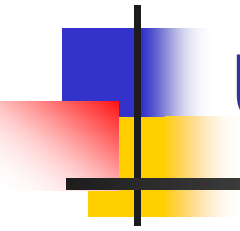
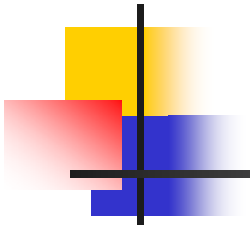


uCOS-II Report (1)

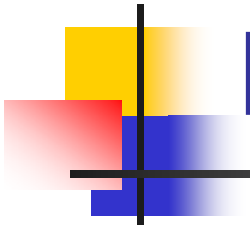
Yihe.Chen

yihect@gmail.com



內容

- Feature
- Compare with other systems
- Tasks in uCOS-II
- Interrupts in uCOS-II
- Clock Tick in uCOS-II
- Porting of uCOS-II
- Q & A



Feature

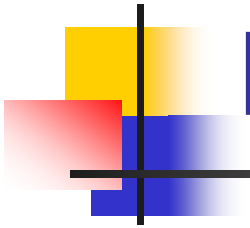
- Simple enough in opinion
  - 没有文件系统(官方有个 μC/FS)
  - 没有复杂的内存管理（VMM）
  - 没有Kernel /User Mode的困扰
  - 任务调度也不复杂（By Priority Only）
  - 不存在驱动模型（As A Disadvantage）

Feature

- Preemptive
  - 完全的抢占式内核，意味着它总是运行一系列可运行任务中“最紧迫”的那个任务
- Multiasking
  - 版本2.86 最高能支持256个任务，系统占据其中的两个用做 Idle task 和 Statistic task
  - 不支持轮转调度，所以每个任务都有唯一的一个优先级与其对应，并支持256个优先级

Feature

- Portable
  - 非常小，大概只有 5500 多行
  - 大部分代码用ANSI C写成
  - 很少部分平台相关的代码则是用汇编语言写的
- Scalable
  - 可根据需要对其进行剪裁，只使用其提供的部分服务，能减少宝贵的ROM/RAM用量
  - 通过条件编译的方式达到



Feature

- Task Stacks
  - 允许每个任务有自己的堆栈，各个任务的堆栈大小可以完全不同，能减少资源的使用
  - 有堆栈检查功能（Stack-checking feature），能帮助我们确定每个任务实际需要多大的堆栈空间

Feature

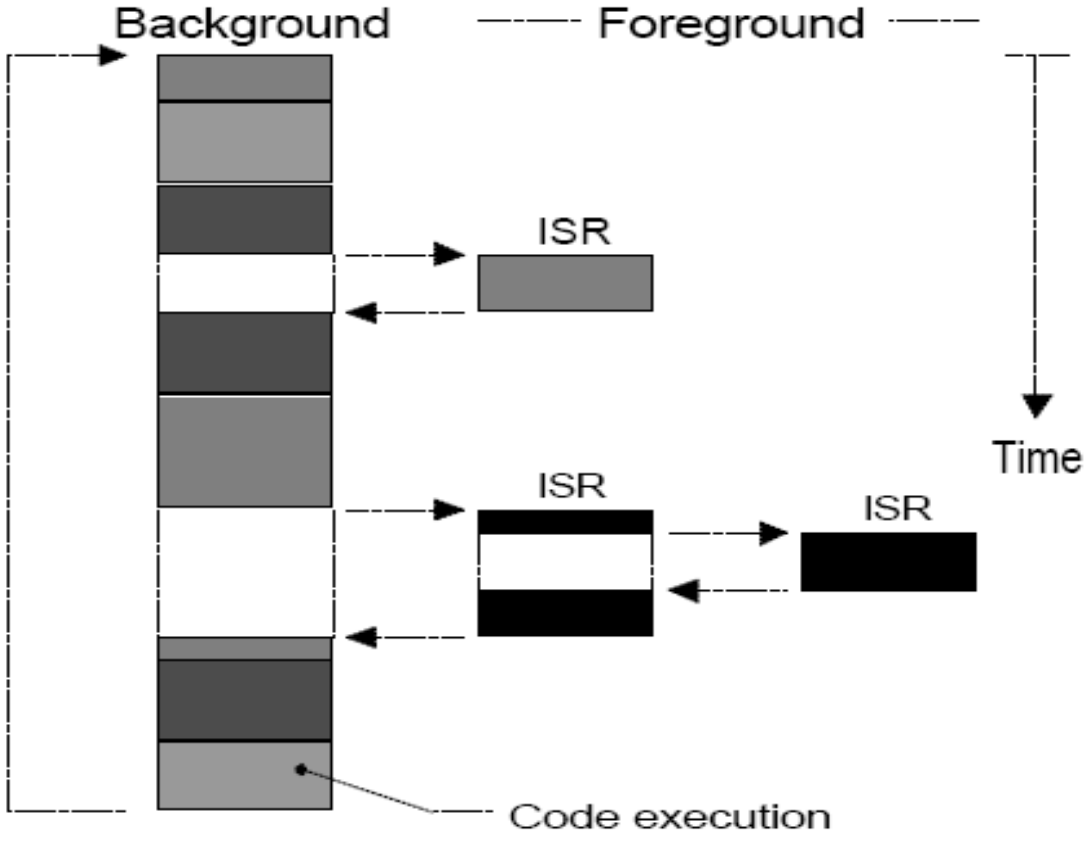
- Deterministic
  - 所有的函数和服务的执行时间都是确定的
  - 除了OSTimeTick()和一些事件标志服务之外，其他所有服务的执行时间都不依赖于任务数量的大小
- Services
  - 提供众多的服务，比方 semaphores, event flags, message mailboxes, message queues, 任务管理，时间管理等

