

嵌入式操作系统

8 uC/OS-II 简单分析

陈香兰 (xlanchen@ustc.edu.cn)

计算机应用教研室@计算机学院
嵌入式系统实验室@苏州研究院
中国科学技术大学
Fall 2014

December 9, 2014

Outline

- 1 概述
- 2 目录分析
- 3 试运行和编译分析
- 4 uC/OS-II的功能解读
- 5 应用样例分析
- 6 调度算法分析
- 7 移植分析

Outline

- 1 概述
- 2 目录分析
- 3 试运行和编译分析
- 4 uC/OS-II的功能解读
- 5 应用样例分析
- 6 调度算法分析
- 7 移植分析

- 免费的公开源码实时操作系统
- 内核提供任务调度和管理、时钟管理、任务间同步与通信、内存管理和中断服务等功能
- 最多支持64个任务，
分别对应优先级0~63，其中0为最高优先级
- 可剥夺实时多任务内核
 - ▶ 调度工作的内容分为两部分：最高优先级任务的寻找和任务切换
- 内核是针对实时系统的要求来设计实现的，相对比较简单，可以满足较高的实时性要求
- 但是没有网络功能和文件系统，对于像媒体播放、需要网络和图形界面支持的应用就比较差

Outline

- 1 概述
- 2 目录分析
- 3 试运行和编译分析
- 4 uC/OS-II的功能解读
- 5 应用样例分析
- 6 调度算法分析
- 7 移植分析

目录分析

```
tree SOFTWARE/uCOS-II/ -L 1
```

```
SOFTWARE/uCOS-II/
```

```
|—— DOC
|—— EX1_x86L
|—— EX2_x86L
|—— EX3_x86L
|—— EX4_x86L.FP
|—— Ix86L
|—— Ix86L-FP
|—— SOURCE
```

序号	目录名	含义
1	DOC	一些文档，可以看看
2	EX1_x86L	第一个基于x86的应用实例
3	EX2_x86L	第二个基于x86的应用实例
4	EX3_x86L	第三个基于x86的应用实例
5	EX4_x86L.FP	第四个基于x86的应用实例
6	Ix86L	嵌入式x86开发板
7	Ix86L-FP	带浮点的嵌入式x86开发板
8	SOURCE	核心源代码

SOURCE 目录

- 在SOURCE目录下一共有10个C文件和1个头文件
 - ▶ 其中，文件uCOS-II.C仅仅是对其他.C文件的包含，因此真正的内核代码仅仅9个C文件和1个头文件

```
tree SOFTWARE/uCOS-II/SOURCE/ -h
```

```
SOFTWARE/uCOS-II/SOURCE/  
├── [ 49K] OS_CORE.C  
├── [ 43K] OS_FLAG.C  
├── [ 23K] OS_MBOX.C  
├── [ 14K] OS_MEM.C  
├── [ 27K] OS_MUTEX.C  
├── [ 34K] OS_Q.C  
├── [ 19K] OS_SEM.C  
├── [ 35K] OS_TASK.C  
├── [9.7K] OS_TIME.C  
├── [1.2K] uCOS-II.C  
└── [ 46K] uCOS-II.H
```

```
0 directories, 11 files
```

体系结构相关目录和板级支持包

- 以x86为例

- ▶ 体系结构相关目录

```
tree SOFTWARE/uCOS-II/Ix86L -h
```

```
SOFTWARE/uCOS-II/Ix86L
```

```
|—— [4.0K] BC45
|   |—— [ 14K] OS_CPU_A.ASM
|   |—— [ 15K] OS_CPU_C.C
|   |—— [6.1K] OS_CPU.H
|—— [4.0K] DOC
|   |—— [ 31K] 80x86L-ROM-RAM.xls
```

```
2 directories, 4 files
```

- ▶ 板级支持包 (PC上的模拟环境)

```
tree SOFTWARE/BLOCKS/
```

```
SOFTWARE/BLOCKS/
```

```
|—— PC
|   |—— BC45
|       |—— PC.C
|       |—— PC.H
```


应用实例相关目录

- 以实例1为例

```
tree SOFTWARE/uCOS-II/EX1_x86L/ -h
```

```
SOFTWARE/uCOS-II/EX1_x86L/
├── [4.0K] BC45
│   ├── [4.0K] SOURCE
│   │   ├── [ 903] INCLUDES.H
│   │   ├── [7.9K] OS_CFG.H
│   │   ├── [ 13K] TEST.C
│   │   └── [ 265] TEST.LNK
│   └── [4.0K] TEST
│       ├── [ 842] MAKETEST.BAT
│       ├── [126K] TEST.EXE
│       ├── [5.8K] TEST.MAK
│       └── [ 48K] TEST.MAP
```

