Task Management - uC/OSII

Yu-Han Li

OSNET Lab. of C.S.

National Chung Hsing University

Contents – 1/2

- OSTaskCreate()、OSTaskCreateExt()
- Task Stacks
- OSTaskStkChk()
- OSTaskDel()、OSTaskDelReq()

Contents – 2/2

- OSTaskChangePrio()
- OSTaskSuspend()
- OSTaskResume()
- OSTaskQuery()

Two kind of task structure

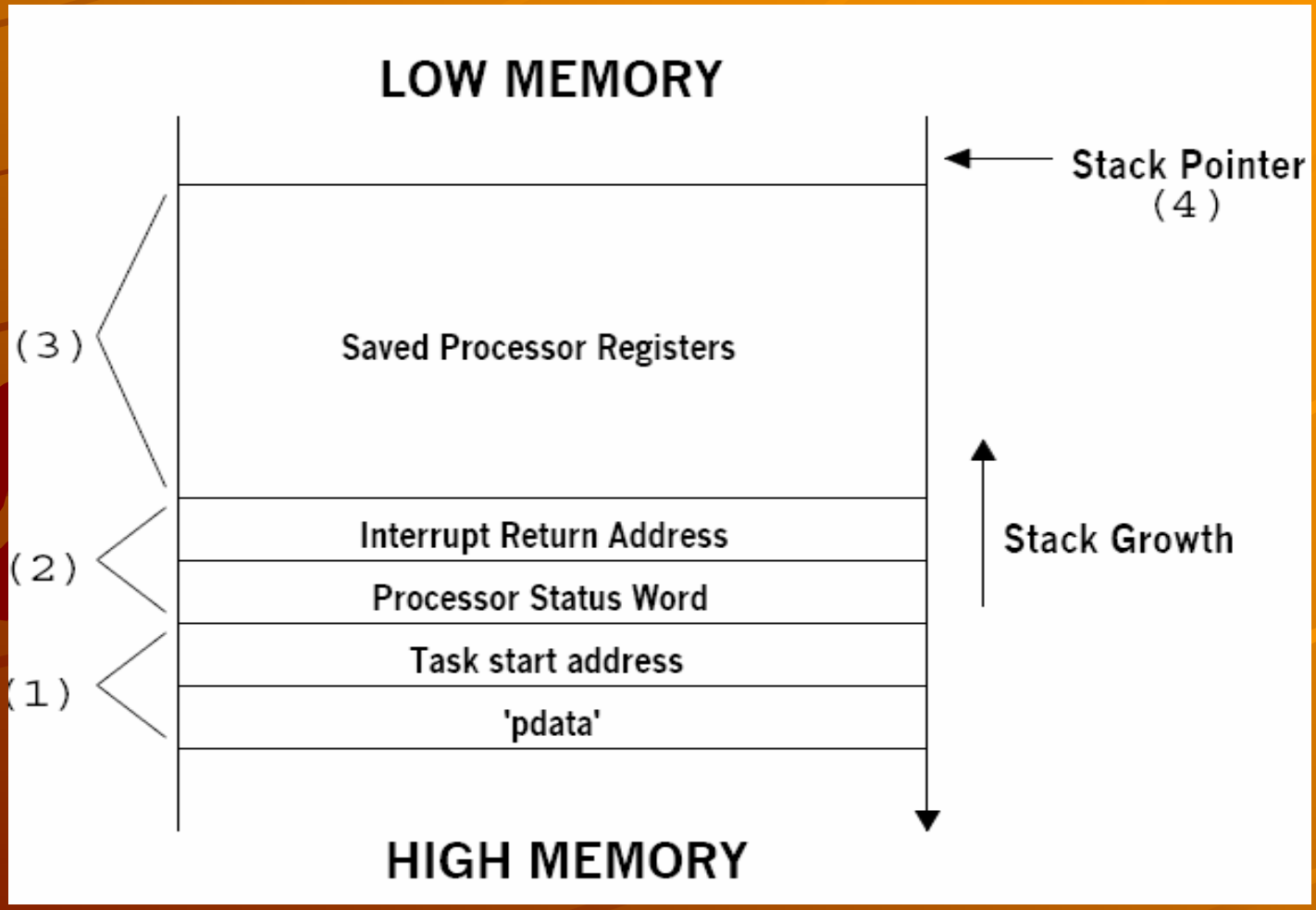
```
void YourTask (void *pdata)
{
for (;;) {
    /* USER CODE */
Call one of uC/OS-II's service:
    OSMboxPend();
    OSQPend();
    OSSemPend();
    OSTaskDel(OS_PRIO_SELF);
    OSTaskSuspend(OS_PRIO_SELF);
    OSTimeDly();
    OSTimeDlyHMSM();
    /* USER CODE */
}
}
```

```
void YourTask (void *pdata)
{
    /* USER CODE */
  OSTaskDel(OS_PRIO_SELF);
}
```

