CooCox CoOS User's Guide

Revision 1.0

August, 2009

CONTENTS

1 OverView

1.1 About CooCox CoOS

CooCox CoOS is an embedded real-time multi-task OS specially for ARM Cortex M series.

1.1.1 CoOS Features

- Specially designed for Cortex-M series.
- Free and open real-time Operating System.
- Scalable, minimal system kernel is only 974Byte.
- Adaptive Task Scheduling Algorithm.
- Supports preemptive priority and round-robin.
- Interrupt latency is 0.
- Semaphore, Mutex, Flag, Mailbox and Queue for communication & synchronisation
- Stack overflow detection option.
- Supports the platforms of ICCARM, ARMCC, GCC

1.1.2 Technical Data

Table 1.1.1 Time Specifications

N Function	Time(No Robin/Robin)
Create defined task, no task switch	5.3us/5.8us
Create defined task, switch task	7.5us/8.6us
Delete task (ExitTask)	4.8us/5.2us
Task switch (SwitchContext)	1.5us/1.5us
Task switch (upon set flag)	7.5us/8.1us
Task switch (upon sent semaphore)	6.3us/7.0us
Task switch (upon sent mail)	6.1us/7.1us
Task switch (upon sent queue)	7.0us/7.6us
Set Flag (no task switch)	1.3us/1.3 us
Send semaphore (no task switch)	1.6us/1.6us
Send mail (no task switch)	1.5us/1.5us
Send queue (no task switch)	1.8us/1.8us
Maximum interrupt lockout for IRQ ISR's	0/0

Table 1.1.2 Space Specifications

Description	Space
RAM Space for Kernel	168 Bytes
Code Space for Kernel	974 Bytes
RAM Space for a Task	TaskStackSize + 24 Bytes(MIN) TaskStackSize + 48 Bytes(MAX)
RAM Space for a Mailbox	16 Bytes
RAM Space for a Semaphore	16 Bytes
RAM Space for a Queue	32 Bytes
RAM Space for a Mutex	8 Bytes
RAM Space for a User Timer	24 Bytes

1.1.3 Supported Devices

- ST STM32 Series

- Atmel ATSAM3U Series

- NXP LPC17xx LPC13xx LPC11xx Series

- Toshiba TMPM330 Series

- Luminary LM3S Series

1.1.4 Source Codes Download

If you want to learn more about CooCox CoOS, you can download the CooCox CoOS source code from the web: http://www.coocox.org/.

1.2 CooCox CoOS getting started

This section describes the useage of CooCox, here we use Keil RealView MDK and EM-LPC1700 evaluation board to develop a simple demo based on CooCox CoOS.

Here we assume that you are able to use Keil RealView MDK to do simple development and basic set. The following will introduce a simple example which includes three tasks:

led   : Used for 8 LEDs' cyclic flickering, and set flags to activate the other two tasks every fixed time interval;

taskA : Wait for task flag 'a_flag', then print taskA through UART1;

taskB : Wait for task flag 'b_flag', then print taskB through UART1;

The overall phenomenon is that the 8 LEDs on board changes every 0.5s, the change order is :LED0→LED1→LED2→…→LED6→LED7→LED0→LED1→…, the serial port prints messages every 0.5s, printing like this:

taskA is running

taskB is running

…

The corresponding relationships between LEDs and GPIOs are as follows:

LED0   ←→   P1.28

LED1   ←→   P1.29

LED2   ←→   P1.31

LED[3…7] ←→   P[2.2…2.6]

Next we will introduce how to achieve the above functions in the environment of   CoOS.

1.2.1 Preparations

1> Visit www.coocox.org website to download 'The first CoOS Program' source code;

2> First of all, create a folder named 'getting_started'(Note: the name of the folder can't has spaces or Chinese);

3> And then enter the 'getting_started' folder, create a inc ,a src ,a ccrtos folder

respectively which used to storing header files and source files;

4> Copy files to the project directory:

(1)Copy *LPC17xx.h* and *system_LPC17xx.h* in the directory of Demo\Sample\CMSIS in 'The first CoOS Program' package to the 'inc' folder, copy *system_LPC17xx.c* and *startup_LPC17xx.s*(the startup code of LPC17xx) to the 'src' folder;

(2)Copy *config.h* in Source folder to the 'inc' folder, copy *main.c* in the directory of Demo\Sample to the 'src' folder;

(3)Copy *LED.c*, *Retarget.c*, *Serial.c* in the directory of Demo\Sample\driver to the 'src' folder, copy *serial.h*, *led.h* to the 'inc' folder;

(4)Copy all the files in the directory of Source\kernel and Source\portable\Keil (except .h files)to the 'ccrtos' folder, copy all the .h files to the 'inc' folder.

1.2.2 Create Project

1> Create an empty project used MDK software, device selects LPC1766 of NXP (do not choose the default startup code);

2> Add application-driven code to the project:

Add all the source files in 'src' folder to the project;

3> Add CoOS source code to the project:

Add all the source files in the 'ccrtos' folder to the project(Header files do not contain);

4> Project Configuration

Make the appropriate configuration to the project, add include path .\inc in C/C++. After that you should have been able to compile successfully, if not, please check whether the steps and the settings are correct.

(If you would like to use first rather than do the work above to save your time, you can use our ready-made project that we have prepared for you, the storage path is \Demo\getting_start_sample)

1.2.3 Write application code

Open '*main.c*', you could find that we have done a part of works for you, including clock initialization, serial port 1 initialization, GPIOs used for the flickering of LEDs initialization. Next you need to do is adding task code and configuring CoOS step by step.

1> Include CoOS header files

To use CoOS, first of all is to add the source code to your project, this step has been completed in front, next is to include CoOS's header files in your user code, that is, adding the following statements in *main.c*:

```
#include <config.h>          /*!< CoOS configure header file*/
#include <ccrtos.h>          /*!< CoOS header file          */
```

Note:It would be preferable to put '#include <config.h>' in front of '#include <ccrtos.h>' .

2> Write task code

You need to specify the stack space for the task when it is created, for CoOS, the stack pointer of the task is designated by user, so we need define three arrays used for the stack of the three tasks:

```
OS_STK      taskA_stk[128];      /*!< define "taskA" task stack  */
OS_STK      taskB_stk[128];      /*!< define "taskB" task stack  */
OS_STK      led_stk  [128];      /*!< define "led" task stack    */
```

taskA, taskB are waiting for their own flags respectively. Obviously, we need to create two flags used for task communication. In addition, taskA and taskB need serial port to print, serial port is a exclusive device which can be occupied only by one task, so we create a mutex used to guarantee the mutual exclusion when serial port printing.

```
OS_MutexID uart_mutex;          /*!< UART1 mutex id      */
OS_FlagID   a_flag,b_flag;       /*!< Save falg id        */
volatile    unsigned int Cnt = 0;    /*!< A counter          */
```

taskA: taskA waits for a flag, when the flag is set, print 'taskA is running', and then continue to wait for the flag, and the cycle repeats. Here we designed taskA as the highest priority task, and it will be implemented at first after starting the system, so we create the flags and the mutex which will be used later in taskA,