3 directories, 8 files

Outline

- 1 概述
- 2 目录分析
- 3 试运行和编译分析**
- 4 uC/OS-II的功能解读
- 5 应用样例分析
- 6 调度算法分析
- 7 移植分析

在dosbox中运行uC/OS-II中的实例 I

- 我们直接运行uC/OS-II中已经编译好的实例，以实例1为例

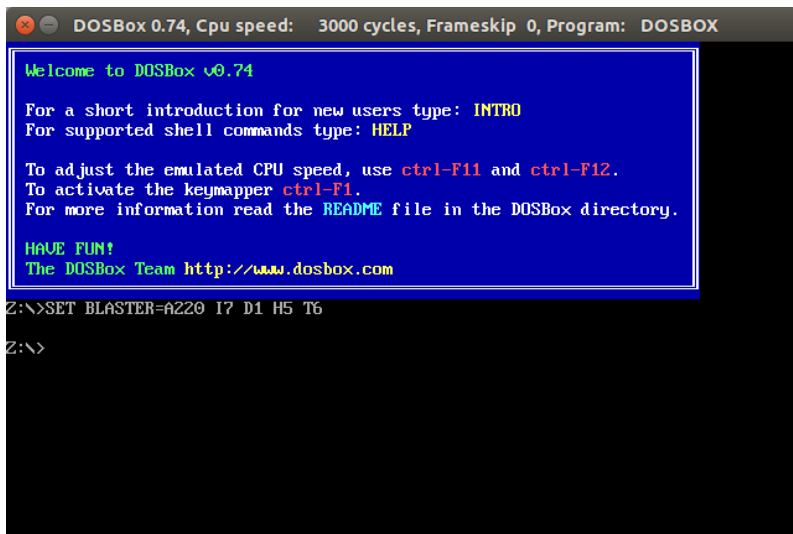
❶ 安装dosbox

```
sudo apt-get install dosbox
```

▶ 运行

```
dosbox
```

在dosbox中运行uC/OS-II中的实例 II



DOSBox 0.74, Cpu speed: 3000 cycles, Frameskip 0, Program: DOSBOX

```
Welcome to DOSBox v0.74

For a short introduction for new users type: INTRO
For supported shell commands type: HELP

To adjust the emulated CPU speed, use ctrl-F11 and ctrl-F12.
To activate the keymapper ctrl-F1.
For more information read the README file in the DOSBox directory.

HAVE FUN!
The DOSBox Team http://www.dosbox.com
```

Z:\>SET BLASTER=A220 I7 D1 H5 T6

Z:\>

- ▶ 在dosbox中运行help以及intro命令，寻找你感兴趣的帮助信息

在dosbox中运行uC/OS-II中的实例 III

② 建立dosbox与主机之间的共享文件夹，作为dosbox中的C盘

- ① 在主机上建立一个目录，作为给dosbox共享的文件夹，例如：

```
mkdir ~/workspace/dosdir
```

- ② 在dosbox中运行mount命令，如下：

```
mount c ~/workspace/dosdir
```

- ③ 进入dosbox的C盘

```
dosbox提示符Z:\>c:
```

在dosbox中运行uC/OS-II中的实例 IV

```
DOSBox 0.74, Cpu speed: 3000 cycles, Frameskip 0, Program: DOSBOX
CTRL-ALT-F8 : Start/Stop the recording of raw MIDI commands.
CTRL-F7      : Decrease frameskip.
CTRL-F8      : Increase frameskip.
CTRL-F9      : Kill DOSBox.
CTRL-F10     : Capture/Release the mouse.
CTRL-F11     : Slow down emulation (Decrease DOSBox Cycles).
CTRL-F12     : Speed up emulation (Increase DOSBox Cycles).
ALT-F12      : Unlock speed (turbo button/fast forward).

Z:\>mount c ~/workspace/dosdir
Drive C is mounted as local directory /home/xlanchen/workspace/dosdir/

Z:\>c:

C:\>ls
Illegal command: ls.

C:\>dir
Directory of C:\.
.                <DIR>                08-12-2014  9:26
..               <DIR>                08-12-2014  9:26
    0 File(s)                0 Bytes.
    2 Dir(s)                262,111,744 Bytes free.

C:\>
```