OSTaskCreate() – 1/3

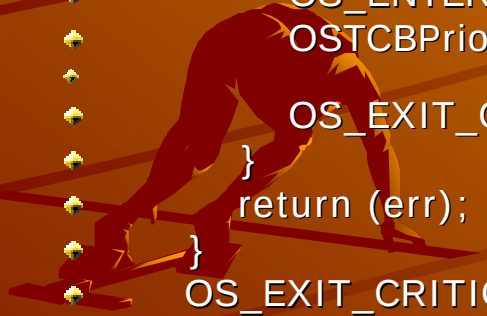
```
INT8U  OSTaskCreate (void (*task)(void *pd), void *pdata, OS_STK *ptos, INT8U
prio){
#if OS_CRITICAL_METHOD == 3                  /* Allocate storage for CPU status register */
    OS_CPU_SR  cpu_sr;
#endif
    OS_STK    *psp;
    INT8U      err;
#if OS_ARG_CHK_EN > 0
    if (prio > OS_LOWEST_PRIO) {             /* Make sure priority is within allowable range*/
        return (OS_PRIO_INVALID);}
#endif
    OS_ENTER_CRITICAL();
    if (OSTCBPrioTbl[prio] == (OS_TCB *)0) { /* Make sure task doesn't already exist at this priority  */
        OSTCBPrioTbl[prio] = (OS_TCB *)1;    /* Reserve the priority to prevent others from doing ...  */
                                             /*... the same thing until task is created.*/
        OS_EXIT_CRITICAL();
        psp = (OS_STK *)OSTaskStkInit(task, pdata, ptos, 0);
        err = OS_TCBInit(prio, psp, (OS_STK *)0, 0, 0, (void *)0, 0);
```

OSTaskCreate()--- OSTaskStkInit() – 2/3



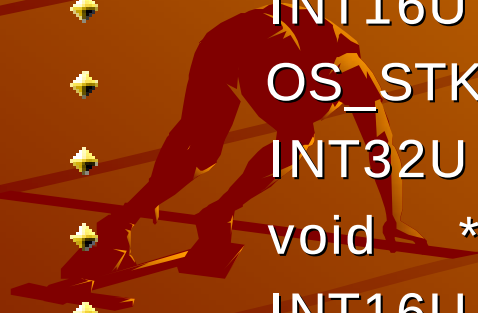
OSTaskCreate() – 3/3

```
if (err == OS_NO_ERR) {
        OS_ENTER_CRITICAL();
        OSTaskCtr++;                    /* Increment the #tasks counter */
        OS_EXIT_CRITICAL();
        if (OSRunning == TRUE) {
                                        /* Find highest priority task if multitasking has started */
            OS_Sched();
        }
    } else {
        OS_ENTER_CRITICAL();
        OSTCBPrioTbl[prio] = (OS_TCB *)0;
                                        /* Make this priority available to others */
        OS_EXIT_CRITICAL();
    }
    return (err);
}
OS_EXIT_CRITICAL();
return (OS_PRIO_EXIST);
}
```



OSTaskCreateExt() – 1/3

```
INT8U  OSTaskCreateExt (
       void   (*task)(void *pd),
       void    *pdata,
       OS_STK  *ptos,
       INT8U    prio,
       INT16U   id,
       OS_STK  *pbos,
       INT32U   stk_size,
       void    *pext,
       INT16U   opt
       )
```

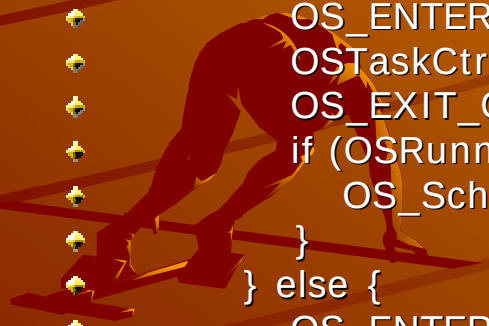


OSTaskCreateExt() – 2/3

```
{
#if OS_CRITICAL_METHOD == 3    /* Allocate storage for CPU status register */
    OS_CPU_SR  cpu_sr;
#endif
    OS_STK    *psp;
    INT8U      err;

#if OS_ARG_CHK_EN > 0
    if (prio > OS_LOWEST_PRIO) { /* Make sure priority is within allowable range */
        return (OS_PRIO_INVALID);
    }
#endif
    OS_ENTER_CRITICAL();
    if (OSTCBPrioTbl[prio] == (OS_TCB *)0) {
                                                /*Make sure task doesn't already exist at this priority  */
        OSTCBPrioTbl[prio] = (OS_TCB *)1;       /* Reserve the priority to prevent others from doing ..*/
                                                /* ... the same thing until task is created. */
        OS_EXIT_CRITICAL();

        if (((opt & OS_TASK_OPT_STK_CHK) != 0x0000) ||
                                                 /* See if stack checking has been enabled   */
            ((opt & OS_TASK_OPT_STK_CLR) != 0x0000)) {
                                                 /* See if stack needs to be cleared  */
```

