Memory Management – uC/OSII

Yu-Han Li
OSNET Lab. of C.S.
National Chung Hsing University

Outline

- Introduction
- Memory Control Block
- Operation functions
  - OSMemCreate()
  - OSMemGet()
  - OSMemPut()
  - OSMemQuery()
- Partitions usage example
- Waiting for Memory Blocks
- Conclusion

Introduction – (1/2)

- Problem

  Malloc() and Free() in embedded systems are dangerous.
  - Fragmentation is the development of a large number of separate free areas.
  - Execution times are also nondeterministic.
- Solution

  uC/OS provides fixed-sized memory blocks from a partition
  - A partition made of a contiguous memory area.
  - Allocation and deallocation are done in constant time and deterministic.
  - We can obtain memory blocks of different sizes.

Introduction – (2/2)

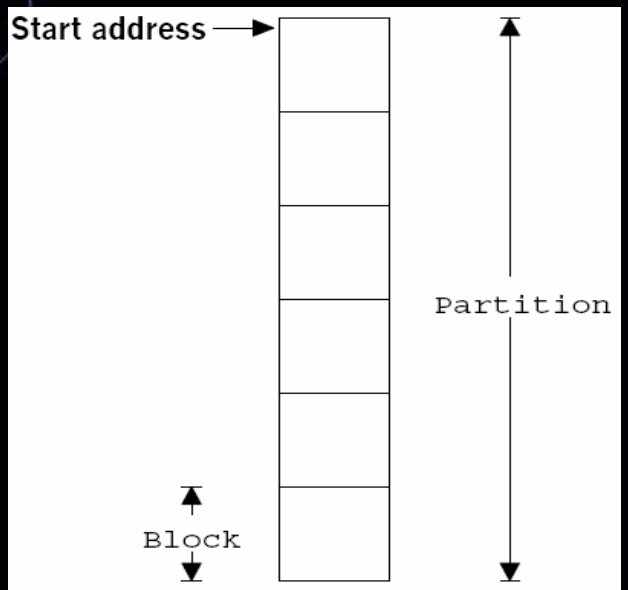


Figure 7-1, Memory partition

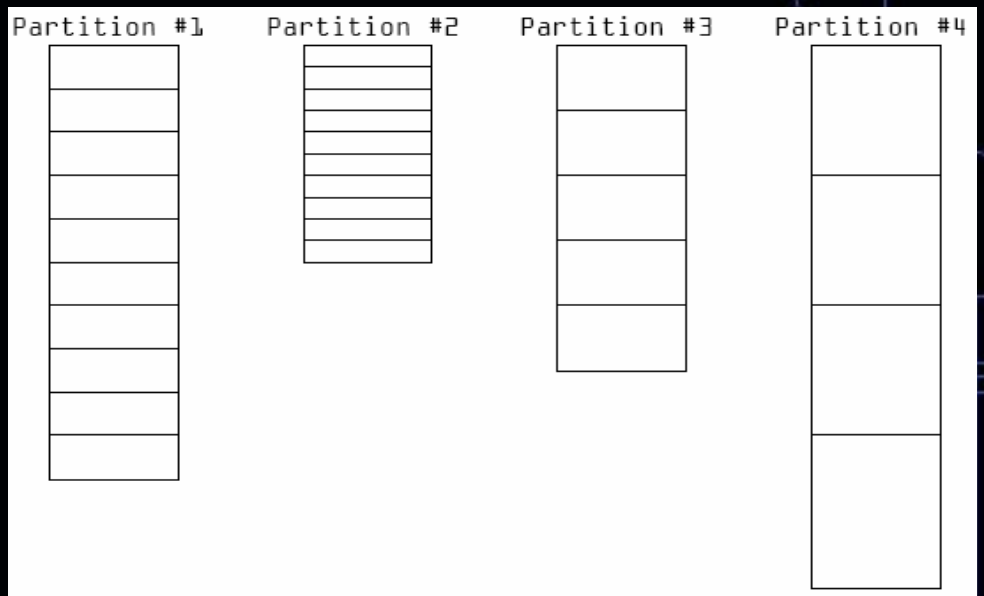
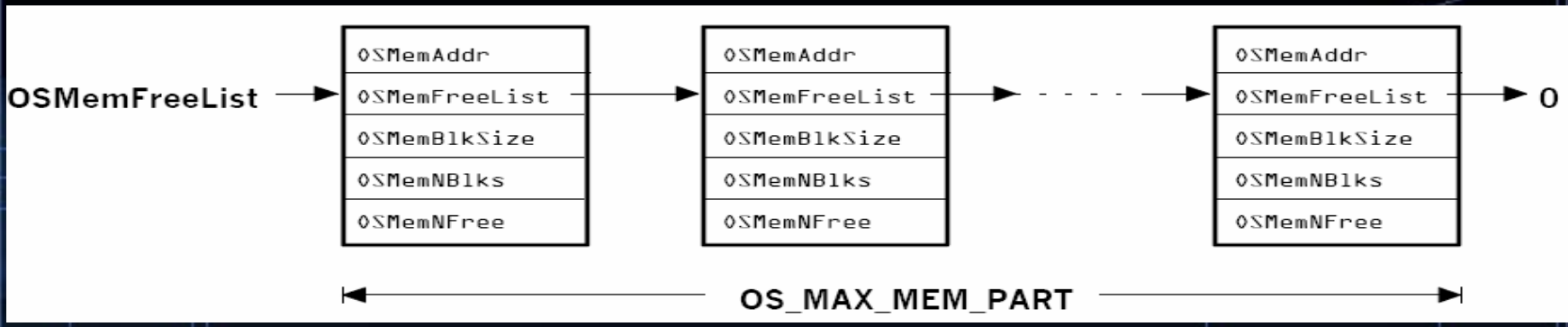


