Ch2 进程状态转换

**背景知识**

2.1 uC/OS-II中的任务管理

2.1.1 进程与任务

通常将进程理解为程序的一次执行，反映程序动态运行过程。当程序由外存装载入内存并开始执行时，操作系统会为其创建进程控制块PCB，并通过PCB对进程进行管理。

在uC/OS-II运行时，一般不会从外存装载新的程序执行，所有需要执行的用户程序随内核同时编译并生成一个二进制镜像。从这个意义上说，uC/OS-II中的用户程序与内核处于同一个实体中，共享实体的所有资源，用户程序以线程为单位运行并接受内核调度。在uC/OS-II中，习惯用任务（task）来描述用户程序的执行过程，任务也表达了接受调度的基本单位。从代码上看，它是一个与main函数并列的C函数。

**例2-1** 一个用C函数表示的任务

void TaskStart(void \*pdata)

{

……

while (1) { //起始任务的循环体，每隔100ticks输出1次

printf("Now running for %6u ticks - press ESC to stop\n", OSTime);

if (PC\_GetKey(&key) == TRUE) { //检测是否有键按下

if (key == 0x1B) exit(0); //如果按下ESCAPE键，退出程序

}

OSTimeDly(100); //等待100ticks

}

}

上述起始任务TaskStart来自实验1工程项目中的Test.c，它是系统创建的第一个用户任务。

2.1.2 任务控制块TCB

uC/OS-II的任务控制块TCB作用类似进程控制块PCB，其定义在内核文件ucos\_ii.h下：

typedef struct os\_tcb {

OS\_STK \*OSTCBStkPtr; //指针指向当前的任务栈栈顶

OS\_STK \*OSTCBStkBottom; //指针指向任务栈栈底

INT32U OSTCBStkSize; //当前任务栈大小

INT16U OSTCBId; //Task ID 取值范围(0~65535)

struct os\_tcb \*OSTCBNext; //指针指向下一个TCB

struct os\_tcb \*OSTCBPrev; //指针指向上一个TCB

……

INT32U OSTCBDly; //任务需等待ticks（延时时间）

INT8U OSTCBStat; //任务状态\*

INT8U OSTCBStatPend; //事件等待标志

INT8U OSTCBPrio; //任务优先级（0为最高）

……

INT8U OSTCBDelReq; //指示任务是否需要删除

INT32U OSTCBCtxSwCtr; //任务被切换次数

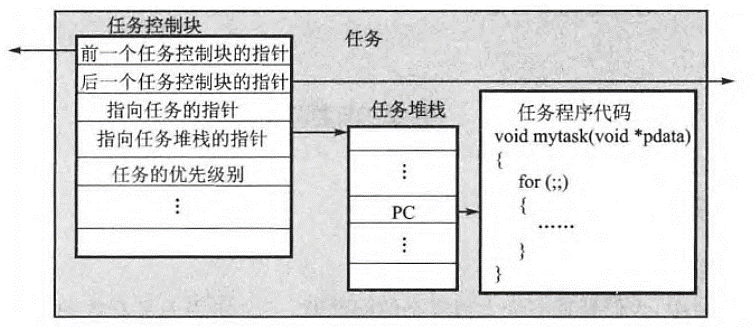
INT32U OSTCBCyclesTot; //任务已运行时钟周期数

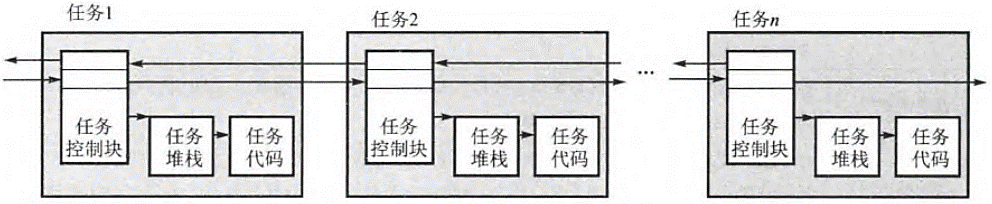
……

INT8U \*OSTCBTaskName; //任务名

……

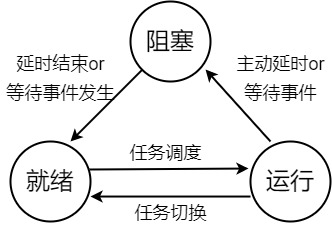
} OS\_TCB;

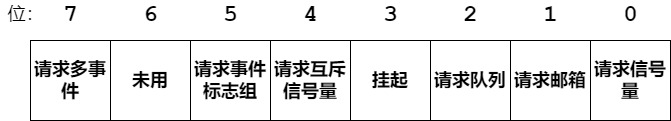
可将uC/OS-II中任务的组成看成如下图所示：

事实上，uC/OS-II作为多任务操作系统，运行过程中一般存在有多个任务，这些任务被一个双向链表所链接。uC/OS-II基于这个链表管理所有的任务。链表如图示：

注意uC/OS-II中的任务控制块TCB与一般意义上的进程控制块PCB有所区别。在宿主操作系统Windows看来，这些任务与内核共用一个内存空间（即属于同一个工程项目生成的可执行文件进程）。在用户程序Test.c内每创建一个任务，执行时对应了一个Windows线程。