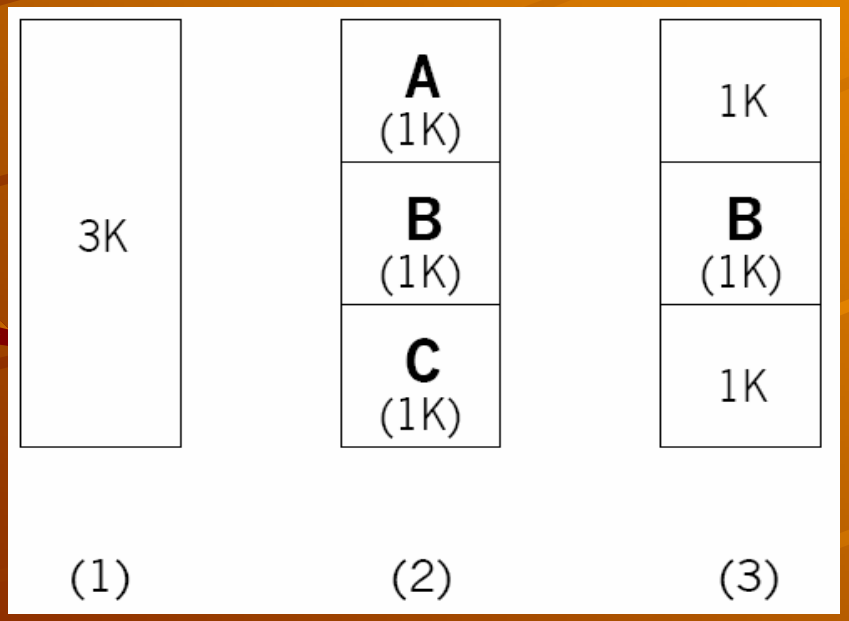
OSTaskCreateExt() – 3/3

```
            #if OS_STK_GROWTH == 1
            (void)memset(pbos, 0, stk_size * sizeof(OS_STK));
            #else
            (void)memset(ptos, 0, stk_size * sizeof(OS_STK));
            #endif }
        psp = (OS_STK *)OSTaskStkInit(task, pdata, ptos, opt);
                                                /* Initialize the task's stack */
        err = OS_TCBInit(prio, psp, pbos, id, stk_size, pext, opt);
        if (err == OS_NO_ERR) {
            OS_ENTER_CRITICAL();
            OSTaskCtr++;                        /* Increment the #tasks counter */
            OS_EXIT_CRITICAL();
            if (OSRunning == TRUE) {            /* Find HPT if multitasking has started */
                OS_Sched();
            }
        } else {
            OS_ENTER_CRITICAL();
            OSTCBPrioTbl[prio] = (OS_TCB *)0;   /* Make this priority avail. to others */
            OS_EXIT_CRITICAL();
        }
        return (err);
    }
    OS_EXIT_CRITICAL();  return (OS_PRIO_EXIST);
}
```



Task Stacks – 1/3

- Static --- OS_STK Stackname[stack_size]
- Dynamic ---by malloc()->Fragmentation



Task Stacks – 2/3

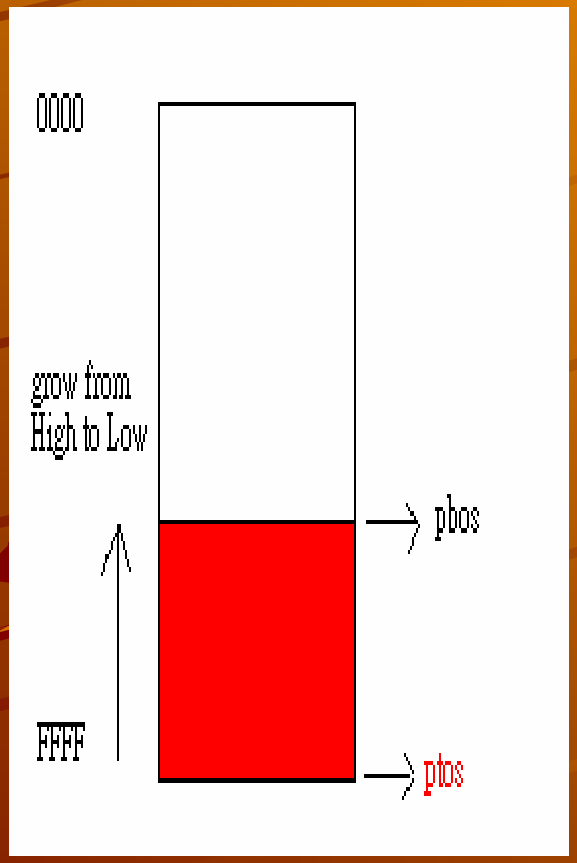
- Stack grows from **Low to High** memory:

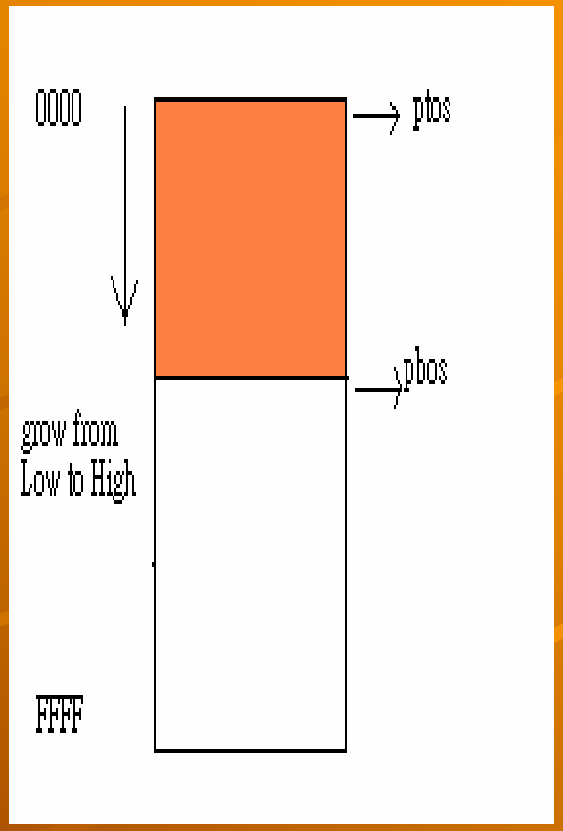
  OS_STK TaskStk[TASK_STK_SIZE];
  OSTaskCreate(task,pdata, &TaskStk[0], prio);

- Stack grows from **High to Low** memory:

  OS_STK TaskStk[TASK_STK_SIZE];
  OSTaskCreate(task,pdata, &TaskStk[Task_STK_SIZE-1], prio);

Task Stacks – Direction of stack grow – 3/3





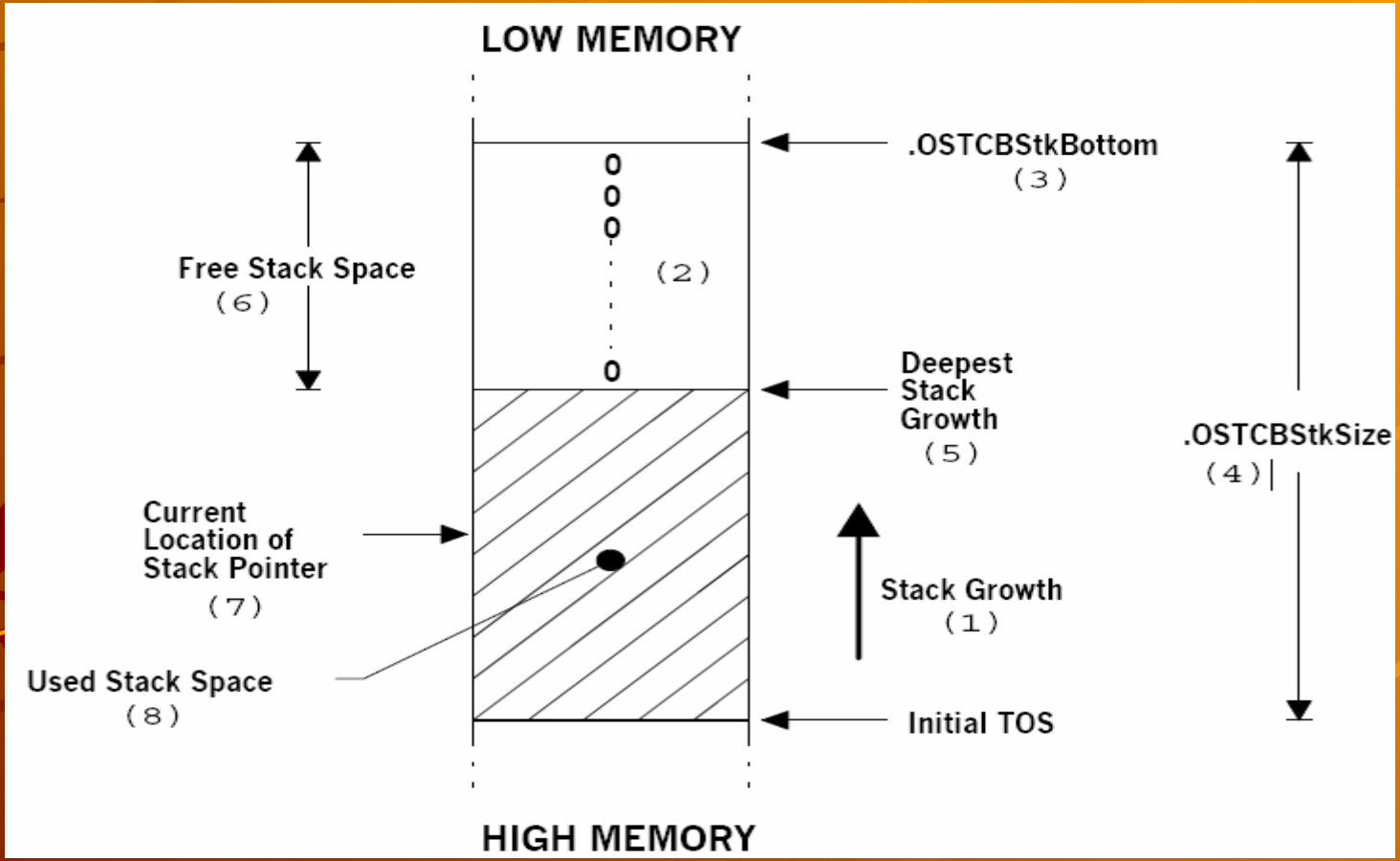
OSTaskStkChk() – 1/3

```
INT8U  OSTaskStkChk (INT8U prio, OS_STK_DATA *pdata){
#if OS_CRITICAL_METHOD == 3     /*Allocate storage for CPU status register */
    OS_CPU_SR  cpu_sr;
#endif
    OS_TCB    *ptcb;
    OS_STK    *pchk;
    INT32U     free;
    INT32U     size;
#if OS_ARG_CHK_EN > 0
    if (prio > OS_LOWEST_PRIO && prio != OS_PRIO_SELF) {
                                            /* Make sure task priority is valid    */
        return (OS_PRIO_INVALID);  }
#endif
    pdata->OSFree = 0;                      /* Assume failure, set to 0 size  */
    pdata->OSUsed = 0;
    OS_ENTER_CRITICAL();
    if (prio == OS_PRIO_SELF) {             /* See if check for SELF */
        prio = OSTCBCur->OSTCBPrio;
    }
    ptcb = OSTCBPrioTbl[prio];
    if (ptcb == (OS_TCB *)0) {              /* Make sure task exist   */
        OS_EXIT_CRITICAL();
        return (OS_TASK_NOT_EXIST); }
```