Figure 7-2, Multiple memory partitions.

- To enable management services, set OS_MEM_EN to 1 in OS_CFG.H.
- To use OSMemQuery(), set OS_MEM_QUERY_EN to 1 in OS_CFG.H additionally.

Memory Control Block – (1/2)

- uC/OS-II keeps track of partitions through memory control block(MCB).
- Each partition requires its own MCB.
- Initialization is done by OS_MemInit(), and creates a linked list of free MCBs.
- Specify the maximum number of partitions with OS_MAX_MEM_PART in OS_CFG.H.



Memory Control Block – (2/2)

- Data structure

```
typedef struct{
   void       *OSMemAddr;       ->  A
   void       *OSMemFreeList;   ->  B
   INT32U  OSMemBlkSize;     ->  C
   INT32U  OSMemNBlks;       ->  D
   INT32U  OSMemNFree;       ->  E
}OS_MEM;
```

- A is a ptr to the beginning of memory partition (not changed).
- B is a ptr to either the next free MCB or the next free memory block.
- C determines the size of each memory block in the partition.
- D determines the total number of memory blocks available.
- E determines how many memory blocks available.

Operation functions – OSMemCreate() – (1/4)

- Main function

```
OS_MEM *Comm TxBuf;
INT8U CommTxPart[100][32];
void main(void){
    INT8U err;
    OSInit();
        . . .
    CommTxBuf = OSMemCreate(CommTxPart, 100, 32, &err);
        . . .
    OSStart(); }
```

- OSMemCreate()

```
OS_MEM  *OSMemCreate (void *addr, INT32U nblks, INT32U blksize, INT8U *err)  {
#if OS_CRITICAL_METHOD == 3
    OS_CPU_SR  cpu_sr;
#endif
    OS_MEM    *pmem;
    INT8U     *pblk;
    void     **plink;
    INT32U     i;
```

Operation functions – OSMemCreate() – (2/4)

```
#if OS_ARG_CHK_EN > 0
    if (addr == (void *)0) {          /* Must pass a valid address for the memory part.*/
        *err = OS_MEM_INVALID_ADDR;
        return ((OS_MEM *)0);  }
    if (nblks < 2) {                  /* Must have at least 2 blocks per partition */
        *err = OS_MEM_INVALID_BLKS;
        return ((OS_MEM *)0);  }
    if (blksize < sizeof(void *)) {   /* Must contain space for at least a pointer*/
        *err = OS_MEM_INVALID_SIZE;
        return ((OS_MEM *)0);  }
#endif
    OS_ENTER_CRITICAL();
    pmem = OSMemFreeList;             /* Get next free memory partition  */
    if (OSMemFreeList != (OS_MEM *)0) {
                                      /*See if pool of free partitions was empty*/
        OSMemFreeList = (OS_MEM *)OSMemFreeList->OSMemFreeList;
    }
    OS_EXIT_CRITICAL();
```