Compare with other sys

- 超级循环系统（前后台系统）
  - 后台是一个无穷循环，处理所有业务逻辑；前台由ISRs处理所有的外部事件（BOT）
  - 因为ISRs不应该处理太多事情，而一般只是用于设置某些标志以代表不同事件的发生。
  - 这些事件如果要想真正地被处理器处理，则要等到后台的大循环运行到某一点时才行。
  - 降低响应速度，而且循环中每两次到达同一个点的时间间隔并不确定。

Compare with other sys

- 超级循环系统（前后台系统）



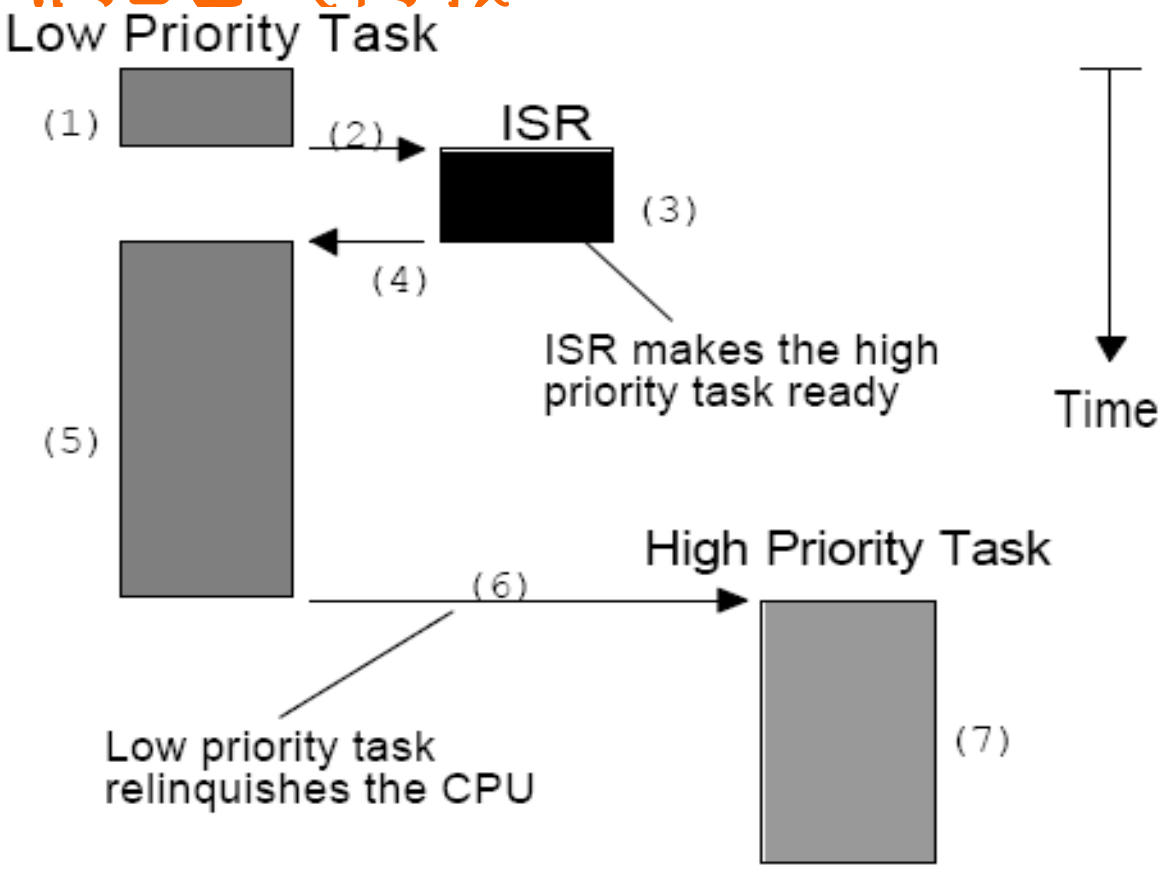
Compare with other sys

- 非抢占式内核
  - 协同式多任务。系统如果想把CPU交由其他任务使用，必须经由当前运行着的任务同意才行。
  - 允许使用Non-Reentrant 的函数，但必须保证在执行这样的函数期间不主动让出CPU
  - 很少的内核系统是非抢占式的