OSTaskStkChk() – 2/3

```
if ((ptcb->OSTCBOpt & OS_TASK_OPT_STK_CHK) == 0) {
                                                /* Make sure stack checking option is set */
    OS_EXIT_CRITICAL();
    return (OS_TASK_OPT_ERR);
}
free = 0;
size = ptcb->OSTCBStkSize;
pchk = ptcb->OSTCBStkBottom;
OS_EXIT_CRITICAL();
#if OS_STK_GROWTH == 1
while (*pchk++ == (OS_STK)0) {
                                        /* Compute the number of zero entries on the stk */
    free++; }
#else
while (*pchk-- == (OS_STK)0) {
    free++; }
#endif
pdata->OSFree = free * sizeof(OS_STK);
                                        /* Compute number of free bytes on the stack */
pdata->OSUsed = (size - free) * sizeof(OS_STK);
                                        /* Compute number of bytes used on the stack */
return (OS_NO_ERR);
}
```

OSTaskStkChk() – 3/3



OSTaskDel() – 1/4

```
INT8U  OSTaskDel (INT8U prio){
#if OS_CRITICAL_METHOD == 3    /* Allocate storage for CPU status register */
    OS_CPU_SR     cpu_sr;
#endif
#if OS_EVENT_EN > 0
    OS_EVENT     *pevent;
#endif
#if (OS_VERSION >= 251) && (OS_FLAG_EN > 0) && (OS_MAX_FLAGS > 0)
    OS_FLAG_NODE *pnode;
#endif
    OS_TCB       *ptcb;
    BOOLEAN       self;

    if (OSIntNesting > 0) {            /* See if trying to delete from ISR  <not allowed>*/
        return (OS_TASK_DEL_ISR);
    }
#if OS_ARG_CHK_EN > 0
    if (prio == OS_IDLE_PRIO) {     /* Not allowed to delete idle task    */
        return (OS_TASK_DEL_IDLE);
    }
    if (prio >= OS_LOWEST_PRIO && prio != OS_PRIO_SELF) {
                                    /* Task priority valid ?  */
        return (OS_PRIO_INVALID); }
#endif
```

OSTaskDel() – 2/4

```
OS_ENTER_CRITICAL();
  if (prio == OS_PRIO_SELF) {          /* See if requesting to delete self    */
     prio = OSTCBCur->OSTCBPrio;       /* Set priority to delete to current   */
  }
  ptcb = OSTCBPrioTbl[prio];
  if (ptcb != (OS_TCB *)0) {                        /* Task to delete must exist */
     if ((OSRdyTbl[ptcb->OSTCBY] &= ~ptcb->OSTCBBitX) == 0x00) {
                                                    /* Make task not ready  */
        OSRdyGrp &= ~ptcb->OSTCBBitY;
     }
#if OS_EVENT_EN > 0
     pevent = ptcb->OSTCBEventPtr;
     if (pevent != (OS_EVENT *)0) {   /* If task is waiting on event   */
        if ((pevent->OSEventTbl[ptcb->OSTCBY] &= ~ptcb->OSTCBBitX) == 0)
{                                                   /*... remove task from */
           pevent->OSEventGrp &= ~ptcb->OSTCBBitY;  /* ... event ctrl block */
        }
     }
#endif
#if (OS_VERSION >= 251) && (OS_FLAG_EN > 0) && (OS_MAX_FLAGS > 0)
     pnode = ptcb->OSTCBFlagNode;
     if (pnode != (OS_FLAG_NODE *)0) {    /* If task is waiting on event flag */
        OS_FlagUnlink(pnode);             /* Remove from wait list  */
     }
#endif
```