Operation functions – OSMemCreate() – (3/4)

```
if (pmem == (OS_MEM *)0) {                /* See if we have a memory partition*/
    *err = OS_MEM_INVALID_PART;
    return ((OS_MEM *)0);
  }
  plink = (void **)addr;                  /* Create linked list of free memory blocks*/
  pblk  = (INT8U *)addr + blksize;
  for (i = 0; i < (nblks - 1); i++) {
    *plink = (void *)pblk;
    plink  = (void **)pblk;
    pblk   = pblk + blksize;  }
  *plink              = (void *)0;        /* Last memory block points to NULL */
  pmem->OSMemAddr     = addr;             /* Store start address of memory partition*/
  pmem->OSMemFreeList = addr;             /* Initialize pointer to pool of free blocks*/
  pmem->OSMemNFree    = nblks;            /* Store number of free blocks in MCB*/
  pmem->OSMemNBlks    = nblks;
  pmem->OSMemBlkSize  = blksize;  /* Store block size of each memory blocks*/
  *err                = OS_NO_ERR;
  return (pmem);
}
```

Operation functions –

OSMemCreate() – (4/4)

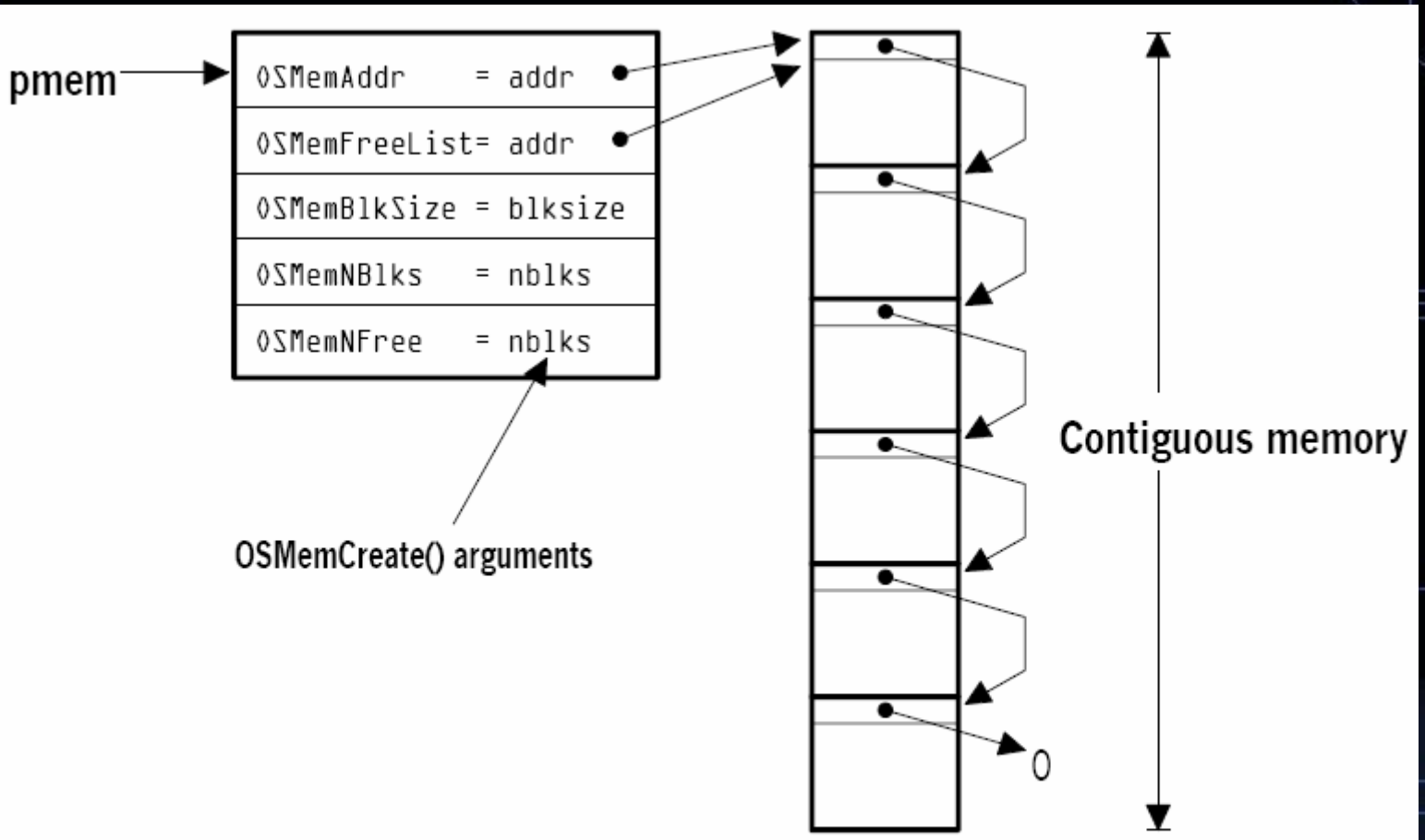


Figure 7-4, OSMemCreate()