Non-Reentrant 函数是指能同时被多个任务使用，
而不需要担心出现数据遭破坏的情况

Compare with other sys

- 非抢占式内核

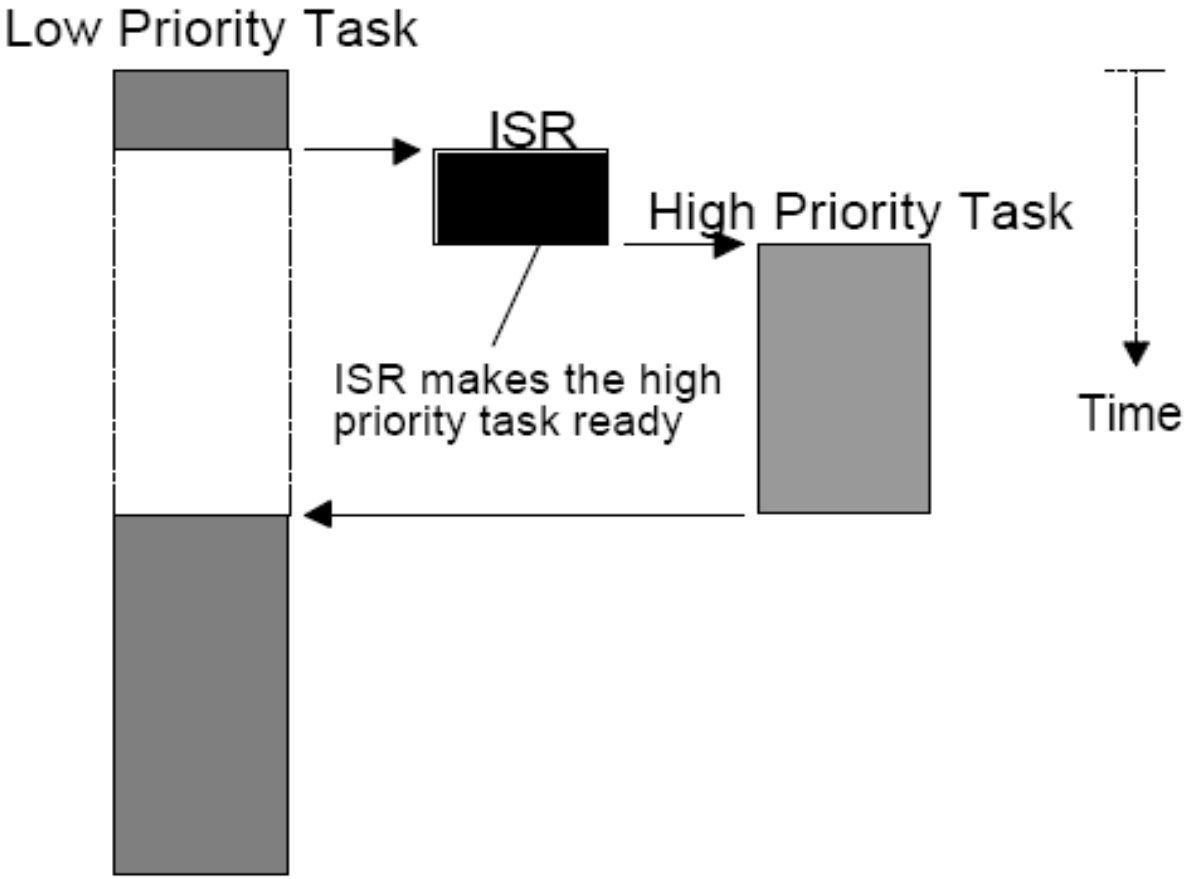


Compare with other sys

- 抢占式内核
  - 强迫式多任务。运行过程中只要出现一个优先级更高的可运行任务，系统就强行终止当前任务的执行，转而执行更高优先级的任务
  - 不应该使用Non-Reentrant 的函数，大量使用各种任务间同步机制
  - 大多数的内核系统都是抢占式的，ucos-ii 也一样

Compare with other sys

- 抢占式内核



Compare with other sys

- 函数可重入性

```
void strcpy(char *dest, char *src)
{
    while (*dest++ = *src++) ;
    *dest = NUL;
}
```

```
int Temp;
void swap(int *x, int *y)
{
    Temp = *x;
    *x = *y;
    *y = Temp;
}
```

Tasks in uCOS-II

- Idle Task
  - 如果没有其他任务可以运行，系统就运行 Idle Task. Idle Task 总是在最低优先级上（OS_LOWEST_PRIO）运行
- Statistic Task
  - 用于提供运行时统计信息，可通过 OS_TASK_STAT_EN 配置项去掉它