OSTaskDel() – 3/4

```
ptcb->OSTCBDly  = 0;          /* Prevent OSTimeTick() from updating  */
ptcb->OSTCBStat = OS_STAT_RDY;   /* Prevent task from being resumed */
if (OSLockNesting < 255) {
   OSLockNesting++; }
OS_EXIT_CRITICAL();           /* Enabling INT. ignores next instruc. */
OS_Dummy();                   /* ... Dummy ensures that INTs will be */
OS_ENTER_CRITICAL();          /* ... disabled HERE! */
if (OSLockNesting > 0) {
   OSLockNesting--; }
OSTaskDelHook(ptcb);       /* Call user defined hook */
OSTaskCtr--;                  /* One less task being managed */
OSTCBPrioTbl[prio] = (OS_TCB *)0;          /* Clear old priority entry */
if (ptcb->OSTCBPrev == (OS_TCB *)0) {   /* Remove from TCB chain*/
   ptcb->OSTCBNext->OSTCBPrev = (OS_TCB *)0;
   OSTCBList = ptcb->OSTCBNext;
} else {
   ptcb->OSTCBPrev->OSTCBNext = ptcb->OSTCBNext;
   ptcb->OSTCBNext->OSTCBPrev = ptcb->OSTCBPrev;
}
ptcb->OSTCBNext = OSTCBFreeList;    /* Return TCB to free TCB list */
OSTCBFreeList   = ptcb;
```

OSTaskDel() – 4/4

```
    OS_EXIT_CRITICAL();
        OS_Sched();          /* Find new highest priority task*/
        return (OS_NO_ERR);
    }
    OS_EXIT_CRITICAL();
    return (OS_TASK_DEL_ERR);
}
```

OSTaskDelReq() - TaskB

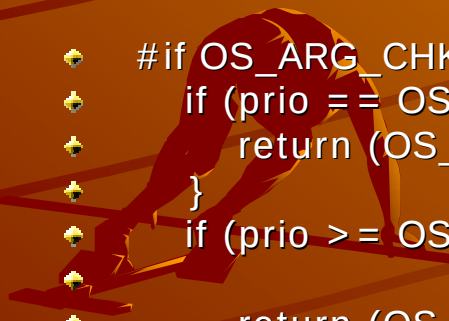
```
void RequestorTask (void *pdata)
{
    INT8U err;
    pdata = pdata;
for (;;) {
        /* Application code */
    if ('TaskToBeDeleted()' needs to be deleted) {
        while (OSTaskDelReq(TASK_TO_DEL_PRIO) != OS_TASK_NOT_EXIST)  {
            OSTimeDly(1);
        }
    }
        /* Application code */
}
}
```

OSTaskDelReq() - TaskA

```
void TaskToBeDeleted (void *pdata)
{
INT8U err;
pdata = pdata;
for (;;) {
    /* Application code */
    if (OSTaskDelReq(OS_PRIO_SELF) == OS_TASK_DEL_REQ) {
        Release any owned resources;
        De-allocate any dynamic memory;
        OSTaskDel(OS_PRIO_SELF);
    } else {
        /* Application code */
    }
}
}
```

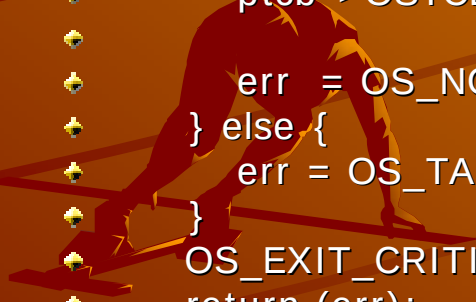
OSTaskDelReq() – 1/2

```
INT8U  OSTaskDelReq (INT8U prio)
{
#if OS_CRITICAL_METHOD == 3       /* Allocate storage for CPU status register*/
    OS_CPU_SR  cpu_sr;
#endif
    BOOLEAN    stat;
    INT8U      err;
    OS_TCB     *ptcb;


#if OS_ARG_CHK_EN > 0
    if (prio == OS_IDLE_PRIO) {          /* Not allowed to delete idle task */
        return (OS_TASK_DEL_IDLE);
    }
    if (prio >= OS_LOWEST_PRIO && prio != OS_PRIO_SELF) {
                                         /* Task priority valid ? */
        return (OS_PRIO_INVALID);
    }
#endif
```