Operation functions – OSMemGet()

```
void  *OSMemGet (OS_MEM *pmem, INT8U *err) {
#if OS_CRITICAL_METHOD == 3
    OS_CPU_SR  cpu_sr;
#endif
    void      *pblk;
#if OS_ARG_CHK_EN > 0
    if (pmem == (OS_MEM *)0) {                  /* Must point to a valid memory partition */
        *err = OS_MEM_INVALID_PMEM;
        return ((OS_MEM *)0);  }
#endif
    OS_ENTER_CRITICAL();
    if (pmem->OSMemNFree > 0) {                 /* See if there are any free memory blocks */
        pblk              = pmem->OSMemFreeList;    /* Yes, point to next free memory block  */
        pmem->OSMemFreeList = *(void **)pblk;         /*  Adjust pointer to new free list   */
        pmem->OSMemNFree--;                         /*  One less memory block in this partition  */
        OS_EXIT_CRITICAL();
        *err = OS_NO_ERR;                          /* No error */
        return (pblk);                             /*  Return memory block to caller */
    }
    OS_EXIT_CRITICAL();
    *err = OS_MEM_NO_FREE_BLKS;      /* No,  Notify caller of empty memory partition  */
    return ((void *)0);                           /* Return NULL pointer to caller */
}
```

Operation functions – OSMemPut()

```
INT8U  OSMemPut (OS_MEM  *pmem, void *pblk)
{
#if OS_CRITICAL_METHOD == 3
    OS_CPU_SR  cpu_sr;
#endif
#if OS_ARG_CHK_EN > 0
    if (pmem == (OS_MEM *)0) {                          /* Must point to a valid memory partition */
        return (OS_MEM_INVALID_PMEM);  }
    if (pblk == (void *)0) {                            /* Must release a valid block */
        return (OS_MEM_INVALID_PBLK);  }
#endif
    OS_ENTER_CRITICAL();
    if (pmem->OSMemNFree >= pmem->OSMemNBlks) {
                                                        /* Make sure all blocks not already returned */
        OS_EXIT_CRITICAL();
        return (OS_MEM_FULL);  }
    *(void **)pblk      = pmem->OSMemFreeList;   /* Insert released block into free block list */
    pmem->OSMemFreeList = pblk;
    pmem->OSMemNFree++;                         /* One more memory block in this partition */
    OS_EXIT_CRITICAL();
    return (OS_NO_ERR);                         /* Notify caller that memory block was released */
}
```

Operation functions – OSMemQuery()