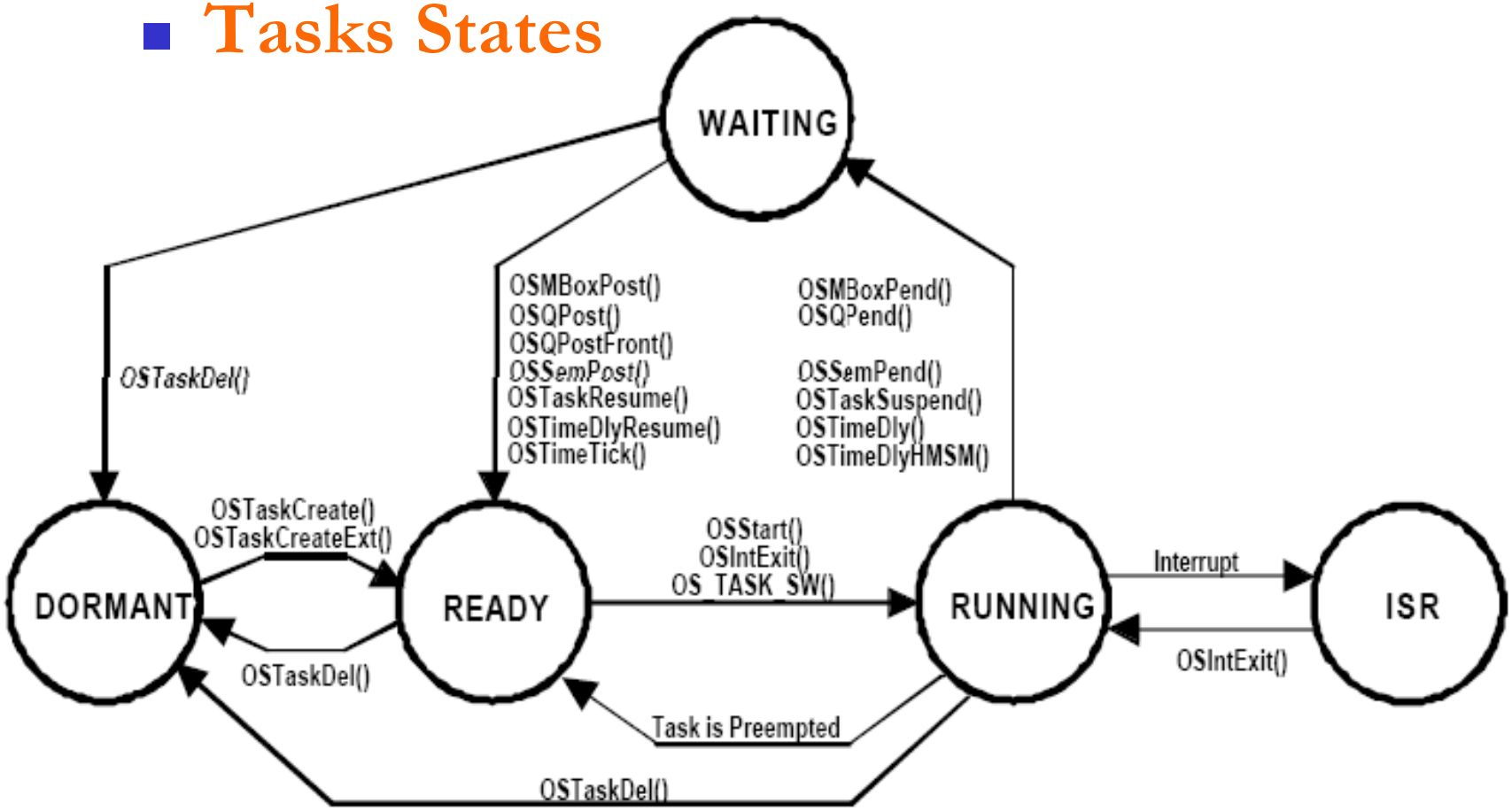
Tasks in uCOS-II

- User Tasks

```
void YourTask (void *pdata)
{
   for (;;) {
   /* USER CODE
   Call one of uC/OS-II's services:
   OSMboxPend();
   OSQPend();
   OSSemPend();
   OSTaskDel(OS_PRIO_SELF);
   OSTaskSuspend(OS_PRIO_SELF);
   OSTimeDly();
   OSTimeDlyHMSM();
    */
   }
}
```

Tasks in uCOS-II

- Tasks States



Tasks in uCOS-II

- Tasks Scheduling in task level

```
void  OS_Sched (void)
{
   OS_CPU_SR  cpu_sr = 0;

   OS_ENTER_CRITICAL();
   if (OSIntNesting == 0) {
      if (OSLockNesting == 0) {
         OS_SchedNew();
         if (OSPrioHighRdy != OSPrioCur) {
            OSTCBHighRdy = OSTCBPrioTbl[OSPrioHighRdy];

            OSCtxSwCtr++;
            OS_TASK_SW();
         }
      }
   }
   OS_EXIT_CRITICAL();
}
```

软中断或陷阱指令

作为 宏 在 Os_cpu.h 文件中定义

```
void OSCtxSw(void)
{
    Save processor registers;
    Save the current task's stack pointer
        into the current task's OS_TCB:
    OSTCBCur->OSTCBStkPtr = Stack pointer;
    Call user definable OSTaskSwHook();
    OSTCBCur = OSTCBHighRdy;
    OSPrioCur = OSPrioHighRdy;
    Get the stack pointer of the task to resume:
    Stack pointer = OSTCBHighRdy->OSTCBStkPtr;
    Restore all processor registers
        from the new task's stack;
    Execute a return from interrupt instruction;
}
```

在 OS_cpu_a.asm 文件中定义

Tasks in uCOS-II

- Tasks Scheduling in task level
  - 任务调度的时间独立于任务数量
  - 对于系统中的每一个处于“READY”状态的任务来说，它的堆栈中保存的内容就好象刚刚发生了一次中断一样，所有的寄存器都已经被压到堆栈中。所以在 OcCtxSw() 汇编函数中只需从堆栈中恢复这些寄存器，并执行一条中断返回指令即可
  - 在整个 OS_Sched() 函数中中断都应该是被关闭的，OcCtxSw() 也必须保证中断不被打开

Tasks in uCOS-II

- Tasks Scheduling in interrupt level
  - 由于是抢占式内核，uCOS-II会在每次中断完成的时候去检查是否有更高优先级的任务进入READY状态：