2.2 uC/OS-II中的任务状态

uC/OS-II中的任务可在5种状态之间转换。除睡眠状态和中断服务状态外，剩余三种基本状态——运行态、就绪态和阻塞态遵循如下的转换关系：

联系上一节中任务控制块TCB的结构体定义，容易想到其成员变量OSTCBStat用来存放任务当前的状态。OSTCBStat被申明为INT8U，它实质上表示了按位设置的单字节信息，具体构成如下：

关于OSTCBStat的取值范围，在内核文件ucos\_ii.h中有如下定义：

#define OS\_STAT\_RDY 0x00u //就绪

#define OS\_STAT\_SEM 0x01u //等待信号量

#define OS\_STAT\_MBOX 0x02u //等待邮箱

#define OS\_STAT\_Q 0x04u //等待消息队列

#define OS\_STAT\_SUSPEND 0x08u //挂起

#define OS\_STAT\_MUTEX 0x10u //等待互斥信号量

#define OS\_STAT\_FLAG 0x20u //等待事件标志

#define OS\_STAT\_MULTI 0x80u //等待多事件

这里简单将OSTCBStat等于0认为处于就绪状态，事实上，当任务仅因主动延时原因而被阻塞后，OSTCBStat依然等于0。通过观察以上定义，易知并没有反映因延时阻塞的状态值。在新版本uC/OS-III中，也因此对OSTCBStat的字节定义作出了一定的改进。

在2.5节，我们将借助另一个成员变量OSTCBDly来帮助正确认识任务的当前状态。

思考：注意到ucos\_ii.h中并没有定义任务正在运行的状态，为什么？

（任何时刻只能有一个任务处于运行状态。当B任务查询A任务的状态时，A任务一定处于就绪或阻塞态，因为B任务正在台上运行。）

2.3 任务优先级

uC/OS-II规定系统中每个任务都按其重要程度分配给一个唯一的优先级，取值范围在0~255，也就是说最多可创建256个任务。规定任务最高优先级为0，在os\_cfg.h中，定义了最低优先级符号常量OS\_LOWEST\_PRIO：

#define OS\_LOWEST\_PRIO 63 /\* the lowest priority that can be assigned\*/

事实上，此配置下系统可提供给用户使用的优先级范围是0~61（OS\_LOWEST\_PRIO-2），因为最低的两个优先级OS\_LOWEST\_PRIO与OS\_LOWEST\_PRIO-1默认被两个系统任务所占用。其中空闲任务优先级最低，统计任务次之。

OS\_LOWEST\_PRIO值可以被更改。当你能够确认系统中执行任务的最大数目时，往往不需要使用最大值（ucos的63，ucos-ii的255）。例如只有5个用户任务，加上空闲与统计任务总任务数为7，则可将符号量OS\_LOWEST\_PRIO配置为6。如此可大大节省内存空间，体现了uC/OS操作系统精简可裁剪的特性。

在uC/OS-III版本中，不再为OS\_LOWEST\_PRIO指定63或255的上限，其值由用户根据实际情况设定。

2.4 空闲任务与统计任务

为了避免CPU在系统无用户任务时无事可做，uC/OS-II规定系统中必须包含一个空闲任务OS\_TaskIdle()，其代码如下：

void OS\_TaskIdle (void \*p\_arg)

{

……

for (;;) {

OS\_ENTER\_CRITICAL(); //进临界区（防中断）

OSIdleCtr++; //空闲计数全局量加1

OS\_EXIT\_CRITICAL(); //出临界区

OSTaskIdleHook(); //钩子函数，默认为空

}

}

从以上代码可看出，空闲任务除了不停地将空闲计数OSIdleCtr值加1，不做其它任何事情。而全局量OSIdleCtr的主要作用是提供给统计任务以计算CPU利用率。

空闲任务是系统任务，其代码不可更改且任务不可删除。空闲任务的优先级为最低的OS\_LOWEST\_PRIO。

统计任务OS\_TaskStat()是另一个重要的系统任务。它虽然不像空闲任务那样是系统硬性规定必须要包含，但一般我们不会取消统计任务。在os\_cfg.h中有如下初始设置：

#define OS\_TASK\_STAT\_EN 1 /\* Enable (1) or Disable(0) the statistics task\*/

统计任务依据OSIdleCtr定期计算并更新全局量OSCPUUsage，OSCPUUsage反映了当前系统的利用率。可以通过实时监测OSCPUUsage的值来获知多任务环境下系统运行是否良好。