task code is as followings:

```
void taskA (void* pdata)
{
  /*!< Create a mutex for uart print */
  uart_mutex = CoCreateMutex ();
  if (uart_mutex == E_CREATE_FAIL)
  {  /*!< If failed to create, print message    */
      printf (" Failed to create Mutex! \n\r");
  }
  /*!< Create two flags to communicate between taskA and taskB    */
  a_flag = CoCreateFlag (TRUE,0);
  if (a_flag == E_CREATE_FAIL)
  {
      printf (" Failed to create the Flag! \n\r");
  }
  b_flag = CoCreateFlag (TRUE,0);
  if (b_flag == E_CREATE_FAIL)
  {
      printf (" Failed to create the Flag ! \n\r");
  }
  for (;;)
  {
      CoWaitForSingleFlag (a_flag,0);
      CoEnterMutexSection (uart_mutex);
      printf (" taskA is running \n\r");
      CoLeaveMutexSection (uart_mutex);
  }
}
```

taskB: taskB waits for b_flag to be set, print 'taskB is running' after activating,, and then continue to wait for the flag, and the cycle repeats. task code is as followings:

```
void taskB (void* pdata)
{
  for (;;)
  {
    CoWaitForSingleFlag (b_flag,0);

    CoEnterMutexSection (uart_mutex);
    printf (" taskB is running \n\r");
    CoLeaveMutexSection (uart_mutex);
  }
}
```

led: led task controls the changes of led, here we call CoTickDelay() to delay 0.5s to keep led lighting for 0.5s; at the same time led task will set a_flag and b_flag at the right time to activate taskA,taskB. Code is as followings:

```
void led (void* pdata)
{
  unsigned int led_num;
  for (;;)
  {
    led_num = 1<< (Cnt%8);
    LED_on (led_num);              /*!< Switch on led   */
    CoTickDelay (50);
    LED_off (led_num);             /*!< Switch off led   */
    if ((Cnt%2) == 0)
    {
        CoSetFlag (a_flag);       /*!< Set "a_flag" flag*/
    }
    else if ((Cnt%2) == 1)
    {
        CoSetFlag (b_flag);       /*!< Set "b_flag" flag*/
    }
    Cnt++;
  }
}
```

1.2.4 Create task and start CoOS

Right now we have completed all the task code, next should be initializing OS, creating tasks, start multi-task scheduling. You should initialize CoOS

before using CoOS or calling any CoOS API, that can be done by CoInitOS() function. After initialization, you could call the API functions of CoOS to create tasks,flags, mutexes,semaphores and so on. At last, system will start the first scheduling through CoStartOS() function. The code after CoStartOS() will not be implemented, since OS will not return after the first scheduling.

Add the following code after the initialization in main function:

```
CoInitOS ();                      /*!< Initial CooCox CoOS        */
  /*!< Create three tasks   */
  CoCreateTask (taskA,0,0,&taskA_stk[128-1],128);
  CoCreateTask (taskB,0,1,&taskB_stk[128-1],128);
  CoCreateTask (led   ,0,2,&led_stk[128-1]   ,128);
  CoStartOS();                    /*!< Start multitask           */
```

1.2.5 Configure and clip CoOS

Open Os*Config.h*, here contains all the items which can be configured and clipped. Be sure you have known all the functions of every item before you modify it, there are detailed notes to explain the function of every item in the document.

First of all, we must configure a few of the items which must be checked or modified:

CFG_MAX_USER_TASKS

It implies the maximum tasks that users can create, we have only 3 tasks, so we modify it as 3 to save space.

CFG_CPU_FREQ

It is the system clock that your system used, SystemInit() initialized chip frequence as 72MHz before, so here we modify it as 72000000, corresponding to the operating frequence of the objective chip.

CFG_SYSTICK_FREQ

This is the system ticks period, we set it as 100 for 10ms,100Hz's system tick clock.

Have done all the work, your program should run normally. Compile your project, you could see the phenomenons we described above after downloading the program to the hardware by our Colink emulator.

2 Task Management

2.1 Task

During OS-based application development, an application is usually seperated into a number of tasks. In CooCox CoOS, a task is a C function whose inside is a infinite loop, it also has the return values and parameters. However, since a task will never return, the returned type must be defined as void. Code 1 shows a typical task structure.

Code 1 An infinite loop task

```
void   myTask (void* pdata)
{
    for(;;)
    {
    }
}
```

Which is different from the C function, the quit of a task is achieved by calling system API function. Once you quit the task only through the ending of the code execution, the system would breakdown.

You can delete a task by calling CoExitTask () or CoDelTask (taskID) in CooCox CoOS. CoExitTask () is called to delete the current running task while CoDelTask (taskID) to delete others. If the incoming parameter is the current task ID, CoDelTask (taskID) can also delete the current task. The concrete use is shown in Code 2.

Code 2 Deleting a task

```
void   myTask0 (void* pdata)
{
    CoExitTask();
}
void   myTask1 (void* pdata)
{
    CoDelTask(taskID);
}
```

2.2 Task State

A task can exist in one of the following states in CooCox CoOS.

Ready State(TASK_READY): Ready tasks are those that are able to execute (they are not waiting or dormant) but are not currently executing because a different task of equal or higher priority is already in the Running state. A task will be in this state after being created.

Running State(TASK_RUNNING): When a task is actually executing it is said to be in the Running state. It is currently occupying the processor.

Waiting State(TASK_WAITING): Wait for an event to occur. A task will be in the waiting state if it is currently waiting for a certain event in CooCox CoOS.

The Dormant State(TASK_DORMANT): The task has been deleted and is not available for scheduling. The dormant state is not the same as the waiting state. Tasks in the waiting state will be reactivated and be available for the scheduling when its waiting events have satisfied. However, tasks in the dormant state will never be reactivated.

The state of a task can be changed among the above four states. You can call CoSuspendTask() to convert a task which in the running or ready state to the waiting state. By calling CoAwakeTask() you can also convert the state of a task from the waiting state to the ready state(as shown in Figure 2.2.1).
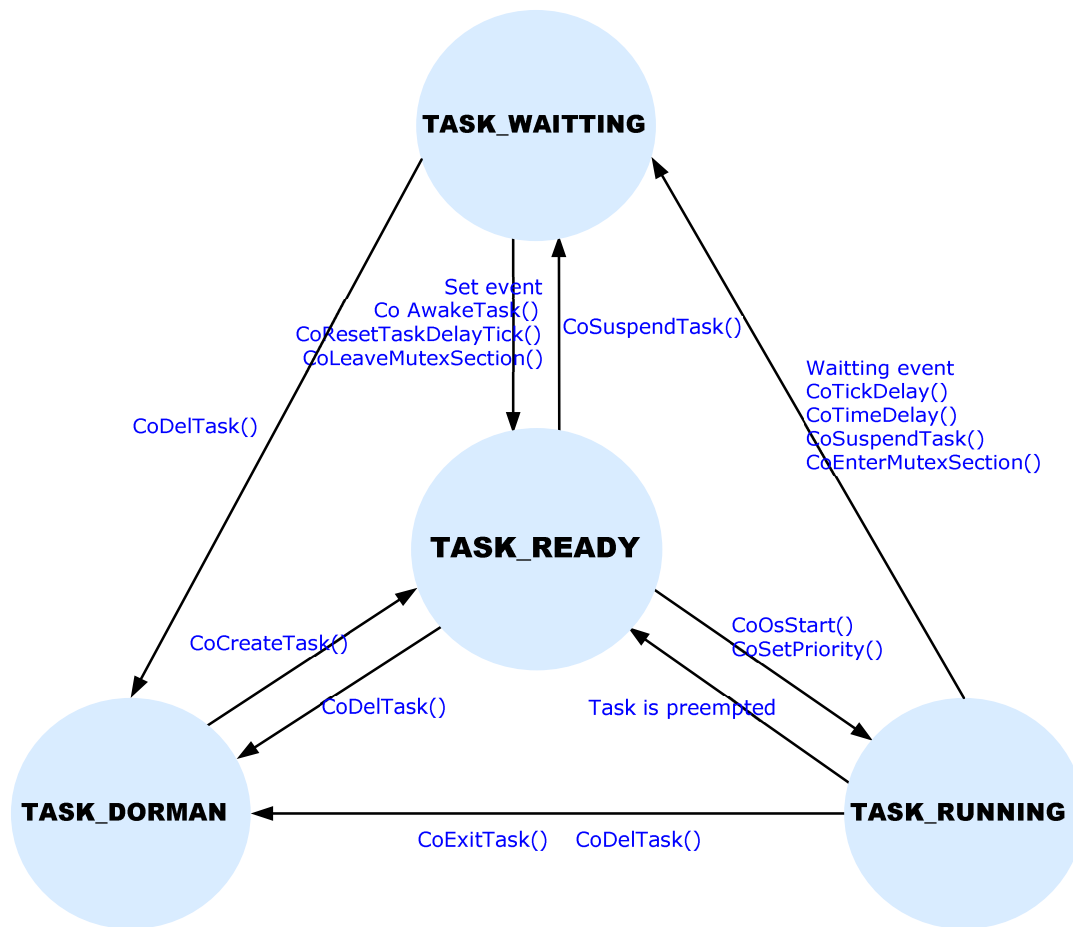
Figure 2.2.1   Valid task state transitions