    如果有，则调度运行这个更高优先级的任务；

    如果没有，则返回到原先被中断的那个任务继续允许
  - 中断级别的任务调度由 汇编函数OSIntCtxSw() 完成，移植时必须实现该函数

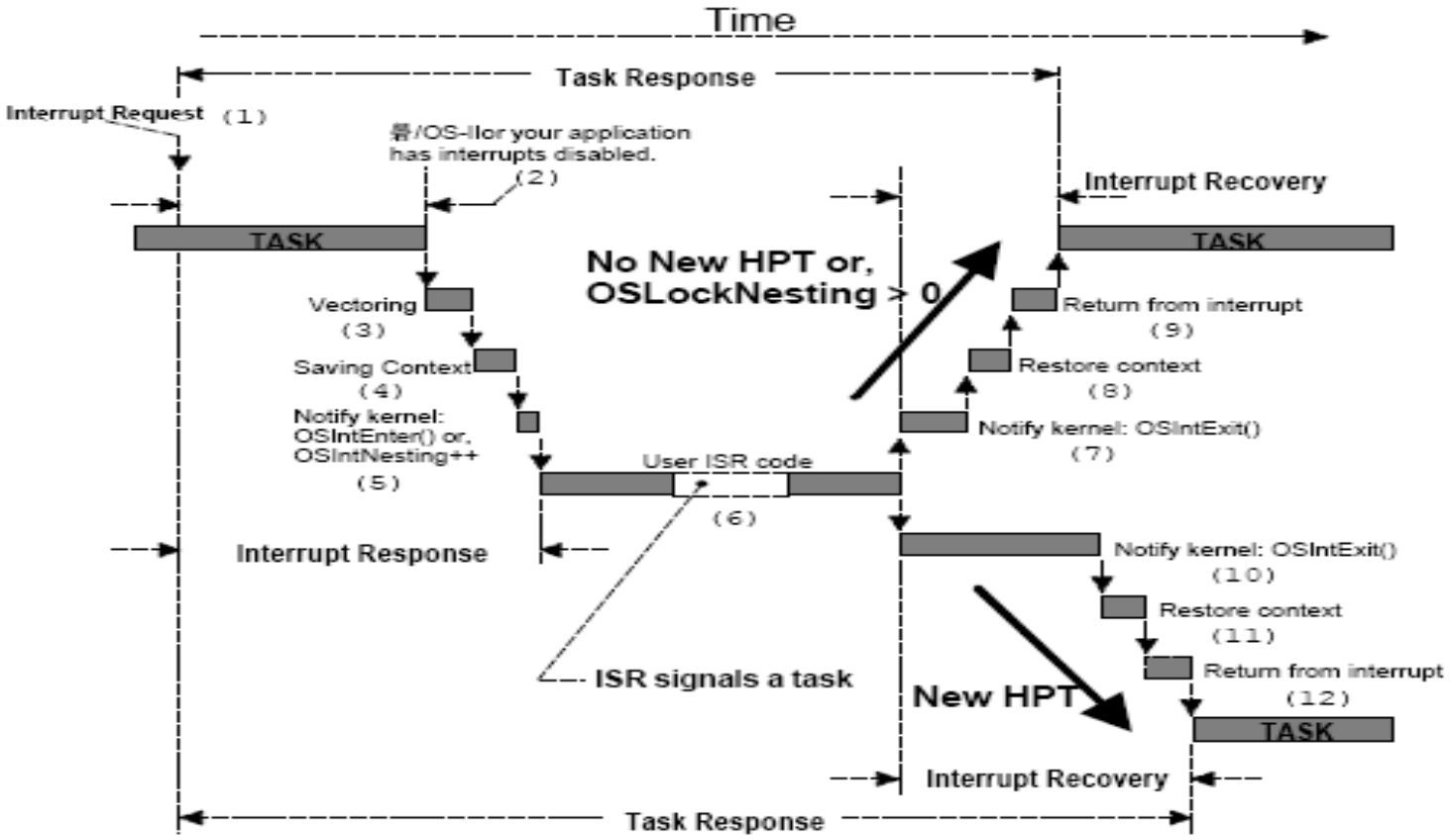
Interrupts in uCOS-II

- ISRs demanded in uCOS-II
  - 要求必须是汇编写的，但是可以以内联汇编的形式存在于.C文件中。ISRs内部可以调用C函数
  - Pseudo Code Of ISRs

```
YourISR:
    Save all CPU registers;                                    (1)
    Call OSIntEnter() or, increment OSIntNesting directly;     (2)
    Execute user code to service ISR;                          (3)
    Call OSIntExit();                                          (4)
    Restore all CPU registers;                                 (5)
    Execute a return from interrupt instruction;               (6)
```

Interrupts in uCOS-II

- Interrupt process



Interrupts in uCOS-II

- OSIntEnter()
  - 告知 uCOS-II 正在进行中断服务
  - 递增中断嵌套级数的简单封装，移植的时候可以不用该函数，而是将递增动作重新安排到别的函数里，用产生的新函数面来代替 OSIntEnter()

```
void  OSIntEnter (void)
{
    if (OSRunning == OS_TRUE) {
        if (OSIntNesting < 255u) {
            OSIntNesting++;
        }
    }
}
```

Interrupts in uCOS-II

- OSIntExit()
  - 与OS_Sched()相似，除了以下两点：

    a, OSIntExit() 要递减嵌套级数

    b, OSIntExit() 调用汇编函数OSIntCtxSw() 调度新任务，而不像 OS_Sched()那样去用宏 OS_TASK_SW() 发出软中断进行任务调度

    原因何在？有二：

    1, 因为任务切换的部分工作（保存被中断任务的寄存器上下文）已经在进入 ISR 的时候完成掉了;