准备源代码并运行实例 I

- 在主机端，将uC-OS_II.rar解压缩到dosdir目录中

```
unrar x uC-OS_II.rar
```

或其他版本

- 重新运行dosbox，挂载c盘到共享目录dosdir，并进入c盘
- 在dosbox中，进入UC-OS II的实例1的相关目录并运行

```
dosbox提示符C:\>cd SOFTWARE\UCOS-II\EX1_X86L\BC45\TEST
```

```
dosbox提示符C:\>TEST.EXE
```

准备源代码并运行实例 II

```
DOSBox 0.74, Cpu speed: 3000 cycles, Frameskip 0, Program: TEST

Jean J. Labrosse

EXAMPLE #1

57810757005080821882289198176898587533654388180987771053807290015591512885031460
77437009097038662593426310381065558867246750728360379857399617278866796614065182
84079232561698023410333077061049130705366183905502371271261573933695390761280073
91518213909731851970032898751561519996115942732303171075127880207912607333721167
04661288986215459985245783804459795410666496749610884621223341851483229884665211
97404134217606322325272832478635030305269920434181714904924938811821263653197790
26820782533629557511400675239483202018362847916083759145385231168779172484198234
27513240097093820138253225962529772699226883225121719450703746945747427072890583
76357485897067857928636862403764733930505371115015490151524064603614819590714866
43567351000998849958797831096748916081336262458924333368600928044644591497974237
07336503042424204906224445085040205362385453149854207591936613338116102829802186
56183804377784847261102934388997497964573206647704357254210443960917491957556385
39495473235116725537016543810533592627720548712568854885485794110773882944584227
49864091165559633535901654278897565071833655596412744995654466903722978861215412
08751865343037927123199679808697601100397001253830749277665787574242767370995160
80352584279832858716184145845886003060905539033900964039872388856841126855441118

#Tasks          : 13 CPU Usage: 69 % 80387 FPU
#Task switch/sec: 2202 U2.52
```


准备编译环境并编译运行

- 准备编译环境？

请自行完成！

- 演示：编译并运行

uC/OS-II的编译过程

- ① 阅读MAKETEST.BAT
- ② 阅读TEST.MAK
 - ▶ 仔细阅读TEST.EXE的生成规则以及相关OBJ文件生成规则
- ③ 阅读TEST.LNK

思考1：如果要自己写一个运行在x86上的实例，如何进行？

思考2：如果要移植到一个真实的x86上，如何进行？

思考3：如果要移植到另一个平台（例如ARM）上，如何进行？

Outline

- 1 概述
- 2 目录分析
- 3 试运行和编译分析
- 4 uC/OS-II的功能解读**
- 5 应用样例分析
- 6 调度算法分析
- 7 移植分析

uC/OS-II的功能解读

- 首先，请仔细阅读一下DOC目录下的文件QuickRefChartV252-Color.PDF

- ① 信号量OS_SEM
- ② 互斥信号量OS_MUTEX
- ③ 标志OS_FLAG
- ④ 信箱OS_MBOX
- ⑤ 消息队列OS_Q
- ⑥ 内存管理OS_MEM
- ⑦ 任务管理OS_TASK
- ⑧ 时间管理OS_TIME
- ⑨ 其他OS_CORE

信号量

- 信号量是基于event实现的
- 一个信号量的主要包含两个部分
 - ① count值
 - ② 等待队列
- 提供6个接口函数对信号量进行操作
 - ① OSSemCreate(): 创建一个新的信号量
 - ② OSSemDel(): 销毁指定的信号量
 - ③ OSSemPend(): 请求一个信号量 (资源数1, 可能等待, 可以超时等待)
 - ④ OSSemAccept(): 请求一个信号量 (资源数1, 不等待)
 - ⑤ OSSemPost(): 释放一个信号量 (资源数1)
 - ⑥ OSSemQuery(): 查询一个信号量的信息

互斥信号量

- 提供6个接口函数对信号量进行操作

- ① OSMutexCreate(): 创建一个新的互斥信号量
- ② OSMutexDel(): 销毁指定的互斥信号量
- ③ OSMutexPend(): 请求一个互斥信号量 (可能等待, 可以超时等待)
- ④ OSMutexAccept(): 请求一个互斥信号量 (不等待)
- ⑤ OSMutexPost(): 释放一个互斥信号量
- ⑥ OSMutexQuery(): 查询一个互斥信号量的信息