统计任务要求必须经过初始化OSStatInit()。一般在系统开始运行后，由其它用户任务调用OSStatInit()一次。例如在第一个工程项目App01中，起始任务TaskStart在进入循环前有如下调用：

#if OS\_TASK\_STAT\_EN > 0

OSStatInit(); /\* Initialize uC/OS-II's statistics \*/

#endif

即当宏设置为打开情况下，TaskStart任务将调用OSStatInit()一次，主要作用是清零空闲计数全局量。

uC/OS规定统计任务的优先级为OS\_LOWEST\_PRIO-1，仅高于空闲任务。

**特别说明：**

1）空闲任务作为优先级最低的任务，其代码中不再含OSTimeDly()函数调用，即它不需要使自己进入延时等待（阻塞）状态。事实上，空闲任务只在运行态和就绪态之间转换。当系统中所有其它任务都处于阻塞状态时，空闲任务将获得运行机会，被内核调度上台执行。当有任何其它任务转为就绪状态，内核基于优先级抢占原则，将剥夺空闲任务的运行权并将其调出至就绪队列。

2）统计任务是系统中优先级次低的任务，其代码中循环体内含有OSTimeDly()函数调用：

OSTimeDly(OS\_TICKS\_PER\_SEC / 10u);

其作用是主动使自己延时等待1/10秒（100ms= 5 ticks）。在这5 ticks内，优先级更低的空闲任务有可能获得运行机会，用以累加空闲计数值。

思考：如果统计任务代码中不调用OSTimeDly()会如何？

（如果统计任务不调用OSTimeDly()，说明其不会进入阻塞状态，只在运行态和就绪态之间转换。当系统中其它任务都已阻塞，仅有统计任务和空闲任务就绪时，根据优先级原则，统计任务获得运行权，而空闲任务永远得不到上台运行的机会。）

3）uC/OS-II是基于优先级抢占的内核，为使高优先级的任务不至于总是独占CPU，让优先级较低的任务获得运行机会，规定除空闲任务之外的所有任务必须在代码中合适位置调用函数OSTimeDly()，使自己延时等待指定的时间以让出CPU使用权。

OSTimeDly()代码在内核文件os\_time.c中：

void OSTimeDly (INT32U ticks)

{

……

if (ticks > 0u) { /\* 0 means no delay! \*/

OS\_ENTER\_CRITICAL();

y = OSTCBCur->OSTCBY; /\* Delay current task \*/

OSRdyTbl[y] &= (OS\_PRIO)~OSTCBCur->OSTCBBitX;

if (OSRdyTbl[y] == 0u) {

OSRdyGrp &= (OS\_PRIO)~OSTCBCur->OSTCBBitY;

}

OSTCBCur->OSTCBDly = ticks; /\* Load ticks in TCB \*/

OS\_EXIT\_CRITICAL();

OS\_Sched(); /\* Find next task to run! \*/

}

}

目前我们只需要关心最后一行，即调用OSTimeDly()将引发一次内核任务调度OS\_Sched()。主动调用OSTimeDly()的任务将由运行态转为阻塞态，当指定的延时时间期满，内核通知此任务转为就绪态以接受调度。

**实验内容**

2.5 修改工程APP01

2.5.1 输出实时监测信息

统计任务主要用于定期计算并更新全局量OSCPUUsage。在统计任务开启情况下，可以输出其值用以监测CPU利用率。

在Test.c中，将起始任务TaskStart作如下修改：

printf("Now running for %6u ticks CPU利用率:%3d %% 任务数:%3d - press ESC to stop\n", OSTime, OSCPUUsage, OSTaskCtr);

全局量OSTaskCtr反映了当前系统中的总任务数。

运行工程并观察结果。

2.5.2 创建更多用户任务

如前文所述，用户任务在中体现为与main函数并列的C函数形式。在main函数中，通过调用内核函数OSTaskCreate创建用户任务。用户任务的结构大致如例2-2所示。

**例2-2** 任务函数结构

void MyTask1(void \*pdata)

{

pdata = pdata; /\* Prevent compiler warning \*/

while (1) {

……

OSTimeDly(150); /\* Wait 150 ticks \*/

}

}

一般情况下，任务含有一个循环体，并且循环体内最后一条语句为调用延时函数（设置了用户任务MyTask1需延时150 ticks）。

按创建TaskStart任务的方式，在main函数中以下语句创建了用户任务MyTask1，并指定其优先级为11：

OSTaskCreate(MyTask1, (void \*)0, &Task1Stk[TASK\_STK\_SIZE - 1], 11);

注意每当创建一个新的任务，应当首先定义好一个任务栈（全局数组，大小为TASK\_STK\_SIZE）。这里MyTask1任务栈的定义与TaskStart任务栈类似：

OS\_STK Task1Stk[TASK\_STK\_SIZE];