2.3 Task Control Blocks

Task control block is a data structure used to save the state of a task in CooCox CoOS. Once a task has been created, CooCox CoOS will assign a task control block to describe it (as shown in code 3). This ensures that the task can execute accurately when it obtains the CPU runtime again.

Task control block always accompanies with the task as a description snapshot. It will not be recovered by the system until the task being deleted.

Code 3   Task control block

```
typedef struct TCB
{
    OS_STK *stkPtr; /*!< The current point of task. */
    U8 prio; /*!< Task priority. */
    U8 state; /*!< TaSk status. */
    OS_TID taskID; /*!< Task ID. */
#if CFG_MUTEX_EN > 0
    OS_MutexID mutexID; /*!< Mutex ID. */
#endif
#if CFG_EVENT_EN > 0
    OS_EventID eventID; /*!< Event ID. */
#endif
#if CFG_ROBIN_EN >0
    U16 timeSlice; /*!< Task time slice */
#endif
#if CFG_STK_CHECKOUT_EN >0
    OS_STK *stack; /*!< The top point of task. */
#endif
#if CFG_EVENT_EN > 0
    void* pmail; /*!< Mail to task. */
    struct TCB *waitNext; /*!< Point to next TCB in the Event waitting list.*/
    struct TCB *waitPrev; /*!< Point to prev TCB in the Event waitting list.*/
#endif
#if CFG_TASK_SCHEDULE_EN == 0
    FUNCPtr      taskFuc;
    OS_STK       *taskStk;
#endif
#if CFG_FLAG_EN > 0
    void* pnode; /*!< Pointer to node of event flag. */
#endif
#if CFG_TASK_WAITTING_EN >0
    U32 delayTick; /*!< The number of ticks which delay. */
#endif
    struct TCB *TCBnext; /*!< The pointer to next TCB. */
    struct TCB *TCBprev; /*!< The pointer to prev TCB. */
}OSTCB,*P_OSTCB;
```

stkPtr: A pointer to the current task's top of stack. CooCox CoOS allows each task to have its own stack of any size. During every task switches, CoOS saves the current CPU running state through the stack that stkPtr pointed to so that the task can come back to the previous running state when it gets the CPU runtime again. Since Cortex-M3 has 16 32-bit general-purpose registers to describe the CPU states, the minimum size of stack for a task is 68 bytes(other 4 bytes are used to checking stack overflow).

prio: The task priority that you assigned. Multiple tasks can share the same priority in CooCox CoOS.

state: The state of the task.

taskID: The task ID that system assigned. Since multiple tasks can share the same priority, the priority can not be used as the unique identifier. We use task ID to distinguish different tasks in CooCox CoOS.

mutexID: The mutex ID that the task waiting for.

eventID: The event ID that the task waiting for.

timeSlice: The time slice of the task.

Stack: A pointer to the bottom of a stack. It can be used to check the stack overflow.

Pmail: The message pointer sent to the task.

waitNext: The TCB of the next task in the event waiting list.

waitPrev: The TCB of the previous task in the event waiting list.

taskFuc: Task function pointer, to active task.

taskStk: A pointer to the current task's top of stack, to active task.

Pnode: The pointer to the node of the event flag.

delayTick: The time difference between the previous delayed event and the task when it is in the delayed state.

TCBnext: The next TCB when a task is in the ready list / delayed list / mutex waiting list. Which list the task belongs to is determined by the task state and the item of the TCB. If the current task is in the ready state, it is in the ready list. If it is in the waiting state, then judged by the mutexID and delayTick: If mutexID is not 0xFFFFFFFF, it is in the mutex waiting list; else if delayTick is not 0xFFFFFFFF, it is in the delayed list.

TCBprev: The previous TCB when a task is in the ready list /delayed list /mutex waiting list. Which list the task belongs to is determined by the task state and the relevant event flag.

System will assign a block to the current task from the current free TCB list while creating a task. The free TCB pointer is designated by FreeTCB in CooCox CoOS. If the FreeTCB is NULL, there is no TCB to assign and the task will fail to create.

While system initializing, CoOS will sort all the assignable TCB resources and then reflect the current state of the TCB through the forms of lists, as follows:

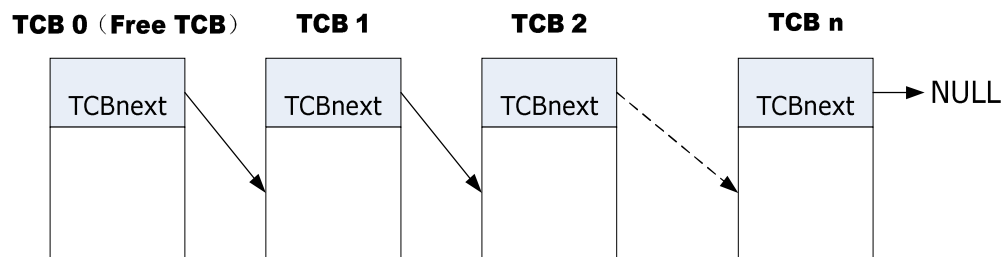Figure 2.3.1 Task control block list

Every time you create a task successfully, the FreeTCB will be assigned to this task and its next item will be the new FreeTCB until it equals NULL. When a task is deleted or exited, the system will recover the TCB which had been assigned to this task when it was created and then assign it as the FreeTCB of the next time so as to reuse the resources of the deleted task.

2.4 Task Ready List

CooCox CoOS links all the TCB of ready tasks together according to the level of the priority through two-way linked list. This ensures that the first item of the list is always the one which has the highest priority and is the most in need of task scheduling.

CooCox CoOS allows multiple tasks to share the same priority level. Therefore, tasks with the same priority will inevitably occur in the ready list. CooCox CoOS follows the principle "first-in-first out (FIFO)": put the latest task in the last of the tasks which share the same priority so that all of them can obtain its own CPU runtime.

