - 下载调试，观察结果。
2. 设计实现一个多任务应用程序
   - 拷贝示例实验源码工程；
   - 设计 7 个任务，并用信号量实现 7 个任务顺序执行，将执行结果在串口上打印出来。

五、 实验结果

通过将源码编译并用 Flash 下载，在超级终端使用 COM4 串口查看串口输出信息，预期的输出结果应如下：

```
Emboard Emsbc2410 boot success!
uCOS-II Running...
Now task1 is running!
Task1 resumed task2!
Now task2 is running!
Task2 resumed task3!
Now task3 is running!
Task3 resumed task4!
Now task4 is running!
Task4 resumed task5!
Now task5 is running!
Task5 resumed task6!
Now task6 is running!
Task6 resumed task7!
Now task7 is running!
Task7 resumed task1!
Now task1 is running!
Task1 resumed task2!
••••••
```

六、 程序说明

本实验程序基于给定的示例工程 Semaphore_test，进行自主设计修改，程序的核心代码如下：

1. 定义指向 OS_EVENT 类型的指针，sem1~sem7 控制任务之间的同步关系；初始化每个人所需要的静态堆栈 TestTask1Stk~TestTask7Stk，同时声

明 7 个任务函数 Task1~Task7，之后生成每个任务所对应的同步信号量，sem1~sem7，并且实验要求是顺序执行，因此使得所有同步信号量的初值只有一个 1，即任务一先运行。

```
OS_EVENT *UART_sem;
OS_EVENT *sem1;
OS_EVENT *sem2;
OS_EVENT *sem3;
OS_EVENT *sem4;
OS_EVENT *sem5;
OS_EVENT *sem6;
OS_EVENT *sem7;

OS_STK TestTask1Stk[TaskStkLeath];
OS_STK TestTask2Stk[TaskStkLeath];
OS_STK TestTask3Stk[TaskStkLeath];
OS_STK TestTask4Stk[TaskStkLeath];
OS_STK TestTask5Stk[TaskStkLeath];
OS_STK TestTask6Stk[TaskStkLeath];
OS_STK TestTask7Stk[TaskStkLeath];

    /*create the first Semaphore in the pipeline with 1  to get the task started. */
    UART_sem = OSSemCreate(1);
    sem1 = OSSemCreate(1);
    sem2 = OSSemCreate(0);
    sem3 = OSSemCreate(0);
    sem4 = OSSemCreate(0);
    sem5 = OSSemCreate(0);
    sem6 = OSSemCreate(0);
    sem7 = OSSemCreate(0);

    /*create the tasks in uC/OS and assign decreasing priority to them */
    OSTaskCreate(Task1,(void *)1,&TestTask1Stk[TaskStkLeath-1],5);
    OSTaskCreate(Task2,(void *)2,&TestTask2Stk[TaskStkLeath-1],6);
    OSTaskCreate(Task3,(void *)3,&TestTask3Stk[TaskStkLeath-1],7);
    OSTaskCreate(Task4,(void *)4,&TestTask4Stk[TaskStkLeath-1],8);
    OSTaskCreate(Task5,(void *)5,&TestTask5Stk[TaskStkLeath-1],9);
    OSTaskCreate(Task6,(void *)5,&TestTask5Stk[TaskStkLeath-1],10);
    OSTaskCreate(Task7,(void *)5,&TestTask5Stk[TaskStkLeath-1],11);
```

2. 创建 7 个并发任务，并用创建的信号量实现任务之间的同步；

其中，任务一首先先运行，然后会唤醒任务二，之后的每个任务依次唤醒下一个任务，直到任务七会再次唤醒任务一，整个流程会形成循环顺序执行。

```
void Task1(void *pdata)
{
    while(1)
    {
        OSSemPend(sem1, 0, &err);
        uart_sendstring("Now task1 is running!\r\n");
        OSTimeDly(30);
        uart_sendstring("Task1 resumed task2!\r\n");
        OSSemPost(sem2);
    }
}

void Task2(void *pdata)
{
    while(1)
    {
        OSSemPend(sem2, 0, &err);
        uart_sendstring("Now task2 is running!\r\n");
        OSTimeDly(30);
        uart_sendstring("Task2 resumed task3!\r\n");
        OSSemPost(sem3);
    }
}

void Task3(void *pdata)
{
    while(1)
    {
        OSSemPend(sem3, 0, &err);
        uart_sendstring("Now task3 is running!\r\n");
        OSTimeDly(30);
        uart_sendstring("Task3 resumed task4!\r\n");
        OSSemPost(sem4);
    }
}

void Task4(void *pdata)
{
    while(1)
    {
        OSSemPend(sem4, 0, &err);
        uart_sendstring("Now task4 is running!\r\n");
        OSTimeDly(30);
        uart_sendstring("Task4 resumed task5!\r\n");
```

```
45.         OSSemPost(sem5);
46.     }
47. }
48. 
49. void Task5(void *pdata)
50. {
51.     while(1)
52.     {
53.         OSSemPend(sem5, 0, &err);
54.         uart_sendstring("Now task5 is running!\r\n");
55.         OSTimeDly(30);
56.         uart_sendstring("Task5 resumed task6!\r\n");
57.         OSSemPost(sem6);
58.     }
59. }
60. void Task6(void *pdata)
61. {
62.     while(1)
63.     {
64.         OSSemPend(sem6, 0, &err);
65.         uart_sendstring("Now task6 is running!\r\n");
66.         OSTimeDly(30);
67.         uart_sendstring("Task6 resumed task7!\r\n");
68.         OSSemPost(sem7);
69.     }
70. }
71. void Task7(void *pdata)
72. {
73.     while(1)
74.     {
75.         OSSemPend(sem7, 0, &err);
76.         uart_sendstring("Now task7 is running!\r\n");
77.         OSTimeDly(30);
78.         uart_sendstring("Task7 resumed task1!\r\n");
79.         OSSemPost(sem1);
80.     }
81. }
```

七、 心得体会

基于这次实验，我对操作系统的知识有了更深刻的理解，在实验中以 C/OS-II 的内核结构和任务管理为内容，着重掌握了如何利用信号量机制实现任务通信。此次实验经历让我为接下来的实验有了充分的准备和坚实的基础。