3 Task Management

Contents

- Creating a Task: `OSTaskCreate()`
- Creating a Task: `OSTaskCreateExt()`
- Task Stacks
- Stack Checking: `OSTaskStkChk()`
- Deleting a Task: `OSTaskDel()`
- Suspend a Task: `OSTaskSuspend()`
- Resuming a Task: `OSTaskResume()`
- Getting Information about a Task: `OSTaskQuery()`

OSTaskCreate() - 1/2

```
INT8U OSTaskCreate(void (*task)(void *pd), void *pdata, OS_STK
   *ptos, INT8U prio)
{
  OS_STK *psp;
  INT8U err;

#if OS_ARG_CHK_EN > 0
  if (prio > OS_LOWEST_PRIO) {
    return (OS_PRIO_INVALID);
  }
#endif
  OS_ENTER_CRITICAL();
  if (OSTCBPrioTbl[prio] == (OS_TCB *)0)
    OSTCBPrioTbl[prio] = (OS_TCB *)1;
    OS_EXIT_CRITICAL();
    psp = (OS_STK *)OSTaskStkInit(task, pdata, ptos, 0);
    err = OS_TCBInit(prio, psp, (OS_STK *)0, 0, 0, (void *)0, 0);
```

check if the task available?

reserve the task first

call these two functions

OSTaskCreate() - 2/2

```
    if (err == OS_NO_ERR) {
      OS_ENTER_CRITICAL();
      OSTaskCtr++;
      OS_EXIT_CRITICAL();
      if (OSRunning == TRUE) {
        OS_Sched();
      }
    } else {
      OS_ENTER_CRITICAL();
      OSTCBPrioTbl[prio] = (OS_TCB *)0;
      OS_EXIT_CRITICAL();
    }
    return (err);
  }
  OS_EXIT_CRITICAL();
  return (OS_PRIO_EXIST);
}
```

increment the task counter

find the highest task if multitasking

release the task

OSTaskCreateExt() - 1/2

```
INT8U OSTaskCreateExt(..., INT16U id, OS_STK *pbos, INT32U stk_size, void
   *pext, INT16U opt)
{
  OS_STK *psp; INT8U err;

  OS_ENTER_CRITICAL();
  if(OSTCBPrioTbl[prio] == (OS_TCB *)0) {
    OSTCBPrioTbl[prio] = (OS_TCB *)1;
    OS_EXIT_CRITICAL();
    if(((opt & OS_TASK_OPT_STK_CHK) != 0x0000) ||
        ((opt & OS_TASK_OPT_STK_CLR) != 0x0000)) {
      #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);
```

if stack checking is enabled?

if stack clearing is enabled?

stack is initialized as 0

OSTaskCreateExt() - 2/2

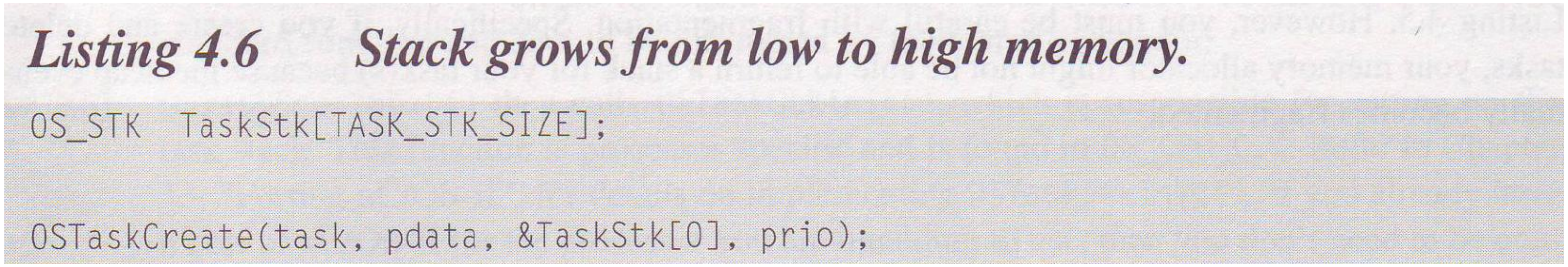
```
    err = OS_TCBInit(prio, psp, pbos, id, stk_size, pext, opt);
    if (err == OS_NO_ERR) {
      OS_ENTER_CRITICAL();
      OSTaskCtr++;
      OS_EXIT_CRITICAL();
      if (OSRunning == TRUE) {
        OS_Sched();
      }
    } else {
      OS_ENTER_CRITICAL();
      OSTCBPrioTbl[prio] = (OS_TCB *)0;
      OS_EXIT_CRITICAL();
    }
    return (err);
  }
  OS_EXIT_CRITICAL();
  return (OS_PRIO_EXIST);
}
```