TCBRdy is the beginning of the ready list in CooCox CoOS. In other words, TCBRdy is the TCB of the task which has the highest priority in the ready list. Therefore, when starting a task scheduling, which only need to be checked is whether the priority of the task that TCBRdy pointed to is higher than the current running one. In this way, the efficiency of the task scheduling can be improved to the maximum.
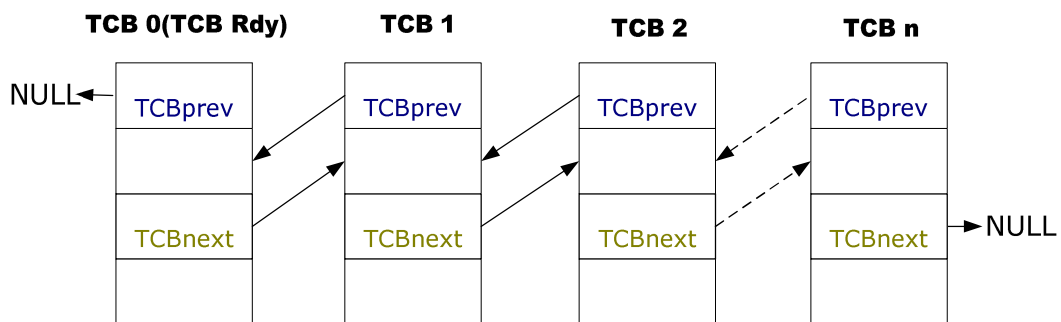

Figure 2.4.1 Task ready list

2.5 Task Scheduling

CooCox CoOS supports two kinds of scheduling mode, preemptive priority and round-robin. The former is used among tasks of different priority, while the latter among tasks of the same priority.

CooCox CoOS will start a task scheduling in the following three situations:

1) A task whose priority is higher than the current running one is converting to the ready state;

2) The current running task is changing from the running state to the waiting or dormant state;

3) A task sharing the same priority with the current running task is in the ready state, and meanwhile the time slice of the current task runs out .

When a system tick interrupt exits or some tasks' states have changed, CooCox CoOS will call the task scheduling function to determine whether it is essential to start a task scheduling or not.

For the scheduling of tasks sharing the same priority, the system starts the rotation scheduling according to the time slice of each task. When the system has run out the time slice of the current task, it will give the right of control to the next task with the same priority. Figure 2.5.1 shows the system running state of the three tasks (A, B, C with their respective time slices 1, 2, 3) with the same priority when they are entering the ready state in turn.
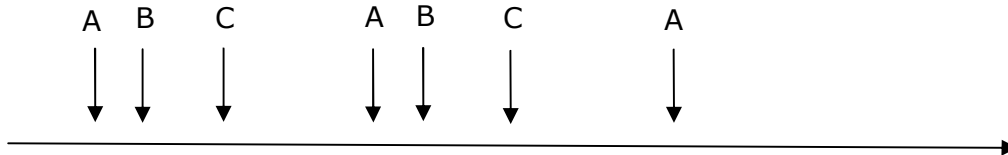
Figure 2.5.1 Round-robin time slice

In CooCox CoOS source codes, the implementing of task scheduling is shown as follows:

16

Code 4 Task with a higher priority is ready

```
/* Is higher PRI task coming in? */
if(RdyPrio < RunPrio )
{
    TCBNext         = pRdyTcb;   /* Yes, set TCBNext and reorder ready list*/
    pCurTcb->state  = TASK_READY;
    pRdyTcb->state  = TASK_RUNNING;
    InsertToTCBRdyList(pCurTcb);
    RemoveFromTCBRdyList(pRdyTcb);
}
```

Code 5 The state of the current task changes

```
/* Does Running task status change */
else if(pCurTcb->state != TASK_RUNNING)
{
    TCBNext         = pRdyTcb;   /* Yes, set TCBNext and reorder ready list*/
    pRdyTcb->state  = TASK_RUNNING;
    RemoveFromTCBRdyList(pRdyTcb);
}
```

Code 6 The task scheduling among the same priority tasks

```
/* Is it the time for robinning */
else if((RunPrio == RdyPrio) && (OSCheckTime == OSTickCnt))
{
    TCBNext         = pRdyTcb;    /* Yes, set TCBNext and reorder ready list*/
    pCurTcb->state  = TASK_READY;
    pRdyTcb->state  = TASK_RUNNING;
    InsertToTCBRdyList(pCurTcb);
    RemoveFromTCBRdyList(pRdyTcb);
}
```

2.6 Critical Section

Different from other kernels, CooCox CoOS does not handle the critical code section by closing interrupts, but locking the scheduler. Therefore, CoOS has a shorter latency for interrupt compared with others.

Since the time of enabling the interrupt relates to system's responsiveness towards the real-time events, it is one of the most important factors offered by the real-time kernel developers. By locking the scheduler we can improve system's real-time feature to the maximum comparing to other approaches.

Since CooCox CoOS manages the critical section by forbidding to schedule task, user's applications cannot call any API functions which will suspend the current running task in critical sections, such as CoExitTask (), CoSuspendTask (), CoTickDelay (), CoTimeDelay (), CoEnterMutexSection (), CoPendSem (), CoPendMail (), CoPendQueueMail (), CoWaitForSingleFlag (), CoWaitForMultipleFlags () and so on.

Code 7 Critical Section

```
void Task1(void* pdata)
{
    ....................
    CoSchedLock();              // Enter Critical Section
    ...............             // Critical Code
    CoSchedUnlock();            // Exit Critical Section
    ....................
}
```

2.7 Interrupts

In CooCox CoOS, the interrupt is divided into two categories according to whether called the system API functions inside or not.

For the ISR which has nothing to do with OS, CooCox CoOS does not force it to do anything and you can operate just like there is not an OS.

However, for the ISR which called the system API functions inside, CooCox CoOS demands that you call the relevant functions when entering or exiting the interrupt (as shown in code 8).

Code 8 The interrupt handler which called the system API

```
void   WWDG_IRQHandler(void)
{
    CoEnterISR();            // Enter the interrupt
    isr_SetFlag(flagID);     // API function
    .................;       // Interrupt service routine


    CoExitISR();             // Exit the interrupt
}
```

All the system API which can be called in the interrupt service routine begin with isr_, such as isr_PostSem (), isr_PostMail (), isr_PostQueueMail () and isr_SetFlag ().The calling of any other API inside the ISR will lead to the system chaos.

When calling the corresponding API functions in the interrupt service routine, system need to determine whether the task scheduling is locked or not. If it is unlocked, system can call it normally. Otherwise, system will send a relevant service request to the service request list and then wait for the unlocking of the scheduler to respond it.

3 Time Management

3.1 System Ticks

CooCox CoOS uses interrupt systick to implement system tick. You need to configure the frequency of system tick in config.h file. CFG_CPU_FREQ is used for CPU's clock frequency. The system needs to determine the specific parameters through CPU's clock frequency while configuring systick. CFG_SYSTICK_FREQ is used for the frequency of system tick that users need. CooCox CoOS supports the frequency from 1 to 1000Hz. The actual value is determined by the specific application while the default value is 100Hz (that is, the time interval is 10ms).

CooCox CoOS increases the system time by 1 in every system tick interrupt service routine, you can get the current system time by calling CoGetOSTime().

CooCox CoOS will also check whether the delayed list and the timer list is empty in system tick interrupt service routine except increasing the system time by 1.If the list is not empty, decrease the delayed time of the first item in the list by 1, and judge whether the waiting time of the first item in the list is due. If it is due, call the corresponding operation function, otherwise, skip to the next step.

CooCox CoOS calls the task scheduling function to determine whether the current system needs to run a task scheduling when exits from the system tick interrupt service.