OSTaskDelReq() – 2/2

```
if (prio == OS_PRIO_SELF) {                 /* See if a task is requesting to ..*/
    OS_ENTER_CRITICAL();                    /* ... this task to delete itself */
    stat = OSTCBCur->OSTCBDelReq;           /* Return request status to caller */
    OS_EXIT_CRITICAL();
    return (stat);
}
OS_ENTER_CRITICAL();
ptcb = OSTCBPrioTbl[prio];
if (ptcb != (OS_TCB *)0) {                  /* Task to delete must exist */
    ptcb->OSTCBDelReq = OS_TASK_DEL_REQ;
                                            /* Set flag indicating task to be DEL. */
    err  = OS_NO_ERR;
} else {
    err = OS_TASK_NOT_EXIST;                /* Task must be deleted  */
}
OS_EXIT_CRITICAL();
return (err);
}
```



OSTaskChangePrio() – 1/3

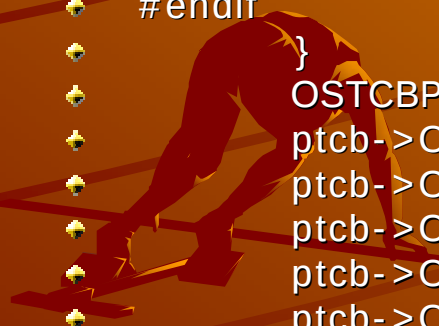
```
INT8U  OSTaskChangePrio (INT8U oldprio, INT8U newprio)
{
#if OS_CRITICAL_METHOD == 3     /* Allocate storage for CPU status register*/
    OS_CPU_SR    cpu_sr;
#endif

#if OS_EVENT_EN > 0
    OS_EVENT    *pevent;
#endif

    OS_TCB      *ptcb;
    INT8U        x;
    INT8U        y;
    INT8U        bitx;
    INT8U        bity;

#if OS_ARG_CHK_EN > 0
    if ((oldprio >= OS_LOWEST_PRIO && oldprio != OS_PRIO_SELF)  ||
        newprio >= OS_LOWEST_PRIO) {
        return (OS_PRIO_INVALID);
    }
#endif
    OS_ENTER_CRITICAL();
```

OSTaskChangePrio() – 2/3

```
if (OSTCBPrioTbl[newprio] != (OS_TCB *)0) {
                                                /*New priority must not already exist */
    OS_EXIT_CRITICAL();
    return (OS_PRIO_EXIST);
  } else {
    OSTCBPrioTbl[newprio] = (OS_TCB *)1;
                                                /* Reserve the entry to prevent others */
    OS_EXIT_CRITICAL();
    y    = newprio >> 3;                        /* Precompute to reduce INT. latency   */
    bity = OSMapTbl[y];                         /*Prepare for ready list*/
    x    = newprio & 0x07;
    bitx = OSMapTbl[x];
    OS_ENTER_CRITICAL();
    if (oldprio == OS_PRIO_SELF) {              /* See if changing self */
        oldprio = OSTCBCur->OSTCBPrio;          /* Yes, get priority */ }
    ptcb = OSTCBPrioTbl[oldprio];
    if (ptcb != (OS_TCB *)0) {                  /* Task to change must exist */
       OSTCBPrioTbl[oldprio] = (OS_TCB *)0;     /* Remove TCB from old priority */
       if ((OSRdyTbl[ptcb->OSTCBY] & ptcb->OSTCBBitX) != 0x00) {
                                                /* If task is ready make it not */
          if ((OSRdyTbl[ptcb->OSTCBY] &= ~ptcb->OSTCBBitX) == 0x00) {
             OSRdyGrp &= ~ptcb->OSTCBBitY;  }
          OSRdyGrp    |= bity;                  /* Make new priority ready to run  */
          OSRdyTbl[y] |= bitx;
```

OSTaskChangePrio() – 3/3

```
#if OS_EVENT_EN > 0
            } else {
               pevent = ptcb->OSTCBEventPtr;
               if (pevent != (OS_EVENT *)0) {          /* Remove from event wait list  */
                if ((pevent->OSEventTbl[ptcb->OSTCBY] &= ~ptcb->OSTCBBitX) == 0)
{
                   pevent->OSEventGrp &= ~ptcb->OSTCBBitY;  }
                  pevent->OSEventGrp  |= bity;      /* Add new priority to wait list */
                  pevent->OSEventTbl[y] |= bitx;  }
#endif
            }
            OSTCBPrioTbl[newprio] = ptcb;    /* Place pointer to TCB @ new priority */
            ptcb->OSTCBPrio      = newprio;  /* Set new task priority */
            ptcb->OSTCBY         = y;
            ptcb->OSTCBX         = x;
            ptcb->OSTCBBitY      = bity;
            ptcb->OSTCBBitX      = bitx;
            OS_EXIT_CRITICAL();
            OS_Sched();                          /* Run highest priority task ready */
            return (OS_NO_ERR);
         } else {
            OSTCBPrioTbl[newprio] = (OS_TCB *)0;  /* Release the reserved prio.  Task not exist.*/
            OS_EXIT_CRITICAL();
            return (OS_PRIO_ERR);     /* Task to change didn't exist  */
         }  }  }
```