opt is used here

call scheduler if kernel is in multitasking

Task Stacks

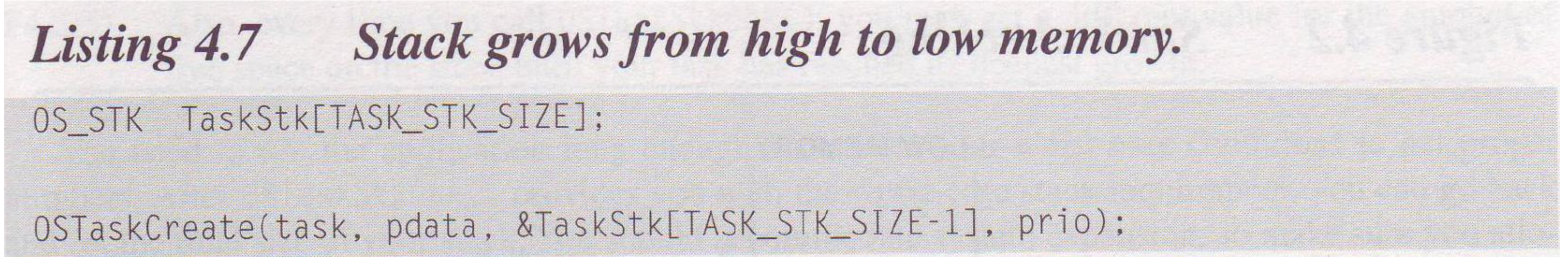
- Stack element must be declared as of type `OS_STK`
- Stack must be contiguous



Listing 4.6 ***Stack grows from low to high memory.***

```
OS_STK  TaskStk[TASK_STK_SIZE];

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



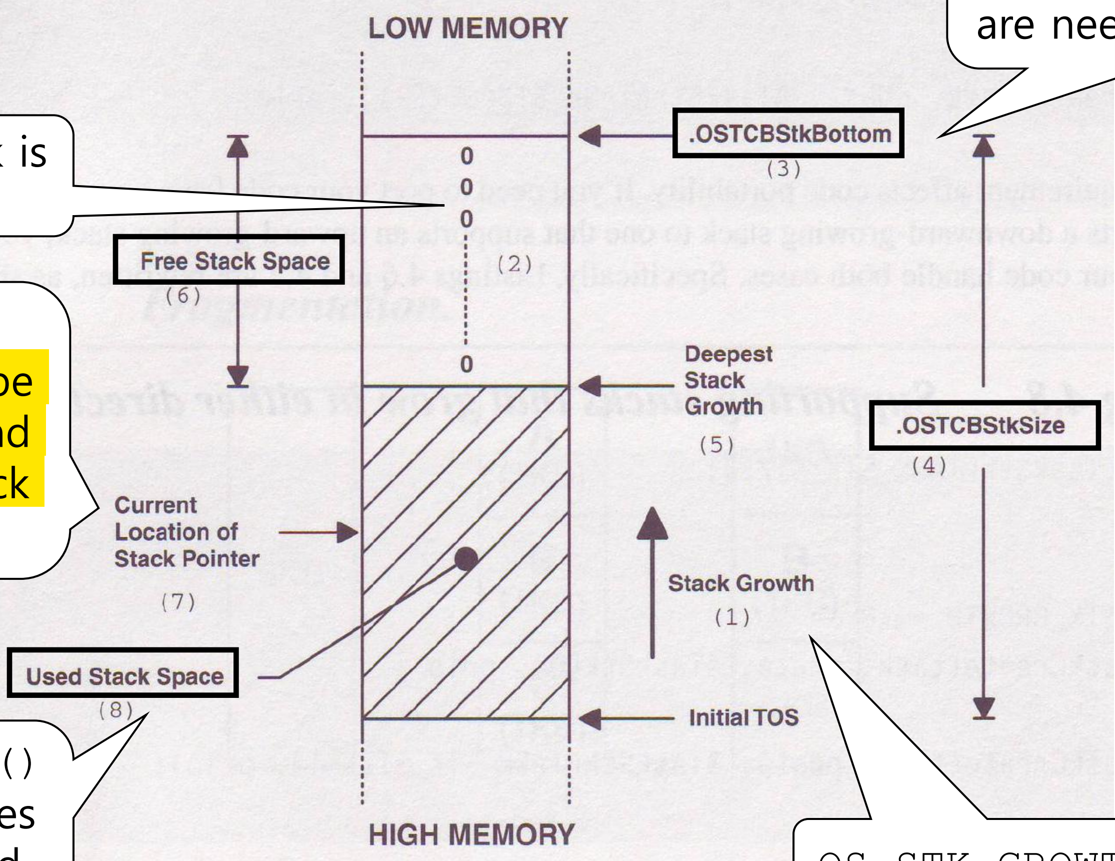
Listing 4.7 ***Stack grows from high to low memory.***