任务函数可参照例2-2，注意循环体内除OSTimeDly()外，增加以下输出语句：

printf("MyTask1 is running \n");

作用是指示当前任务正在台上执行。

同样地，定义任务栈Task2Stk~Task5Stk，增加任务函数MyTask2()~ MyTask5()。函数的循环体内对照增加以上输出语句。创建用户任务MyTask2~MyTask5，所有任务的优先级与OSTimeDly()参数指定遵循下表：

|  |  |  |
| --- | --- | --- |
| **任务名** | **优先级** | **OSTimeDly()参数** |
| MyTask1 | 11 | 150 |
| MyTask2 | 12 | 120 |
| MyTask3 | 13 | 90 |
| MyTask4 | 14 | 70 |
| MyTask5 | 15 | 30 |

运行工程并观察结果，体会不同任务执行频率的区别。

2.5.3 查询任务状态

在起始任务TaskStart中使用内核函数OSTaskQuery()查询其它任务信息。在Test.c文件全局变量定义处增加

OS\_TCB tcb\_copy;

作用是定义一个空的TCB结构体变量，用以复制被查询任务的TCB数据。

在TaskStart函数第一行定义局部变量：

INT8U err;

并在循环体内增加如下代码行：

err = OSTaskQuery (11, &tcb\_copy);

printf("任务状态: MyTask1 %d | ",tcb\_copy.OSTCBStat);

err = OSTaskQuery (12, &tcb\_copy);

printf("MyTask2 %d | ",tcb\_copy.OSTCBStat);

err = OSTaskQuery (13, &tcb\_copy);

printf("MyTask3 %d | ",tcb\_copy.OSTCBStat);

err = OSTaskQuery (14, &tcb\_copy);

printf("MyTask4 %d | ",tcb\_copy.OSTCBStat);

err = OSTaskQuery (15, &tcb\_copy);

printf("MyTask5 %d\n",tcb\_copy.OSTCBStat);

上述代码的作用是依次查询优先级为11~15的用户任务，并使用全局量tcb\_copy复制被查询任务的TCB数据。**uC/OS-II规定不能直接操作任务控制块，对TCB的查询、更改等操作需要通过相应内核函数进行。**

运行工程并观察结果。思考：为什么输出所有的任务状态都是0？

（根据2.2节中OSTCBStat的定义，当任务只是因为等待延时而阻塞时，成员变量OSTCBStat并没有与之对应的取值）

在以上代码块中，将OSTCBStat替换为OSTCBDly，运行工程并观察结果。（当OSTCBDly值为0时，表示被查询任务的延时结束，即处于真正的就绪状态等待被内核调度执行）

2.6 建立工程APP02

第一个工程APP01的输出是在控制台窗口滚动的方式，观察结果不是很方便。在工程APP02中，我们将实现一个相对固定的界面，以便更直观地了解uC/OS-II系统运行的总体情况。

2.6.1 新建工程

1、在VC2010新建项目窗口选择“Win32控制台应用程序”，取消“预编译头”，勾选“空项目”。工程项目属性配置“常规”-“字符集”下选择“使用多字节字符集”。

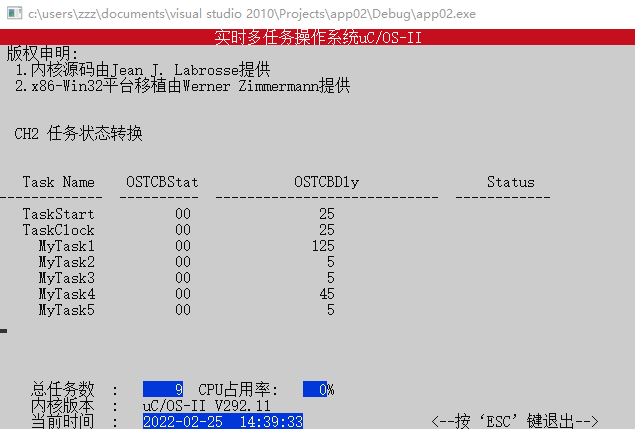
2、使用APP02工程的源码。将App文件夹拷入工程目录下。在“源文件”下添加现有项Test.c，在“头文件”下添加现有项app\_cfg.h、INCLUDE.h、OS\_CFG.h。

将Port文件夹以及Source文件夹完整拷入工程目录下。在项目中新建筛选器“Port”以及“Source”，对应分别将上述二个文件夹中的所有文件按现有项添加。

3、在项目属性配置“C/C++”-“常规”-“附加包含目录”下添加App、Port和Source目录的完整路径。

4、在“链接器”-“输入”-“附件依赖项”下添加winmm.lib。

2.6.2 运行界面

工程APP02运行结果如下：

在APP02中，使用了定点输出函数PC\_DispStr()以代替滚动输出方式的printf()。

关于函数PC\_DispStr()，它有如下参数：

void PC\_DispStr(INT8U x, INT8U y, INT8U \* s, INT8U color)