OSTaskSuspend() – 1/2

```
INT8U  OSTaskSuspend (INT8U prio)
{
#if OS_CRITICAL_METHOD == 3    /* Allocate storage for CPU status register*/
    OS_CPU_SR  cpu_sr;
#endif
    BOOLEAN    self;
    OS_TCB    *ptcb;


#if OS_ARG_CHK_EN > 0
    if (prio == OS_IDLE_PRIO) {          /* Not allowed to suspend idle task    */
        return (OS_TASK_SUSPEND_IDLE);
    }
    if (prio >= OS_LOWEST_PRIO && prio != OS_PRIO_SELF) {
                                         /* Task priority valid ?               */
        return (OS_PRIO_INVALID);
    }
#endif
    OS_ENTER_CRITICAL();
    if (prio == OS_PRIO_SELF) {          /* See if suspend SELF                 */
        prio = OSTCBCur->OSTCBPrio;
        self = TRUE;
```

OSTaskSuspend() – 2/2

```
    } else if (prio == OSTCBCur->OSTCBPrio) {      /* See if suspending self */
        self = TRUE;
    } else {
        self = FALSE;                              /* No suspending another task */
    }
    ptcb = OSTCBPrioTbl[prio];
    if (ptcb == (OS_TCB *)0) {                     /* Task to suspend must exist */
        OS_EXIT_CRITICAL();
        return (OS_TASK_SUSPEND_PRIO);
    }
    if ((OSRdyTbl[ptcb->OSTCBY] &= ~ptcb->OSTCBBitX) == 0x00) {
                                                   /* Make task not ready */
        OSRdyGrp &= ~ptcb->OSTCBBitY;
    }
    ptcb->OSTCBStat |= OS_STAT_SUSPEND;  /* Status of task is 'SUSPENDED'*/
    OS_EXIT_CRITICAL();
    if (self == TRUE) {                            /* Context switch only if SELF  */
        OS_Sched();
    }
    return (OS_NO_ERR);
}
```

OSTaskResume() – 1/2

```
INT8U  OSTaskResume (INT8U prio)
{
#if OS_CRITICAL_METHOD == 3    /* Allocate storage for CPU status register*/
    OS_CPU_SR  cpu_sr;
#endif
    OS_TCB    *ptcb;


#if OS_ARG_CHK_EN > 0
    if (prio >= OS_LOWEST_PRIO) {  /* Make sure task priority is valid    */
        return (OS_PRIO_INVALID);
    }
#endif
    OS_ENTER_CRITICAL();
    ptcb = OSTCBPrioTbl[prio];
    if (ptcb == (OS_TCB *)0) {        /* Task to suspend must exist        */
        OS_EXIT_CRITICAL();
        return (OS_TASK_RESUME_PRIO);
    }
```

OSTaskResume() – 2/2

```
if ((ptcb->OSTCBStat & OS_STAT_SUSPEND) != OS_STAT_RDY) {
                                                        /* Task must be suspended   */
    if (((ptcb->OSTCBStat &= ~OS_STAT_SUSPEND) == OS_STAT_RDY) &&
                                                        /* Remove suspension        */
        (ptcb->OSTCBDly  == 0)) {                       /* Must not be delayed      */
        OSRdyGrp               |= ptcb->OSTCBBitY;      /* Make task ready to run   */
        OSRdyTbl[ptcb->OSTCBY] |= ptcb->OSTCBBitX;
        OS_EXIT_CRITICAL();
        OS_Sched();
    } else {
        OS_EXIT_CRITICAL();
    }
    return (OS_NO_ERR);
  }
  OS_EXIT_CRITICAL();
  return (OS_TASK_NOT_SUSPENDED);
}
```

OSTaskQuery()

```
INT8U  OSTaskQuery (INT8U prio, OS_TCB *pdata)
{
#if OS_CRITICAL_METHOD == 3                    /* Allocate storage for CPU status register */
    OS_CPU_SR  cpu_sr;
#endif
    OS_TCB    *ptcb;

#if OS_ARG_CHK_EN > 0
    if (prio > OS_LOWEST_PRIO && prio != OS_PRIO_SELF) {
                                               /* Task priority valid ?                    */
        return (OS_PRIO_INVALID);  }
#endif
    OS_ENTER_CRITICAL();
    if (prio == OS_PRIO_SELF) {                /* See if suspend SELF                      */
        prio = OSTCBCur->OSTCBPrio;  }
    ptcb = OSTCBPrioTbl[prio];
    if (ptcb == (OS_TCB *)0) {                 /* Task to query must exist                 */
        OS_EXIT_CRITICAL();
        return (OS_PRIO_ERR);  }
    memcpy(pdata, ptcb, sizeof(OS_TCB));       /* Copy TCB into user storage area          */
    OS_EXIT_CRITICAL();
    return (OS_NO_ERR);
}
```

Thank you !!