问题：在uC/OS-II中，信号量（二进制）和互斥信号量有什么不一样？

事件标志

- OSFlagCreate()：创建
- OSFlagDel()：销毁
- OSFlagPend()：请求一个
- OSFlagAccept()：请求一个
- OSFlagPost()：发送
- OSFlagQuery()：查询一个信号量的信息

- 邮箱提供了一个消息指针
- 提供7个接口函数对邮箱进行操作
 - ① OSMboxCreate(): 创建一个新的邮箱
 - ② OSMboxDel(): 销毁一个邮箱
 - ③ OSMboxAccept(): 从邮箱中接收一个消息 (不等待)
 - ④ OSMboxPend(): 从邮箱中接收一个消息 (可能等待, 可以超时等待)
 - ⑤ OSMboxPost(): 发送一个消息到邮箱中
 - ⑥ OSMboxPostOpt(): 发送一个消息到邮箱中 (可以选择是否广播给所有等待任务)
 - ⑦ OSMboxQuery(): 查询一个邮箱的信息

消息队列

- 消息队列使用一个循环缓冲区来管理消息
- 提供7个接口函数对消息队列进行操作
 - ▶ OSQCreate()：创建一个新的消息队列
 - ▶ OSQDel()：销毁一个消息队列
 - ▶ OSQAccept()：从消息队列上接收一个消息（不等待）
 - ▶ OSQFlush()：清空一个消息队列
 - ▶ OSQPend()：从一个消息队列上接收一个消息（可能等待，可以超时等待）
 - ▶ OSQPost()：发送一个消息到消息队列末尾
 - ▶ OSQPostFront()：发送一个消息到消息队列首部
 - ▶ OSQPostOpt()：发送一个消息到消息队列首部（可以选择是末尾还是首部，是否广播）
 - ▶ OSQQuery()：查询一个消息队列的信息

问题：邮箱和消息队列有什么不一样？

内存管理

- 提供按固定大小管理内存的管理机制
- 提供7个接口函数对内存进行管理操作
 - ① OSMemCreate()：按固定大小管理一个给定的内存区域
 - ② OSMemGet()：分配一个固定大小的内存块
 - ③ OSMemPut()：释放一个固定大小的内存块
 - ④ OSMemQuery()：查询一个内存区的信息

任务管理

● 提供9个任务管理接口

- ① OSTaskCreate(): 创建一个新的任务
- ② OSTaskCreateExt(): 创建一个新的任务
- ③ OSTaskDel(): 销毁一个任务
- ④ OSTaskDelReq(): 请求一个任务自我销毁 (仅通知)
- ⑤ OSTaskSuspend(): 暂停任务运行
- ⑥ OSTaskResume(): 恢复任务运行
- ⑦ OSTaskChangePrio(): 改变任务优先级
- ⑧ OSTaskStkChk(): 任务堆栈检查
- ⑨ OSTaskQuery(): 获得任务TCB中的信息

时间管理OS_TIME

- 系统时间：OSTime（滴答数）
 - ▶ 系统初始化时设置为0；（OS_InitMisc()）
 - ▶ 发生时钟中断时，加1；（OSTimeTick）
 - ▶ 可以设置系统时间（OSTimeSet）
 - OSTimeDly()：任务延迟n个滴答（0~65535）
 - OSTimeDlyHMSM()：任务延迟指定的时间长度（小时，分钟，秒，毫秒）
 - OSTimeDlyResume()：因任务延迟到期而恢复任务
 - OSTimeGet()：获得系统时间
 - OSTimeSet()：设置系统时间
-
- OSTimeTick()：时钟滴答函数（OS_CORE.C）

事件机制

● 事件的类型

```
/*
*****
* OS_EVENT types
*****
*/
#define OS_EVENT_TYPE_UNUSED 0
#define OS_EVENT_TYPE_MBOX 1
#define OS_EVENT_TYPE_Q 2
#define OS_EVENT_TYPE_SEM 3
#define OS_EVENT_TYPE_MUTEX 4
#define OS_EVENT_TYPE_FLAG 5
```

- 事件控制块OS_EVENT，参见uCOS_II.H
- 事件控制块数组

事件机制

- 事件机制提供给其他服务模块的内部接口：

- ▶ OS_EventTaskRdy()：因事件到来，使等待任务就绪
- ▶ OS_EventTaskWait()：因事件未到来，使任务等待
- ▶ OS_EventT0()：因事件超时，使等待任务就绪
- ▶ OS_EventWaitListInit()：事件等待队列初始化
- ▶ OS_InitEventList初始化事件控制块数组