    2, 在执行到 OSIntCtxSw() 内部代码的时候，被中断任务的堆栈内容已不是我们想要的，需要依赖于OSIntCtxSw()来调整堆栈指针使得被中断任务的堆栈变成我们想要的样子。

Interrupts in uCOS-II

- OSIntExit()
  - 与调用OS_Sched()进行task level时一样，调用OSIntExit()（步骤4）和步骤5/6进行任务上下文的恢复都要求中断是被关闭的，否则可能陷入死循环

注意：有些处理器为了支持 Nestting of Interrupts 在保存寄存器上下文的时候可能会同时打开中断，所以在进行新任务上下文恢复之前，应该想办法关闭中断，以使得任务的切换能顺利进行。
移植的时候要考虑这一点。

Interrupts in uCOS-II

- OSIntExit() Calls OSIntCtxSw to switch

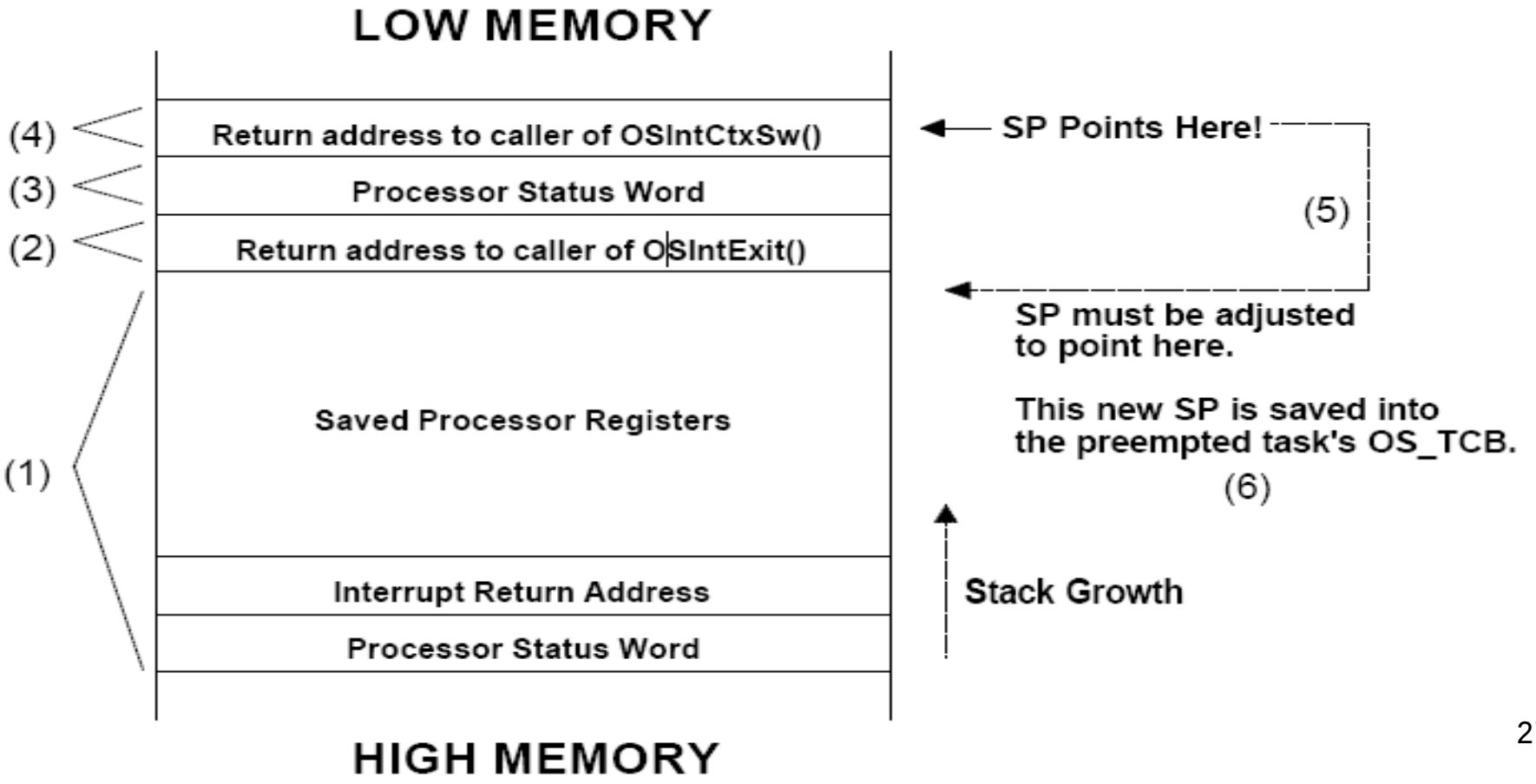
```
void  OSIntExit (void)                          (1)
{
    OS_CPU_SR  cpu_sr = 0;

    if (OSRunning == OS_TRUE) {
        OS_ENTER_CRITICAL();                    (2)
        if (OSIntNesting > 0) {
            OSIntNesting--;                     (3)
        }
        if (OSIntNesting == 0) {
            if (OSLockNesting == 0) {
                OS_SchedNew();
                if (OSPrioHighRdy != OSPrioCur) {
                    OSTCBHighRdy  = OSTCBPrioTbl[OSPrioHighRdy];

                    OSTCBHighRdy->OSTCBCtxSwCtr++;
                    OSCtxSwCtr++;

                    OSIntCtxSw();               (4)
                }
            }
        }
        OS_EXIT_CRITICAL();
    }
}
```