- Data structure

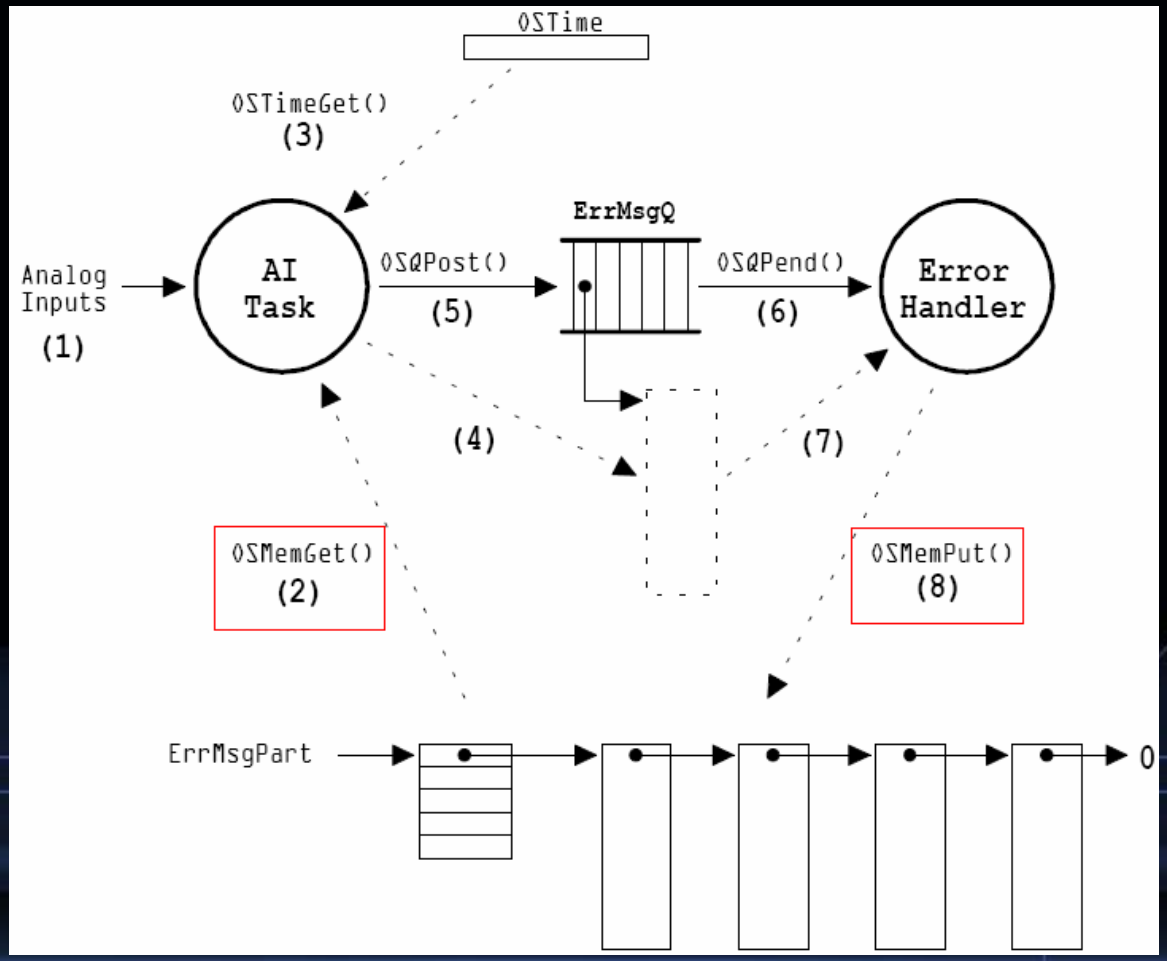
```
typedef struct {
    void   *OSAddr;
    void   *OSFreeList;
    INT32U  OSBlkSize;
    INT32U  OSNBlks;
    INT32U  OSNFree;
    INT32U  OSNUsed;
} OS_MEM_DATA;
```

- OSMemQuery() – the most important parts

```
    . . .
    OS_ENTER_CRITICAL();
    pdata->OSAddr     = pmem->OSMemAddr;
    pdata->OSFreeList = pmem->OSMemFreeList;
    pdata->OSBlkSize  = pmem->OSMemBlkSize;
    pdata->OSNBlks    = pmem->OSMemNBlks;
    pdata->OSNFree    = pmem->OSMemNFree;
    OS_EXIT_CRITICAL();
    pdata->OSNUsed    = pdata->OSNBlks - pdata->OSNFree;
    . . .
```

Partitions usage example

- Using dynamic memory allocation (partitions)



Waiting for Memory Blocks

```
OS_EVENT  *SemaphorePtr;                                          (1)
OS_MEM    *PartitionPtr;
INT8U      Partition[100][32];
OS_STK     TaskStk[1000];


void main (void)
{
    INT8U err;


    OSInit();                                                     (2)
    .
    .
    SemaphorePtr = OSSemCreate(100);                              (3)
    PartitionPtr = OSMemCreate(Partition, 100, 32, &err);         (4)
    .
    OSTaskCreate(Task, (void *)0, &TaskStk[999], &err);           (5)
    .
    OSStart();                                                    (6)
}


void Task (void *pdata)
{
    INT8U  err;
    INT8U *pblock;


    for (;;) {
        OSSemPend(SemaphorePtr, 0, &err);                         (7)
        pblock = OSMemGet(PartitionPtr, &err);                    (8)
        .
        .  /* Use the memory block */
        .
        OSMemPut(PartitionPtr, pblock);                           (9)
        OSSemPost(SemaphorePtr);                                 (10)
    }
}
```

Conclusion

- It doesn’t make sense to use partition unless we share memory blocks with other tasks.