其中x、y表示控制台的坐标位置，字符串指针s表示输出内容。

2.6.3 对工程APP02的说明

1、除起始任务和用户任务MyTask1~ MyTask5外，额外创建了一个时钟任务TaskClock，它负责每隔一秒在屏幕中指定位置更新当前系统标准时间（按年月日-时分秒的格式）。

2、观察工程运行结果截图，其中有三个任务的OSTCBDly值出现了较小的数字（5 ticks）。因内核完成任务的切换也需要耗费一定的时钟节拍，可认为在查询值OSTCBDly小于某个节拍数的情况下（可取≤10），任务实际的OSTCBDly已归零，即处于就绪状态。结合前面的OSTCBStat值判断，这里第2、3、5号任务可认为处于已就绪状态，

3、任务数、CPU利用率和系统时间由用户任务负责更新并打印。而位于屏幕中间的两个值——OSTCBStat与OSTCBDly的更新打印由统计任务负责。统计任务的延时相对较短，故更新频率相对较快。

**附：实验报告——实验1**



**计算机与信息 学院实验报告**

|  |  |  |
| --- | --- | --- |
| 实验课程： | 操作系统实验 | |
| 实验编号： | 2 | |
| 实验名称： | 进程状态转换 | |
| 实验人员： | 学号 | 21111301083 |
| 姓名 | 房尚鹏 |
| 班级 | 21级人工智能 |
| 指导教师： | 李炳火 | |
| 实验室： | 2060203 | |
| 实验日期： | 2022年3月 17日 | |

【注】每次实验后，将此电子版实验报告与可执行文件（\*.exe）等打包整体上传。

一、实验目的

1．阅读实验指导书，熟悉μC/OS-II下任务管理的相关代码，包括任务的创建、任务查询、任务优先级以及任务状态。

2．掌握使用空闲任务与统计任务的方法。

3．深入理解uC/OS-II下任务控制块的组成结构。

4．通过实际创建多任务，认识任务之间切换和任务状态的转换。

二、实验内容与要求

1．按2.5.1节至2.5.3节顺序修改上一章工程app01，运行并观察结果，回答以下问题：

1）除在main函数中显式创建的任务外，系统中还含有什么任务？这些任务做了什么事情？

还包含空闲任务与统计任务

空闲任务除了不停地将空闲计数OSIdleCtr值加1，不做其它任何事情。而全局量OSIdleCtr的主要作用是提供给统计任务以计算CPU利用率。

统计任务主要用于定期计算并更新全局量OSCPUUsage。在统计任务开启情况下，可以输出其值用以监测CPU利用率。

1. 如何在uC/OS-II中设置任务最低优先级？

在os\_cfg.h中，定义了最低优先级符号常量OS\_LOWEST\_PRIO：

#define OS\_LOWEST\_PRIO 63 /\* Defines the lowest priority that can be assigned ... \*/

63便是当前的最低优先级，修改便可设置任务最低优先级

1. 通过以不同参数创建任务并观察运行结果，回答系统中影响不同任务上台执行频率的最大因素是什么。

任务的优先级

4）在2.5.3节，为什么查询输出所有任务的OSTCBStat值时都是0？

根据OSTCBStat的定义，当任务只是因为等待延时而阻塞时，成员变量OSTCBStat并没有与之对应的取值

2．新建工程app02，调试程序并回答问题：

1）总任务数、CPU利用率和系统时间分别由哪几个任务负责更新显示？更新频率如何？

任务数、CPU利用率和系统时间由用户任务负责更新并打印，统计任务的延时相对较短，故更新频率相对较快。

2）统计任务负责更新显示所有其它任务的OSTCBStat和OSTCBDly值，因更新频率较快导致不易观察。删除统计任务的显示功能，将其移至时钟任务TaskClock中。

注意到主界面中除OSTCBStat和OSTCBDly栏外，还有Status栏没有显示。规定每个任务的Status（区别于State）由“Ready”和“Waiting”两种状态组成，其中Ready状态表示当OSTCBStat==0并且OSTCBDly<=10时成立；当OSTCBDly>10时为Waiting状态。

请在合适地方补充和完善代码，完成Status栏的显示功能。

将OSTaskStatHook中都统计任务的显示功能删除，将其移至时钟任务TaskClock中。

void TaskClock (void \*pdata)

{

char s[40];

INT8U i;

pdata = pdata;

for (;;) {

PC\_GetDateTime(s);

PC\_DispStr(18, 24, s, DISP\_FGND\_YELLOW + DISP\_BGND\_BLUE);

for (i = 0; i < 7; i++) { /\* Display task user data \*/

DispTaskStat(i);

}

OSTimeDly(OS\_TICKS\_PER\_SEC);

}

}

void OSTaskStatHook (void)

{

}

修改DispTaskStat中的语句，添加其判断条件

void DispTaskStat (INT8U id)