在 文件 Os_cpu_a.asm 文件中定义

```
void OSIntCtxSw(void)
{
    Adjust the stack pointer to remove calls to:
      OSIntExit(),
      OSIntCtxSw() and possibly the push of the
        processor status word;

    Save the current task's stack pointer into
        the current task's OS_TCB:
      OSTCBCur->OSTCBStkPtr = Stack pointer;

    Call user definable OSTaskSwHook();
    OSTCBCur = OSTCBHighRdy;
    OSPrioCur = OSPrioHighRdy;

    Get the stack pointer of the task to resume:
      Stack pointer = OSTCBHighRdy->OSTCBStkPtr;

    Restore all processor registers from the new
        task's stack;
    Execute a return from interrupt instruction;
}
```

Interrupts in uCOS-II

- Adjust SP of old task in OSIntCtxSw()



Clock Tick in uCOS-II

- Usage of Clock Tick
  - 时钟滴答仅用于下列两点，uCOS-II不用时钟滴答来实现任务的分时调度：

    1，支持任务delay 多少个 Ticks, 滴答数全部用完，该任务即可被重新调度；

    2，支持任务在等待某一事件时的超时机制，一旦达到 Timer 的限制，该任务即可被重新调度。

    注意：如果一个任务是被“显式”挂起的，该任务不会被uCOS-Ii恢复，直到别的任务或者 ISRs “显式”唤醒为止。
  - 时钟滴答的中断使能时机问题（后续报告中再说）

Interrupts in uCOS-II

- OSTickISR()
  - uCOS-II时钟滴答依赖于在一个定时器ISR中调用 OSTimeTick() 完成：

```
void OSTickISR(void)
{
    Save processor registers;
    Call OSIntEnter() or increment OSIntNesting;
    Call OSTimeTick();
    Call OSTimeTick();
    Restore processor registers;
    Execute a return from interrupt instruction;
}
```

Interrupts in uCOS-II

- OSTimeTick ()

```
void  OSTimeTick (void)
{
    //....
    OSTimeTickHook();

    OS_ENTER_CRITICAL();
    OSTime++;
    OS_EXIT_CRITICAL();

    if (OSRunning == OS_TRUE) {
         //....Other Code

        ptcb = OSTCBList;
        while (ptcb->OSTCBPrio != OS_TASK_IDLE_PRIO) {
            OS_ENTER_CRITICAL();
            if (ptcb->OSTCBDly != 0) {
                if (--ptcb->OSTCBDly == 0) {
                    if ((ptcb->OSTCBStat & OS_STAT_PEND_ANY) != OS_STAT_RDY) {
                        ptcb->OSTCBStat  &= ~(INT8U)OS_STAT_PEND_ANY;          /* Yes, Clear status flag   */
                        ptcb->OSTCBStatPend = OS_STAT_PEND_TO;                 /* Indicate PEND timeout    */
                    } else {
                        ptcb->OSTCBStatPend = OS_STAT_PEND_OK;
                    }

                    if ((ptcb->OSTCBStat & OS_STAT_SUSPEND) == OS_STAT_RDY) {  /* Is task suspended?       */
                        OSRdyGrp               |= ptcb->OSTCBBitY;             /* No,   Make ready         */
                        OSRdyTbl[ptcb->OSTCBY] |= ptcb->OSTCBBitX;
                    }
                }
            }
            ptcb = ptcb->OSTCBNext;
            OS_EXIT_CRITICAL();
        }
    }
}
```

Interrupts in uCOS-II

- OSTimeTick()
  - 该函数的执行时间完全取决于系统中的任务数。
  - 一般要求在ISR中不能做太多耗时的事情。所以在应用的时候，如果任务数量比较多，而且又希望得到比较高的响应要求，可以考虑把 OSTimeTick() 放在一个新创建的最高优先级的任务里面，同时通过Semaphore 或者 Message Box在定时器ISR里面唤醒该任务完成 OSTimeTick()

Porting of uCOS-II

- Requiements
  - 必须有能产生可重入代码的C编译器
  - 必须能够直接在C 或 inline 汇编里面 打开和关闭中断
  - 处理器必须能够支持中断，其中必须有一个能定时触发的中断可用（10/100Hz）
  - 处理器必须支持硬件堆栈，并且该堆栈必须能容纳Kbytes数量级的内容
  - 处理器必须提供Load/Store指令，用于在堆栈或内存里保存SP或其他CPU 寄存器的值