```
OS_STK  TaskStk[TASK_STK_SIZE];

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

Stack Checking

Figure 4.2 ***Stack checking.***



`OSTCBStkBottom` and `OSTCBStkSize` are needed

initially the stack is filled with 0

current stack pointer should be between TOS and the deepest stack growth

`OSTaskStkChk()` return no of bytes for free and used spaces

`OS_STK_GROWTH` is 1

OSTaskStkChk() - 1/2

```
INT8U OSTaskStkChk(INT8U prio, OS_STK_DATA * pdata)
{
  OS_TCB *ptcb, OS_STK *pchk, INT32U free, INT32U size;

  pdata->OSFree = 0;
  pdata->OSUsed = 0;
  OS_ENTER_CRITICAL();
  if (prio == OS_PRIO_SELF) {
    prio = OSTCBCur->OSTCBPrio;
  }

  ptcb = OSTCBPrioTbl[prio];
  if (ptcb == (OS_TCB *) 0) {
    OS_EXIT_CRITICAL();
    return (OS_TASK_NOT_EXIST);
  }
```

find the priority if `prio==OS_PRIO_SELF`

check if the task is an existing task

OSTaskStkChk() - 2/2

```
  if((ptcb->OSTCBOpt & OS_TASK_OPT_STK_CHK) == 0) {
    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) free++;
#else
  while (*pchk-- == (OS_STK) 0) free++;
#endif
  pdata->OSFree = free * sizeof(OS_STK);
  pdata->OSUsed = (size - free) * sizeof(OS_STK);
  return (OS_NO_ERR);
}
```

check if the task was created with option `OS_TASK_OPT_STK_CHK`

get stack size and bottom of the stack

calculate free area size

calculate the free and used size in byte

OSTaskDel() - 1/3

```
INT8U OSTaskDel(INT8U prio)
{
  OS_EVENT *pevent; OS_FLAG_NODE *pnode; OS_TCB *ptcb; BOOLEAN self;

  if (OSIntNesting > 0) return (OS_TASK_DEL_ISR);
  OS_ENTER_CRITICAL();
  if (prio == OS_PRIO_SELF) prio = OSTCBCur->OSTCBPrio;
  ptcb = OSTCBPrioTbl[prio];
  if (ptcb != (OS_TCB *) 0) {
    if ((OSRdyTbl[ptcb->OSTCBY] &= ~ptcb->OSTCBBitX) == 0x00)
      OSRdyGrp &= ~ptcb->OSTCBBitY;
    pevent = ptcb->OSTCBEventPtr;
    if (pevent != (OS_EVENT *) 0)
      if ((pevent->OSEventTbl[ptcb->OSTCBY] &= ~ptcb->OSTCBBitX) == 0)
        pevent->OSEventGrp &= ~ptcb->OSTCBBitY;
```