{

INT8U err;

char s[80];

err = OSTaskQuery (id+9, &tcb\_copy);

if(tcb\_copy.OSTCBStat == 0 && tcb\_copy.OSTCBDly <=10)

sprintf(s, " %02x %5d Ready", tcb\_copy.OSTCBStat, tcb\_copy.OSTCBDly);

else

sprintf(s, " %02x %5d Waiting", tcb\_copy.OSTCBStat, tcb\_copy.OSTCBDly);

PC\_DispStr(13, (INT8U) (id+11), s, DISP\_FGND\_BLACK + DISP\_BGND\_LIGHT\_GRAY);

}

3）函数OSTaskStatHook做了什么事情？它如何被调用的？调研钩子函数的基本概念与作用。

OSTaskStatHook做了统计任务的显示功能

它在os\_core.c文件中的OS\_TaskStat函数调用

钩子函数：钩子函数是在一个事件触发的时候，在系统中捕获到他，然后做一些操作。是一段用于处理系统消息的程序。

3．在控制台适当位置输出开发者信息（学号、姓名）。

4．提交test.c文件、实验报告、最终生成的exe可执行文件。

三、设计步骤（学生填写）

1．概要设计（文字）

1）打开VS2010，修改工程app01，工程属性需要（不需要）更改

2）为实现xxx功能，需要修改xxx文件的xxx函数（部分），理由是

在统计任务开启情况下，可以输出其值用以监测CPU利用率。

为实现创建更多用户任务，需要修改Test.c文件的main函数。

为实现查询任务状态，需要修改Test.c文件的TaskStart函数。理由是依次查询优先级为11~15的用户任务，并使用全局量tcb\_copy复制被查询任务的TCB数据。

3）打开VS2010建立工程app02，将app02工程的xxx属性值作如下调整

1、在VC2010新建项目窗口选择“Win32控制台应用程序”，取消“预编译头”，勾选“空项目”。工程项目属性配置“常规”-“字符集”下选择“使用多字节字符集”。

2、使用APP02工程的源码。将App文件夹拷入工程目录下。在“源文件”下添加现有项Test.c，在“头文件”下添加现有项app\_cfg.h、INCLUDE.h、OS\_CFG.h。

将Port文件夹以及Source文件夹完整拷入工程目录下。在项目中新建筛选器“Port”以及“Source”，对应分别将上述二个文件夹中的所有文件按现有项添加。

3、在项目属性配置“C/C++”-“常规”-“附加包含目录”下添加App、Port和Source目录的完整路径。

4、在“链接器”-“输入”-“附件依赖项”下添加winmm.lib。

4）为实现易观察OSTCBStat和OSTCBDly值，删除OSTaskStatHook中统计任务的显示功能，将其移至时钟任务TaskClock中。

为了实现Status栏的显示功能，修改DispTaskStat中的语句，添加其判断条件

2．详细设计（文字+关键代码）

实现输出实时监测信息，需要修改Test.c文件的TaskStart函数作如下关键代码更改：

printf("Now running for %6u ticks CPU利用率:%3d %% 任务数:%3d - press ESC to stop\n", OSTime, OSCPUUsage, OSTaskCtr);

为实现创建更多用户任务在main函数中，通过调用内核函数OSTaskCreate创建用户任务。

void MyTask1(void \*pdata)

{

pdata = pdata; /\* Prevent compiler warning \*/

while (1) {

printf("MyTask1 is running \n");

OSTimeDly(150); /\* Wait 150 ticks \*/

}

}

为了实现查询任务状态，在TaskStart函数作如下关键代码更改：

while (1) /\* Startup task's infinite loop \*/

{

/\* ToDo: Place additional code for your startup task, which you want to run once or periodically here \*/

//printf("Now running for %6u ticks - press ESC to stop\n", OSTime);

printf("Now running for %6u ticks CPU利用率:%3d %% 任务数:%3d - press ESC to stop\n", OSTime, OSCPUUsage, OSTaskCtr);

if (PC\_GetKey(&key) == TRUE) { /\* See if key has been pressed \*/

if (key == 0x1B) { /\* Yes, see if it's the ESCAPE key \*/

exit(0); /\* End program \*/

}

}

err = OSTaskQuery (11, &tcb\_copy);

printf("任务状态: MyTask1 %d | ",tcb\_copy.OSTCBDly);

err = OSTaskQuery (12, &tcb\_copy);

printf("MyTask2 %d | ",tcb\_copy.OSTCBDly);

err = OSTaskQuery (13, &tcb\_copy);

printf("MyTask3 %d | ",tcb\_copy.OSTCBDly);

err = OSTaskQuery (14, &tcb\_copy);

printf("MyTask4 %d | ",tcb\_copy.OSTCBDly);

err = OSTaskQuery (15, &tcb\_copy);

printf("MyTask5 %d\n",tcb\_copy.OSTCBDly);