Code 1 System tick interrupt handling

```c
void SysTick_Handler(void)
{
    OSSchedLock++;                  /* Lock the scheduler. */
    OSTickCnt++;                    /* Increment system time. */
    if(DlyList != NULL)             /* Have task in delayed list? */
    {
        DlyList->delayTick--;       /* Decrease delay time of the list head. */
        if(DlyList->delayTick == 0) /* Delay time == 0? */
        {
            isr_TimeDispose();      /* Call hander for delay time list */
        }
    }
#if CFG_TMR_EN > 0
    if(TmrList != NULL)             /* Have timer be in working? */
    {
        TmrList->tmrCnt--;          /* Decrease timer time of the list head. */
        if(TmrList->tmrCnt == 0)    /* Timer time == 0? */
        {
            isr_TmrDispose();       /* Call hander for timer list. */
        }
    }
#endif
    OSSchedLock--;                   /* Unlock scheduler. */
    if(OSSchedLock==0)
    {
        Schedule();                  /* Call task scheduler */
    }
}
```

3.2 Delay Management

CooCox CoOS manages all the tasks' delay and timeout through delayed list. When you call CoTickDelay(),CoTimeDelay(),CoResetTaskDelayTick() or other API functions to apply for delay. CooCox CoOS will sort the delay time from short to long, and then insert them into the delayed list. The delayTick item in the task control block preserves the difference value of the delay time between the current task and the previous task. The first item in the list is the value of the delay time or the timeout value, while the subsequent item is the difference value with the former. For example, Task A, B, C delay 10,18,5 respectively, then they will be sequenced as follows in the delayed list:

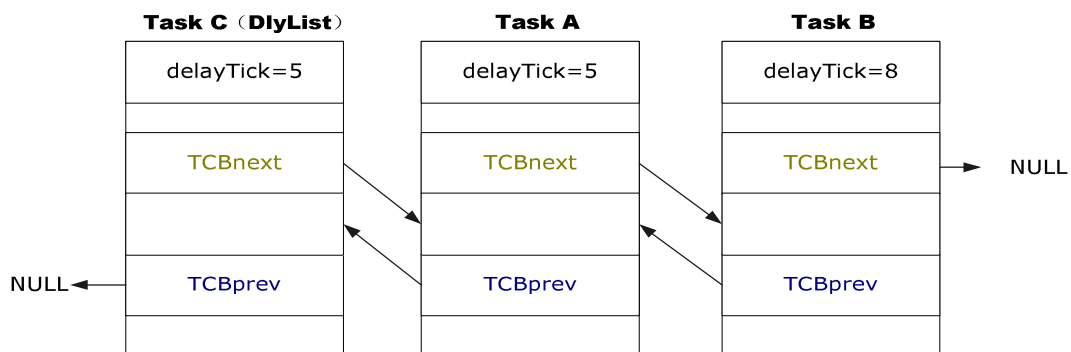| Task C（DlyList）	Task A	Task B
delayTick=5	delayTick=5	delayTick=8
TCBnext	TCBnext	TCBnext → NULL
NULL ← TCBprev	TCBprev	TCBprev

Figure 3.2.1 Task delayed list

System will decrease the first item of the delayed list by 1 in every system tick interrupt, and then move it from the delayed list to the ready list until it becomes 0. When moving the tasks whose time is due out of the list in system tick interrupt, system should determine whether the task is a delay operation or a timeout operation. Towards the tasks with a delay operation, CooCox CoOS moves it to the ready list after it being moved out from the delayed list. Towards the tasks with a timeout operation, CooCox CoOS will judge which event leads to the overtime first, and then move the task from the waiting list to the ready list.

CooCox CoOS's system delay can't guarantee the accuracy under the following conditions: '

1)There is a higher priority task preempting to run when task A is delayed, the delay time will be determined by the running time of the higher priority task;

2)There is a task whose priority is the same to task A preempting to run when task A is delayed, the delay time will be determined by the number of the tasks whose priority are the same to task A in the ready list and the length of their timeslice, as well as the time that the delay time is due.

3.3 Software Timer

Software timer is a high precision timer that CooCox CoOS takes the system tick as the benchmark clock source. CooCox CoOS software supports up to 32 software timers, the operation mode of each timer can be set to the periodic mode or the one-shot mode.

You can create a software timer by calling CoCreateTmr(). When creating a software timer, the system will assign a corresponding timer control block to describe the current state of the timer. After being created successfully, the timer's default state is stopping. It won't work normally until you call CoStartTmr(). To the timer which works normally, CooCox CoOS manages them through the timer list. The same as the delayed list, the timer list is also sorted by the length of the time that is due: the task whose time is due earlier will be in front of the list, and subtract expiration time of the previous timer in the list from expiration time of all the timers, and then save the result into the timer control block. For example, Task A, B, C are set to 10,18,5 respectively, then they will be sequenced as follows in the timer list:
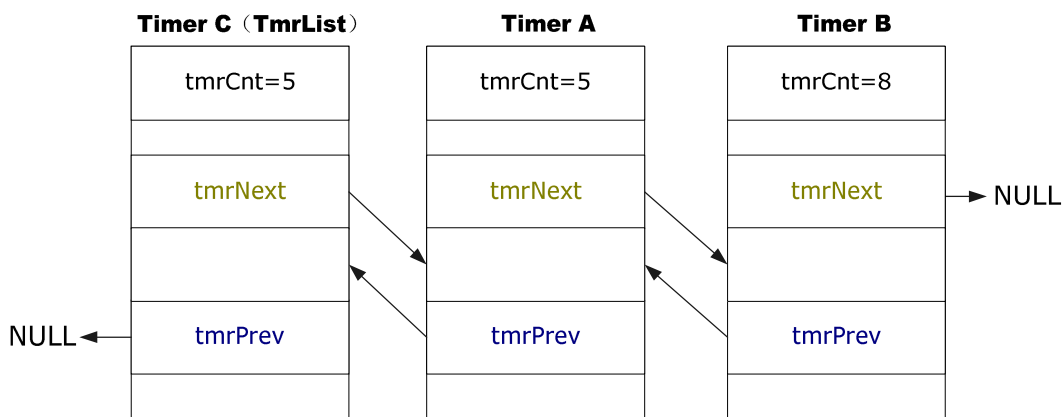
Figure 3.3.1 Timer list

Once the software timer starts, it will be independent to other modules completely, and only be related to the system tick interrupt. CooCox CoOS takes all the timers into the timer list by the length of the time that is due. Decrease the first item of the timer list by 1(until it becomes 0) in every sysytem tick interrupt.

When the waiting time of the timer is due, towards the periodic timer CooCox CoOS will reset the next timing according to the tmrReload that you set and then put it into the timer list. However, to the one-shot timer, it will be moved out of the list, and its state will be set to stop.

From the above, we know that the expiration time of the timer is only determined by the number of system ticks, but has no relation to whether

there is a higher priority task running or whether the task which created the timer is in the ready state.

Software timer provides a function entrance (software timer callback function) for your operation inside the systick interrupt. But some of the APIs which will cause some errors can't be called. Many APIs can not be called in the following situation:

1.Should not call the APIs whose functions do not match, such as CoEnterISR()\CoExitISR(), because the software timer callback function is not a ISR.

2.Software timer callback function is not a task but a function may be called in all tasks, so the APIs which will change the state of the current task (running task) should not be called, such as CoExitTask() \ CoEnterMutexSection() \ CoLeaveMutexSection() \ CoAcceptSem() \ CoPendSem() \ CoAcceptMail() \ CoPendMail() \ CoAcceptQueueMail() \ CoPendQueueMail() \ CoAcceptSingleFlag() \ CoAcceptMultipleFlags() \ CoWaitForSingleFlag() \ CoWaitForMultipleFlags().