Porting of uCOS-II

- Something to be known
  - 处理器的中断处理

    1，如何打开和关闭中断

    2， cpu寄存器上下文的保存过程（保存次序）

    3，中断的返回过程（如何取返回地址，如何恢复cpu寄存器上下文）

    4，如何支持软中断和定时器中断

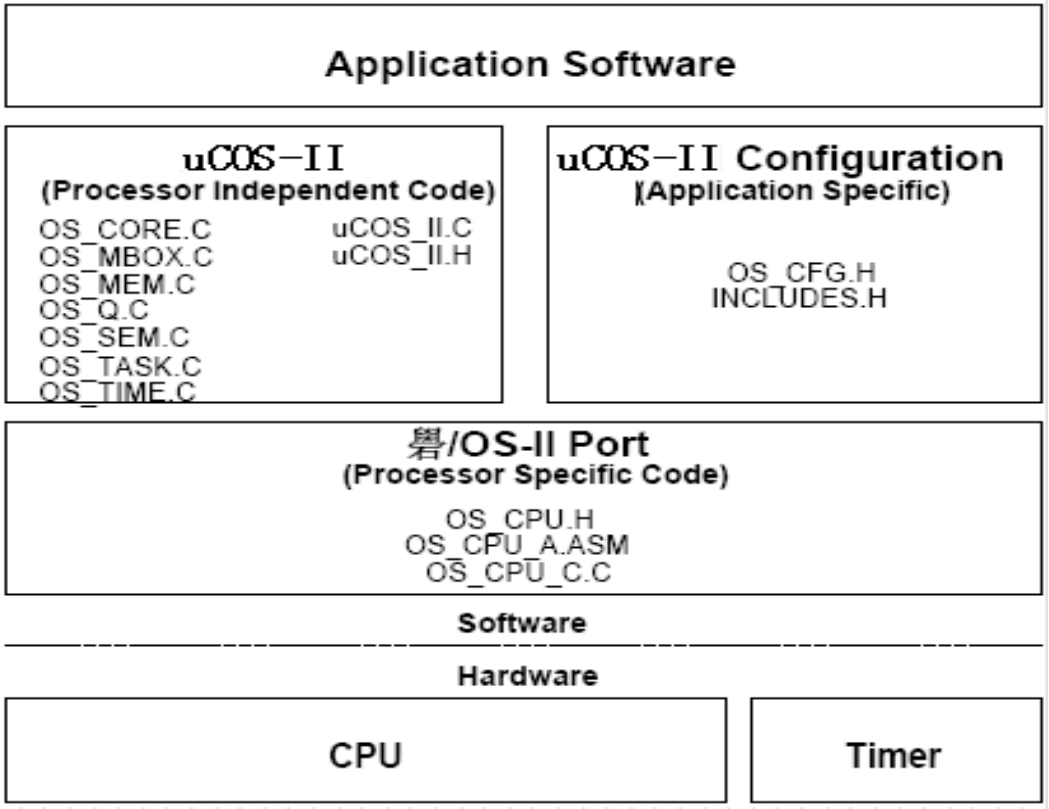
    5，如何支持中断的嵌套
  - 编译器的C Run-time Mode

    1，函数调用的堆栈处理（C函数之间，C函数与汇编例程之间）

    2，函数参数和返回值的传递

Porting of uCOS-II

- Something to be known
  - uC0S-II 的整体结构



Porting of uCOS-II

- Roadmap
  - 设置OS_cpu.h 中的常量宏
  - 在OS_cpu.h中定义一些数据类型
  - 在OS_cpu.h中定义三个宏
  - 在OS_cpu_c.c中定义十个函数
  - 在OS_cpu_a.asm定义四个函数

Porting of uCOS-II

- 设置OS_cpu.h中的常量宏

/*堆栈增长方向*/

- #define OS_STK_GROWTH

/* 打开-关闭中断的方式*/

- #define OS_CRITICAL_METHOD

Porting of uCOS-II

- 在OS_cpu.h中定义一些数据类型

```
typedef unsigned char    BOOLEAN;
typedef unsigned char    INT8U;     /* Unsigned 8 bit quantity */
typedef signed char      INT8S;     /* Signed 8 bit quantity */
typedef unsigned int     INT16U;    /* Unsigned 16 bit quantity */
typedef signed int       INT16S;    /* Signed 16 bit quantity */
typedef unsigned long    INT32U;    /* Unsigned 32 bit quantity */
typedef signed long      INT32S;    /* Signed 32 bit quantity */
typedef float            FP32;      /* Single precision floating point */
typedef double           FP64;      /* Double precision floating point */
typedef unsigned int     OS_STK;    /* Each stack entry is 16-bit wide */
```

Porting of uCOS-II

- 在OS_cpu.h中定义三个宏

  /* 发出软中断指令 */

  - #define OS_TASK_SW()

  /* 关闭中断 */

  - #define OS_ENTER_CRITICAL()

  /* 打开中断 */

  - #define OS_EXIT_CRITICAL()

Porting of uCOS-II

- 在OS_cpu_c.c中定义十个函数

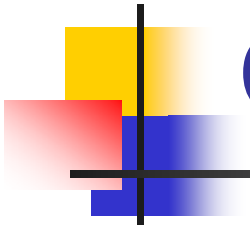
```
void     OSTaskCreateHook        ()
void     OSTaskDelHook           ()
void     OSTaskSwHook            ()
void     OSTCBInitHook           ()
void     OSInitHookBegin         ()
void     OSInitHookEnd           ()
void     OSTaskStatHook          ()
void     OSTimeTickHook          ()
void     OSTaskIdleHook          ()
OS_STK  *OSTaskStkInit           ()
```

注意：只有最后一个任务堆栈初始化的函数是必须的，其他函数可以为空

Porting of uCOS-II

- 在OS_cpu_a.asm定义四个函数

```
OSStartHighRdy()     /* 被OSStart()调用开始最高优先级任务的运行      */

OSCtxSw()            /* 被映射成软件中断的 ISR，实现任务切换         */
OSIntCtxSw()         /* 被OSIntExit()调用以实现中断返回时的任务切换 */

OSTickISR()          /* 定时器中断，内部需要调用 OSTimeTick()        */
                     /* 以实现ClockTick的管理                        */
```



Q & A

THE END

Thanks a lot, any suggestion?