/\* ToDo: Don't forget to call the uCOS scheduler with OSTimeDly etc., to give other tasks a chance to run \*/

OSTimeDly(100); /\* Wait 100 ticks \*/

}

为了实现更加容易观察OSTCBStat和OSTCBDly值，将OSTaskStatHook中都统计任务的显示功能删除，将其移至时钟任务TaskClock中。

void TaskClock (void \*pdata)

{

char s[40];

INT8U i;

pdata = pdata;

for (;;) {

PC\_GetDateTime(s);

PC\_DispStr(18, 24, s, DISP\_FGND\_YELLOW + DISP\_BGND\_BLUE);

for (i = 0; i < 7; i++) { /\* Display task user data \*/

DispTaskStat(i);

}

OSTimeDly(OS\_TICKS\_PER\_SEC);

}

}

void OSTaskStatHook (void)

{

}

为了实现Status栏的显示功能，修改DispTaskStat中的语句，添加其判断条件

void DispTaskStat (INT8U id)

{

INT8U err;

char s[80];

err = OSTaskQuery (id+9, &tcb\_copy);

if(tcb\_copy.OSTCBStat == 0 && tcb\_copy.OSTCBDly <=10)

sprintf(s, " %02x %5d Ready", tcb\_copy.OSTCBStat, tcb\_copy.OSTCBDly);

else

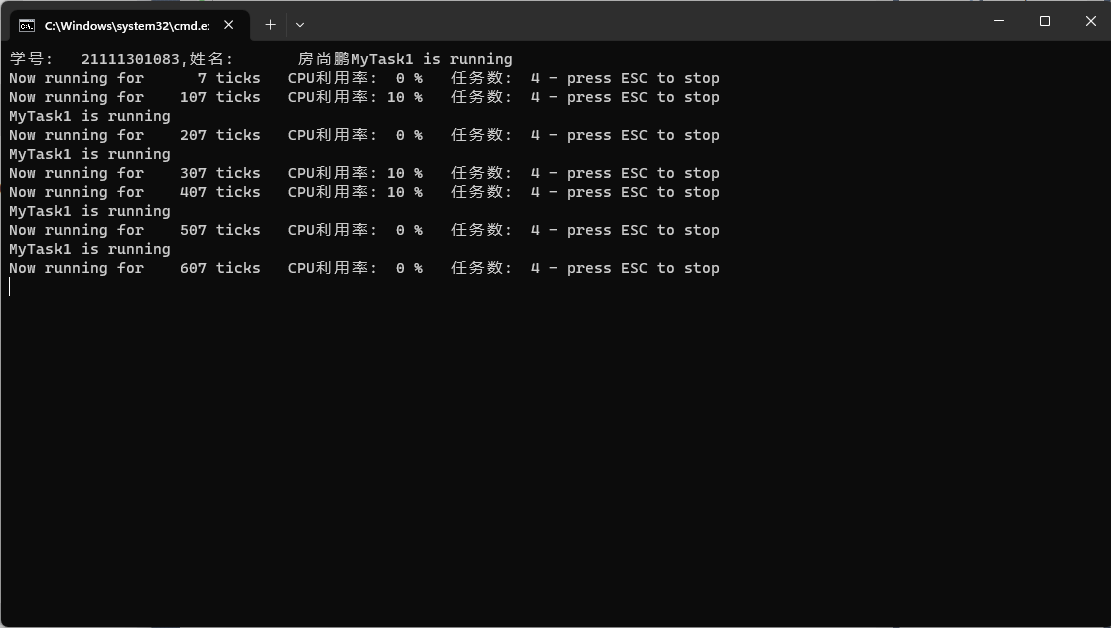
sprintf(s, " %02x %5d Waiting", tcb\_copy.OSTCBStat, tcb\_copy.OSTCBDly);

PC\_DispStr(13, (INT8U) (id+11), s, DISP\_FGND\_BLACK + DISP\_BGND\_LIGHT\_GRAY);