3.Each timer's callback function is implemented inside the systick interrupt when it is due, which requires the code of the software timer must be simplified and should not run a long time to affect the precision of the systick interrupt. So users should not call CoTimeDelay() and CoTickDelay() which will not only affect the precision of the systick interrupt but also cause the kernel error.

4 Memory Management

4.1 Static Memory Allocation

Static memory allocation applies to the condition that you know how much memory you need to occupy while compiling, and the static memory can't be released or redistributed while system running. Compared with the dynamic memory allocation, static memory allocation does not need to consume CPU resources. At the same time, it would not lead to allocate unsuccessfully (because allocating unsuccessfully will directly result in the failure of the compiler). Therefore it is faster and more secure.

In CooCox CoOS, the memory that each module's control block needs is allocated statically, such as the task control block(TCB),event control block(ECB) ,flag control block(FCB),flag node(FLAG_NODE)and so on.

Code 1 CooCox CoOS's space allocation of the TCB

```
config.h
    #define CFG_MAX_USER_TASKS (8)   // Determine the largest number of
                                     // user's task in the system
task.h
    #define   SYS_TASK_NUM        (1)    // Determine the number of system tasks
    task.c
                                     // Allocate TCB space statically
OSTCB   TCBTbl[CFG_MAX_USER_TASKS+SYS_TASK_NUM];
```

4.2 Dynamic memory management

Dynamic memory allocation applies to the conditions that the memory size can not be determined while compiling but by the runtime environment of the code during the system is running. It could be said that the static memory allocation is based on plans while the dynamic memory allocation is based on need.

Dynamic memory allocation is more flexible, and can greatly improve the utilization of the memory. However, since every allocation and release will consume CPU resources, and there may be a problem of allocation failure and memory fragmentation, you need to determine whether the allocation is successful or not during every dynamic memory allocation.

Here are some conventional methods of dynamic memory allocation—— the implementation of malloc() and free(). In the usual kernel or compiler, the memory block is allocated to the free list or the list of allocation respectively according to its allocation condition.

The related steps of the system are as follows while calling malloc():

1)Find a memory block whose size meets user's demand. Since the search algorithm is different, the memory blocks found by the system are not the same. The most commonly used algorithm is the first matching algorithm, that is, allocate the first memory block which meets user's demand. By doing this, it could avoid to traverse all the items in the free list during each allocation.

2)Divide the memory block into two pieces: the size of first piece is the same to user's demand, and the second one storages the remaining bytes.

3)Pass the first piece of memory block to the user. The other one (if any) will be returned to the free list.

System will link the memory block that you released to the free list, and determine whether the memory's former or latter memory is free. If it is free, combine them to a larger memory block.

Assigned memory block list
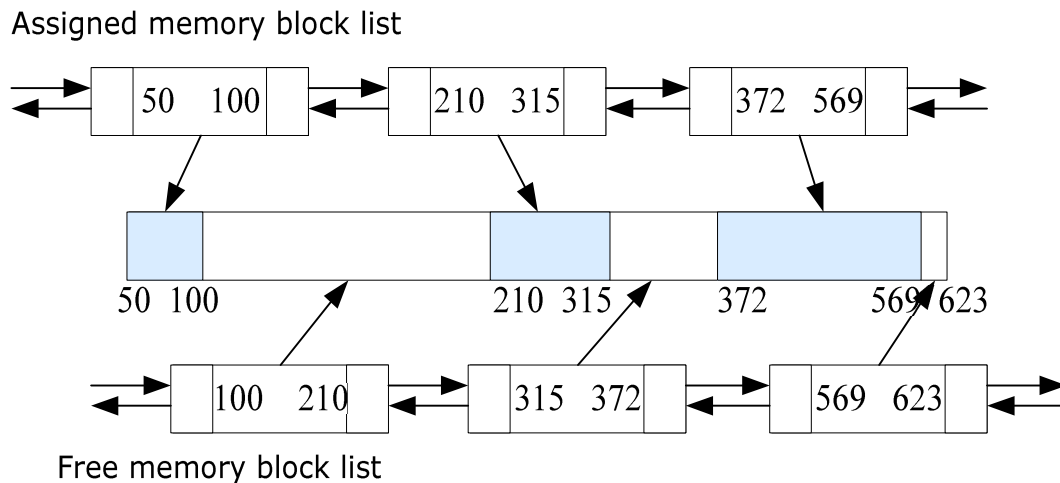
Figure 4.2.1 Memory management list

From the above, we can see that the free list will be divided into many small pieces after allocating and releasing the memory many times. If you want to apply for a large memory block at this time, the free list may not have the fragment that meets user's demand. This will lead to the so-called memory fragmentation. In addition, the system needs to check the whole free list from the beginning to determine the location that the memory block needs to plug in, which leads the time of releasing the memory too long or uncertain.

CooCox CoOS provides two mechanisms of partitioning to solve these problems: the partition of fixed length and the partition of variable length.

4.2.1 Fixed-length partition

It provides memory allocation mode of fixed length partition in CooCox CoOS. The system will divide a large memory block into numbers of pieces whose size are fixed, then link these pieces through the linked list. You can allocate or release the memory block of fixed length through it. In this way, we not only can ensure that the time of allocation or release is fixed, but also can solve the problem of memory fragmentation.
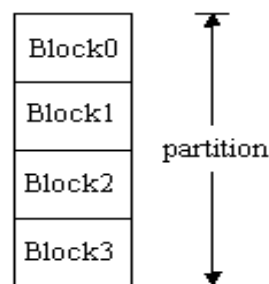
P4.2.2 Partition of fixed length

CooCox CoOS can manage a total of 32 fixed-length partitions of different size. You can create a partition of fixed length by calling CoCreateMemPartition(). After being created successfully, you can allocate or release the memory block by calling CoGetMemoryBuffer() and CoFreeMemoryBuffer(). You can also get the number of free memory blocks in current memory partition by calling CoGetFreeBlockNum().

Code 2   The creation and use of fixed-length partition

```
U8 memPartition[128*20];
OS_MMID memID;
void   myTask (void* pdata)
{
    U8* data;
    memID = CoCreateMemPartition(memPartition,128,20);
    if(CoGetFreeBlockNum(memID ) != 0)
    {
        data = (U8*)CoGetMemoryBuffer(memID );
    }
    .................
    CoFreeMemoryBuffer(memID ,data);
}
```

4.2.2 Variable-length partition

From the implement of conventional dynamic memory shown above, we can see that it need to operate the free list and the allocated list at the same time while release the memory, which require a long operation time and has impaction to the CPU. For these reasons, CooCox CoOS redesigns the list to ensure that both the allocation and release of memory only require to search just one list.
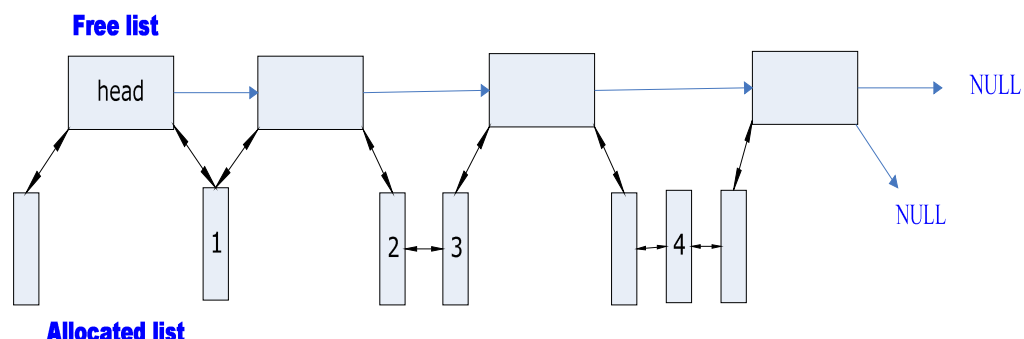

Figure 4.2.3 List of variable-length partition

From the figure we can see that all the free memory in the system can separately form a one-way list so that it is more convenient to look up the list

while allocate the memory. To all the memory blocks, whether they are free or have been allocated, CooCox CoOS will link them through a doubly linked list. Thus, when an allocated memory releases, there is no need to find the insertion point of the memory from the beginning of the free list. You only need to find the former free block in the doubly linked list and then insert the memory into the free list, which has greatly improved the speed of memory releasing.