● 提供给Application的外部接口

- ① OSInit(): 操作系统初始化
- ② OSIntEnter(): 进入中断处理前的准备
- ③ OSIntExit(): 离开中断处理后的收尾
- ④ OSSchedLock(): 禁止调度
- ⑤ OSSchedUnlock(): 允许调度
- ⑥ OSStart(): 操作系统开始运行 (多任务开始)
- ⑦ OSStatInit(): 操作系统统计相关的初始化
- ⑧ OSVersion(): 获得OS版本号

● 内部接口

- ❶ OS_Dummy(): 空操作
- ❷ OS_InitMisc(): 一些系统参数的初始化
- ❸ OS_InitRdyList(): 就绪队列初始化
- ❹ OS_InitTaskIdle(): idle任务初始化 (创建)
- ❺ OS_InitTaskStat(): 统计任务初始化 (创建)
- ❻ OS_InitTCBList(): 任务控制块数组初始化
- ❼ OS_Sched(): 调度函数
- ❽ OS_TaskIdle(): idle任务函数
- ❾ OS_TaskStat(): 统计任务函数
- ❿ OS_TCBInit(): 初始化一个TCB
- ⓫ ...

Outline

- 1 概述
- 2 目录分析
- 3 试运行和编译分析
- 4 uC/OS-II的功能解读
- 5 应用样例分析**
- 6 调度算法分析
- 7 移植分析

Application 样例 I

```
/*
*****
* MAIN
*****
*/
void main (void) {
    PC_DispcClrScr(DISP_FGND_WHITE + DISP_BGND_BLACK); /* Clear the screen */
    OSInit(); /* Initialize uC/OS-II */
    PC_DOSSaveReturn(); /* Save environment to return to DOS */
    PC_VectSet(uCOS, OSCtxSw); /* Install uC/OS-II's context switch vector */
    RandomSem = OSSemCreate(1); /* Random number semaphore */
    OSTaskCreate(TaskStart, (void *)0, &TaskStartStk[TASK_STK_SIZE - 1], 0);
    OSStart(); /* Start multitasking */
}
```

Application 样例 II

```
/* *****  
 * STARTUP TASK  
***** */  
  
void TaskStart (void *pdata) {  
#if OS_CRITICAL_METHOD == 3          /* Allocate storage for CPU status register */  
    OS_CPU_SR cpu_sr;  
#endif  
    char s[100];  
    INT16S key;  
  
    pdata = pdata;                    /* Prevent compiler warning */  
  
    TaskStartDispInit();              /* Initialize the display */  
  
    OS_ENTER_CRITICAL();  
    PC_VectSet(0x08, OSTickISR);      /* Install uC/OS-II' s clock tick ISR */  
    PC_SetTickRate(OS_TICKS_PER_SEC); /* Reprogram tick rate */  
    OS_EXIT_CRITICAL();  
  
    OSStatInit();                     /* Initialize uC/OS-II' s statistics */  
  
    TaskStartCreateTasks();           /* Create all the application tasks */
```

Application 样例 III

```
for (;;) {  
    TaskStartDisp();                /* Update the display */  
  
    if (PC_GetKey(&key) == TRUE) { /* See if key has been pressed */  
        if (key == 0x1B) {         /* Yes, see if it's the ESCAPE key */  
            PC_DOSReturn();        /* Return to DOS */  
        }  
    }  
  
    OSCtxSwCtr = 0;                /* Clear context switch counter */  
    OSTimeDlyHMSM(0, 0, 1, 0);    /* Wait one second */  
}  
}
```

Outline

- 1 概述
- 2 目录分析
- 3 试运行和编译分析
- 4 uC/OS-II的功能解读
- 5 应用样例分析
- 6 调度算法分析**
- 7 移植分析

调度算法分析

① 系统运行的级别

① 中断处理级别：不允许调度

- ★ 全局量OSIntNesting表示中断嵌套深度（使用一定要对称）
 - =0：不在中断处理级别
 - >0：在中断处理级别
- ★ OSIntEnter()：进入中断处理前的准备
- ★ OSIntExit()：离开中断处理后的收尾