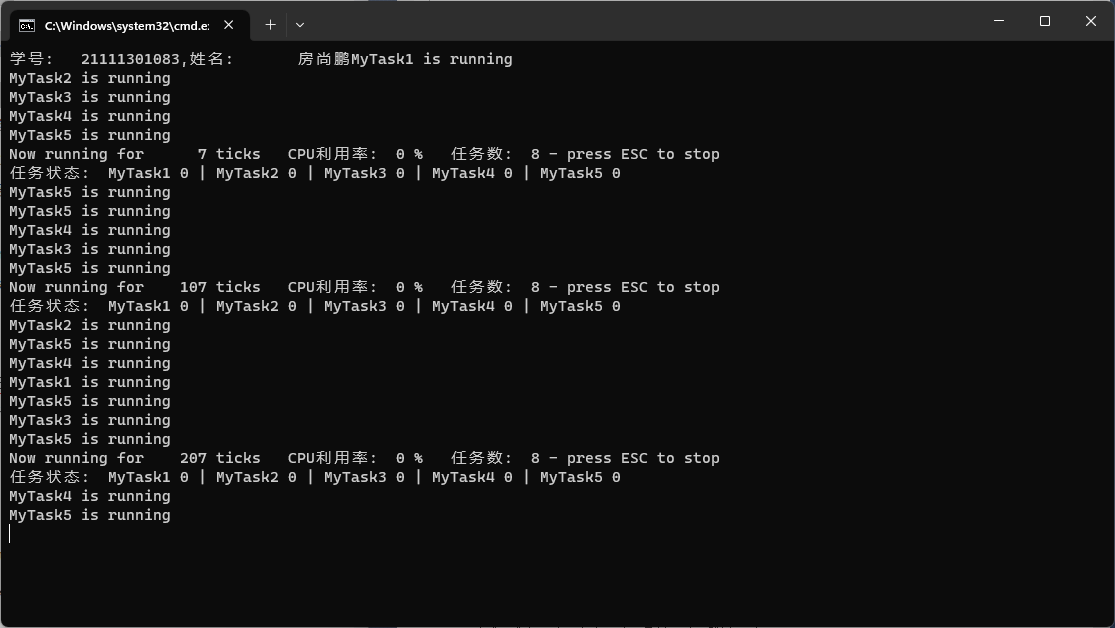
}

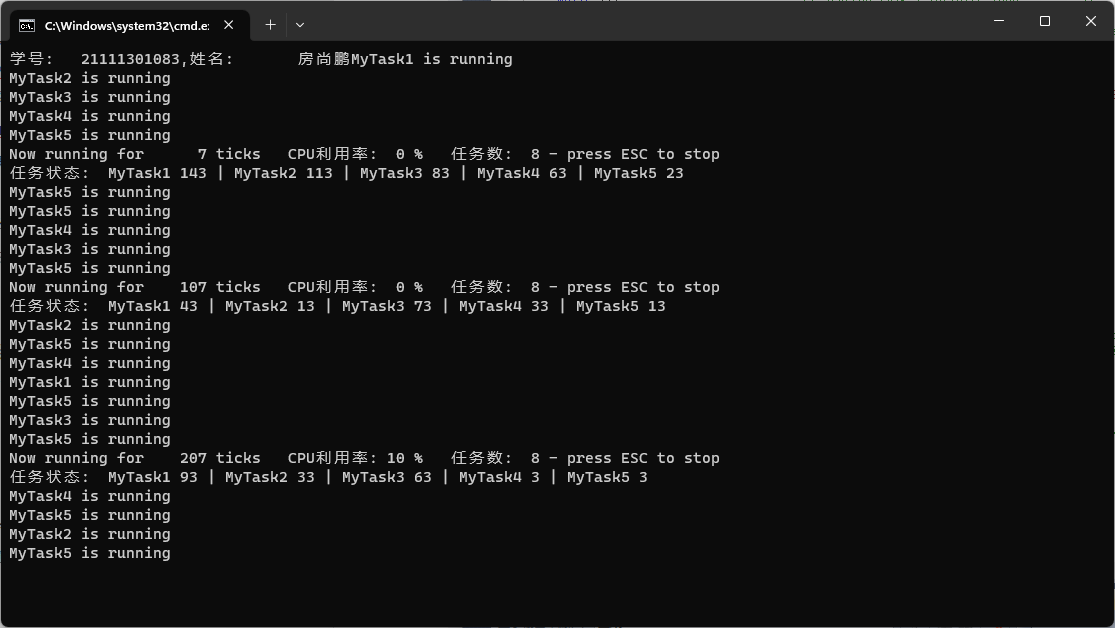
四、结果分析（学生填写）

1．实现基本功能（输出截图）





****

****

****

2．问题分析（文字+部分代码 简要总结）

**在本次实验中，通过app01的修改，熟悉任务的创建、任务查询、任务优先级以及任务状态，掌握使用空闲任务与统计任务的方法。**

OSTaskCreate(MyTask1, (void \*)0, &Task1Stk[TASK\_STK\_SIZE - 1], 11);

OSTaskCreate(MyTask2, (void \*)0, &Task2Stk[TASK\_STK\_SIZE - 1], 12);

OSTaskCreate(MyTask3, (void \*)0, &Task3Stk[TASK\_STK\_SIZE - 1], 13);

OSTaskCreate(MyTask4, (void \*)0, &Task4Stk[TASK\_STK\_SIZE - 1], 14);

OSTaskCreate(MyTask5, (void \*)0, &Task5Stk[TASK\_STK\_SIZE - 1], 15);

**创建不同优先级的任务，进行比较，观察。**

**通过app02的构建与修改，通过实际创建多任务，认识任务之间切换和任务状态的转换。**