In CooCox CoOS, since all the memory is 4-byte alignment (if the space that needs to allocate is not 4-byte alignment, force it to be), the last two bits of the previous and next memory address that saved in the head of the memory are all invalid. Therefore, CooCox CoOS determines whether the memory is free through the least significant bit: bit 0 refers to the free block, otherwise refers to the allocated ones. If the memory list points to the free block, get the list address directly. If it points to the allocated one, decrease it by one.

Code 3 The head of allocated memory block

```
typedef struct UsedMemBlk
{
  void* nextMB;
  void* preMB;
}UMB,*P_UMB;
```

Code 4   The head of free memory block

```
typedef struct FreeMemBlk
{
  struct FreeMemBlk* nextFMB;
  struct UsedMemBlk* nextUMB;
  struct UsedMemBlk* preUMB;
}FMB,*P_FMB;
```

For memory block 1 and 2, the memory block itself preserves the address of the previous free memory block when released, so it is very easy to plug back to the free list. You only need to decide whether to merge them according to if the memory block's next block is free.

For memory block 3 and 4, its previous memory block is not a free memory block when released. It is essential to get the previous free memory address through two-way list when plug it back to the free list.

Code 5 To get the address of the previous free memory block

```
P_FMB GetPreFMB(P_UMB usedMB)
{


    P_UMB preUMB;
    preUMB = usedMB;
    while(((U32)(preUMB->preMB)&0x1))    /* Is previous MB as FMB?*/
    {                                    /* No, get previous MB */
        preUMB = (P_UMB)((U32)(preUMB->preMB)-1);
    }
    return (P_FMB)(preUMB->preMB);       /* Yes, return previous MB*/
}
```

In the file config.h, you can determine whether it is essential to add variable-length partition to the kernel, and set the size of the memory partition at the same time.

Code 6 config.h file

```
Config.h
#define CFG_KHEAP_EN            (1)
#if CFG_KHEAP_EN >0
#define KHEAP_SIZE              (50)        // size(word)
#endif
```

You could implement the allocation and release of the memory by calling CoKmalloc() and CoKfree() respectively after ensuring enabling the variable-length partition. The memory size that CoKmalloc() applied is in bytes.

Code 7 The use of the variable-length partition

```
void myTask (void* pdata)
{
    void* data;
    data = CoKmalloc(100);
    ......
    CoKfree(data );
}
```

4.3 Stack Overflow Check

Stack Overflow refers to that the size of the stack used when a task is running exceeds the size that assigned to the task, which results in writing data to the memory outside the stack.   This may lead to the coverage of the system or other tasks' data as well as the exception of memory access. The stack size assigned to each task is fixed in multi-tasking kernel. Once the stack overflow is not handled when the system is running, it may lead to system crashes.

When creating a task in CooCox CoOS, the system will save the stack bottom address in the task control block and write a special value into the memory block of the stack bottom address in order to judge whether the stack overflows. CooCox CoOS will check whether there is a stack overflow during each task scheduling.

Code 8   Stack overflow inspection

```
if((pCurTcb->stkPtr < pCurTcb->stack)||(*(U32*)(pCurTcb->stack) != MAGIC_WORD))
{
    CoStkOverflowHook(pCurTcb->taskID);         /* Yes,call hander */
}
```

When stack overflow in a task, the system will call CoStkOverflowHook (taskID) automatically. You can add the handling of stack overflow in the function. The parameter of this function is the ID of the task which has stack overflow.

Code 9   The handling function of stack overflow

```
void CoStkOverflowHook(OS_TID taskID)
{
    /* Process stack overflow in here */
    for(; ;)
    {
        …
    }
}
```

5 Intertask Synchronization&Communication

5.1 Intertask Synchronization

Intertask synchronization refers to that one task can only keep on executing after it has got the synchronization signal sent by another task or the ISR. There are semaphores, mutexes and flags to implement intertask synchronization in CooCox CoOS.

5.1.1 Semaphores

Semaphores provide an effective mechanism for the system to handle the critical section and implement intertask synchronization.

The action of semaphore can be described as the classical PV operation:

P Operation: while(s==0); s--;
V Operation: s++;

You can create a semaphore by calling CoCreateSem () in CooCox CoOS. After the semaphore has been created successfully, you can obtain it by calling CoPendSem () or CoAcceptSem (). If there is no free semaphore, CoPendSem () will wait for a semaphore to be released while CoAcceptSem () will return the error immediately. You can also call CoPostSem () in the task or isr_PostSem () in the ISR to release a semaphore for the purpose of achieving synchronization.

Code 1   The creation of the semaphore

```
ID0 = CoCreateSem(0,1,EVENT_SORT_TYPE_FIFO); // initCnt=0,maxCnt=1,FIFO
ID1 = CoCreateSem(2,5,EVENT_SORT_TYPE_PRIO); // initCnt=2,maxCnt=5,PRIO
```

Code 2   The use of the semaphore

```
void myTaskA(void* pdata)
{
    ..........
    semID = CoCreateSem(0,1,EVENT_SORT_TYPE_FIFO);
    CoPendSem(semID,0);
    ..........
}
void myTaskB(void* pdata)
{
    ......
    CoPostSem(semID);
    ......
}
void myISR(void)
{
    CoEnterISR();
    ......
    isr_PostSem(semID);
    CoExitISR();
}
```

5.1.2 Mutexes

Mutexes have solved the "mutually exclusion" problem in CooCox CoOS. It forbids multiple tasks to enter the critical code section at the same time. Therefore, only one task can enter it at any time.

In CooCox CoOS, the mutex section has also considered the issue of priority inversion. CooCox CoOS has solved this issue by the method of priority inheritance.

Priority inversion refers to that the high-priority task is waiting for the low-priority task to release resources, at the same time the low-priority task is waiting for the middle priority task's.

There are two classical methods to prevent the inversion at present:

1) The priority inheritance strategy: The task which is possessing the critical section inherits the highest priority of all the tasks that request for this critical section. When the task exits from the critical section, it will restore to its original priority.

2) The ceiling priority strategy: Upgrade the priority of the task which requests a certain resource to the highest priority of all the tasks that be likely

to access this resource (and the highest priority is called the ceiling priority of this resource).

The priority inheritance strategy has a less impact to the flow of the task execution since it only upgrades the priority of the low-priority task when a high-priority task is requesting for the critical resource that being occupied by the low-priority task. However, the ceiling priority strategy upgrades one task's priority to the highest when the task is occupying the critical resource.

CooCox CoOS prevents the priority inversion by the method of priority inheritance.