② 任务运行级别：调度可以被禁止和允许

在任务运行级别，任务按照优先级来（抢占）调度

- ★ OSSchedLock()：禁止调度
- ★ OSSchedUnlock()：允许调度

调度算法分析

② 调度的时机

- ① 从中断处理级别进入任务运行级别时，若允许调度，则调度

★ 参见OSIntExit

- ② 任务因某些原因而等待或被唤醒时
以消息队列为例：

- ① OSQDe1()中，若销毁消息队列时有等待任务被迫就绪，
则因就绪队列发生改变有必要调度
- ② OSQPend()中，若任务因等待消息的到来而阻塞，
必须调度其他任务运行
- ③ OSQPost()、OSQPostFront()、OSQPostOpt()中，
若有任务因消息到来而被唤醒，必须调度

- ③ 新任务创建之后，若已经处于多任务状态，则重新调度
- ④ 任务被删除之后，需要重新调度
- ⑤ 一个任务恢复运行时，若该任务没有处于延迟执行状态，
需要重新调度
- ⑥ 当前任务被挂起时，需要调度一个其他任务运行
- ⑦ 任务优先级改变时，需要重新调度

调度算法分析

⑧ 调度函数

- ① 定位最高优先级 $OSPrioHighRdy$
- ② 若 $OSPrioHighRdy$ 不是当前任务优先级 $OSPrioCur$ ，
 - ① 修改最高优先级任务 $OSTCBHighRdy$ 指向新的最高优先级任务
 - ② 切换上下文

调度的关键：就绪队列的组织

调度的关键：就绪队列的组织

● 前提：

- ① 最多64个优先级，0~63；
数字越大，优先级越低
用户可以定义OS_LOWEST_PRIO，表明实际使用的最低优先级
约定：idle任务的优先级总是OS_LOWEST_PRIO，
统计任务（若使用）的优先级总是OS_LOWEST_PRIO-1
- ② 最多64个任务：任务和优先级一对一

● 思路：按**优先级位图**管理就绪任务

- ▶ 64位，8个字节，每个bit表示对应任务的就绪情况[是/否]
- ▶ 每个字节为一个group，一共8个group；
每个group使用一个bit表示该group中所有任务的就绪情况[有/没有]；
一共8个bit，即一个字节
- ▶ 即采用二级位图

```
OS_EXT INT8U OSRdyGrp;           /* Ready list group */  
OS_EXT INT8U OSRdyTbl[OS_RDY_TBL_SIZE]; /* Table of tasks which are ready to run */
```

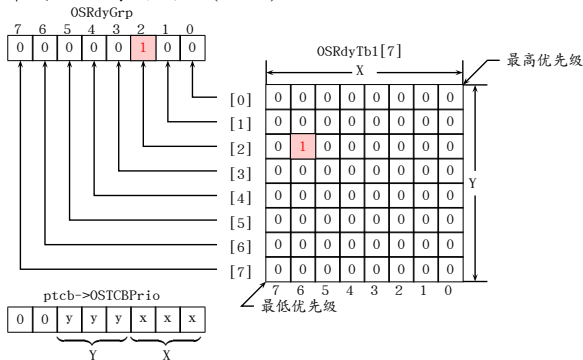
调度的关键：就绪队列的组织

● 算法：任务优先级 \leftrightarrow 位图（的bit位）

① 给定任务优先级A，A的范围[0~63]，有效位为低端6位，则：

★ X=A的末尾3位，表示为 $X=A\&0x7$ ；

Y=A的中间3位，表示为 $Y=(A>>3)\&7$



以任务优先级22为例：

优先级22=00 010 110b

因此OSRdyTb1[2]中第6位置1，OSRdyGrp中第2位置1

对应的掩码：uC/OS-II采用查表法（具体参见OSMapTb1[]）

调度的关键：就绪队列的组织

- 算法：任务优先级 \leftrightarrow 位图（的bit位）

- ② 给定位图（系统中的当前位图），得到最新的最高优先级

- ① 根据OSRdyGrp的值确定最高优先级所在的组号即Y值；
 - ② 根据最高优先级组的值确定最高优先级的X值；
 - ③ 两者组合成最高优先级。

uC/OS-II的方法：查表法（具体参见OSUnMapTbl[]）

关于性能的思考

- 硬实时操作系统应该满足哪些条件？

Outline

- 1 概述
- 2 目录分析
- 3 试运行和编译分析
- 4 uC/OS-II的功能解读
- 5 应用样例分析
- 6 调度算法分析
- 7 移植分析**

移植分析

- 体系结构相关的移植：3个关键的文件

- ▶ 阅读下列文件（以Ix86L为例）

- ① OS_CPU_A.ASM

- ② OS_CPU_C.C

- ③ OS_CPU.H

- 板极支持包

Thanks !

The end.