ISR can not call it

check if existing task

if in ready list, remove it from the list

if in event waiting list, remove it from the list

OSTaskDel() - 2/3

```
pnode = ptcb->OSTCBFlagNode;
if (pnode != (OS_FLAG_NODE *) 0) OS_FlagUnlink(pnode);
ptcb->OSTCBDly = 0;
ptcb->OSTCBStat = OS_STAT_RDY;
if (OSLockNesting < 255) OSLockNesting++;
OS_EXIT_CRITICAL();
OS_Dummy();
OS_ENTER_CRITICAL();
if (OSLockNesting > 0) OSLockNesting--;
OSTaskDelHook(ptcb);
OSTaskCtr--;
OSTCBPrioTbl[prio] = (OS_TCB *) 0;
```

if in event flag list, remove it from the list

delay=0, state=ready(?) and lock scheduling

decrement task counter

OSTaskDel() - 3/3

```
    if (ptcb->OSTCBPrev == (OS_TCB *) 0) {
      ptcb->OSTCBNext->OSTCBPrev = (OS_TCB *) 0;
      OSTCBList = ptcb->OSTCBNext;
    } else {
      ptcb->OSTCBPrev->OSTCBNext = ptcb->OSTCBNext;
      ptcb->OSTCBNext->OSTCBPrev = ptcb->OSTCBPrev;
    }
    ptcb->OSTCBNext = OSTCBFreeList;
    OSTCBFreeList = ptcb;
    OS_EXIT_CRITICAL();
    OS_Sched();
    return (OS_NO_ERR);
  }
  OS_EXIT_CRITICAL();
  return (OS_TASK_DEL_ERR);
}
```

return TCB
into free TCB

OSTaskSuspend()

```
INT8U OSTaskSuspend(INT8U prio)
{
  ...

  OS_ENTER_CRITICAL();

  ...
  ptcb = OSTCBPrioTbl[prio];
  ...
  if ((OSRdyTbl[ptcb->OSTCBY] &= ~ptcb->OSTCBBitX) == 0x00)
    OSRdyGrp &= ~ptcb->OSTCBBitY;

  ptcb->OSTCBStat |= OS_STAT_SUSPEND;
  OS_EXIT_CRITICAL();
  if (self == TRUE) OS_Sched();
  return (OS_NO_ERR);
}
```

if in ready list, remove it from the list

state = OS_STAT_SUSPEND

if the current task is suspended, call scheduler

OSTaskResume()

```
INT8U OSTaskResume(INT8U prio)
{
   ...
   OS_ENTER_CRITICAL();
   ptcb = OSTCBPrioTbl[prio];
   if ((ptcb->OSTCBStat & OS_STAT_SUSPEND) != 0x00) {
     if(((ptcb->OSTCBStat &= ~OS_STAT_SUSPEND)==OS_STAT_RDY) &&
       (ptcb->OSTCBDly==0)){
       OSRdyGrp               |= ptcb->OSTCBBitY;
       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);
}
```

check if in suspended state

Delete suspension bit and check if it is in ready and delay is zero

register in `OSRdyGrp` and `OSRdyTbl`

OSTaskQuery()

```
INT8U OSTaskQuery(INT8U prio, OS_TCB *pdata)
{
  OS_TCB    *ptcb;

  OS_ENTER_CRITICAL();
  if (prio == OS_PRIO_SELF) {
    prio = OSTCBCur->OSTCBPrio;
  }
  ptcb = OSTCBPrioTbl[prio];
  if (ptcb == (OS_TCB *)0) {
    OS_EXIT_CRITICAL();
    return (OS_PRIO_ERR);
  }
  memcpy(pdata, ptcb, sizeof(OS_TCB));
  OS_EXIT_CRITICAL();
  return (OS_NO_ERR);
}
```

copy TCB to pdata