The following figure describes the task scheduling of three tasks when there are mutex sections in CooCox CoOS. TaskA has the highest priority while TaskC has the lowest. The blue boxes refer to the mutex sections.
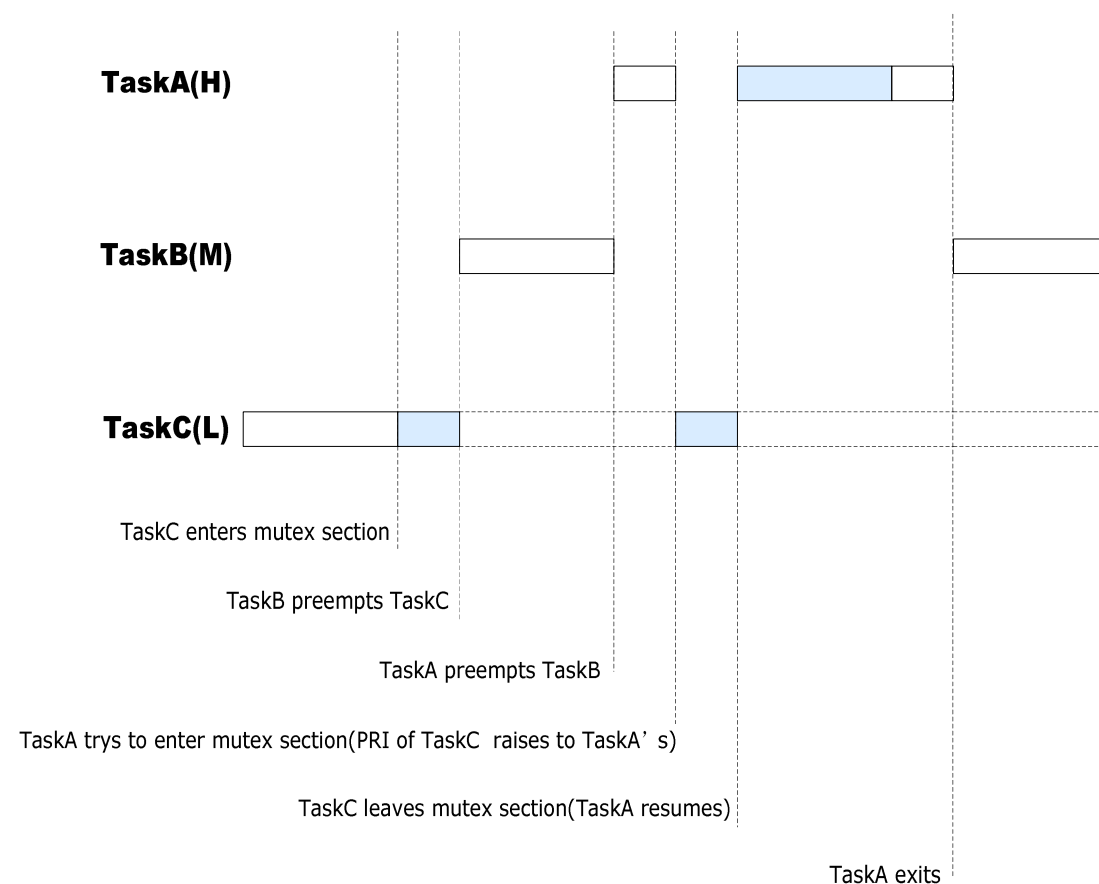

Figure 5.1.1 Task scheduling with mutex sections

You can create a mutex section by calling CoCreateMutex(). Calling CoEnterMutexSection() and CoLeaveMutexSection() to enter or leave the mutex section so that we can protect the codes in critical section.

Code 3   The use of the mutex section

```
void myTaskA(void* pdata)
{
    mutexID = CoCreateMutex();
    CoEnterMutexSection(mutexID );    // enter the mutex section
    ...........                        // critical codes
    CoLeaveMutexSection(mutexID );    // leave the mutex section
}
void myTaskB(void* pdata)
{
    CoEnterMutexSection(mutexID );    // enter the mutex section
    ...........                        // critical codes
    CoLeaveMutexSection(mutexID );    // leave the mutex section
}
```

5.1.3 Flags

When a task wants to synchronize with a number of events, flags are needed. If the task synchronizes with a single event, it can be called independent synchronization (logical OR relationship). If it synchronizes with a number of events, then called associated synchronization (logical AND relationship).

CooCox CoOS supports 32 flags to the maximum at the same time. It supports that multiple tasks waiting for a single event or multiple events. When the flags that the waiting tasks waiting for are in the not-ready state, these tasks can not be scheduled. However, once the flags turn to the ready state, they will be resumed soon.

According to the types of the flags, the side effects are different when the tasks have waited for the flags successfully. There are two kinds of flags in CooCox CoOS: the ones reset manually and the ones reset automatically. When a task has waited for a flag which reset automatically, the system will convert the flag to not-ready state. On the contrary, if the flag is reset manually, there won't be any side effect. Therefore, when a flag which reset manually converts to the ready state, all the tasks which waiting for this event will convert to the ready state as far as you call CoClearFlag() to reset the flag to the not-ready state. When a flag which reset automatically converts to the ready state, only one task which waiting for this event will convert to the ready state. Since the waiting list of the event flags is ordered by the principle of FIFO, towards the event which reset automatically only the first task of the waiting list converts to the ready state and others that waiting for this flag are still in the waiting state.

Suppose there are three tasks (A, B, C) waiting for the same flag I which reset manually. When I is ready, all the tasks will be converted (A, B, C) to the

ready state and then inserted into the ready list. Suppose I is a flag which reset automatically and the tasks (A, B, C) are listed in sequence in the waiting list. When I is ready, it will inform task A. Then I will be converted to the not-ready state. Therefore B and C will keep waiting for the next ready state of flag I in the waiting list.

You can create a flag by calling CoCreateFlag() in CooCox CoOS. After being created, you can call CoWaitForSingleFlag() and CoWaitForMultipleFlags() to wait for a single flag or multiple flags.

Code 4   Wait for a single flag

```
void myTaskA(void* pdata)
{
    ..........
    flagID = CoCreateFlag(0,0);        // Reset manually, the original state is not-ready
    CoWaitForSingleFlag(flagID,0);
    ..........
}
void myTaskB(void* pdata)
{
    ......
    CoSetFlag(flagID);
    ......
}
```

Code5   Wait for multiple flags

```
void myTaskA(void* pdata)
{
    U32 flag;
    StatusType err;
    ..........
    flagID1 = CoCreateFlag(0,0);    // Reset manually, the original state is not-ready
    flagID2 = CoCreateFlag(0,0);    // Reset manually, the original state is not-ready
    flagID3 = CoCreateFlag(0,0);    // Reset manually, the original state is not-ready
    flag = flagID1 | flagID2 | flagID3;
    CoWaitForMultipleFlags(flag,OPT_WAIT_ANY,0,&err);
    ..........
}
void myTaskB(void* pdata)
{
    ......
    CoSetFlag(flagID1);
    ......
}
void myISR(void)
{
    CoEnterISR();
    ......
    isr_SetFlag(flagID2);
    CoExitISR();
}
```

5.2 Intertask Communication

Information transfer is sometimes needed among tasks or between the task and the ISR. Information transfer can be also called intertask communication. There are two ways to implement it: through the global variable or by sending messages.

When using the global variable, it is important to ensure that each task or ISR possesses the variable alone. The only way to ensure it is enabling the interrupt. When two tasks share one variable, each task possesses the variable alone through firstly enabling then disabling the interrupt or by the semaphore (see chapter 5.1). Please note that a task can communicate with the ISR only through the global variable and the task won't know when the global variable has been modified by the ISR (unless the ISR sends signals to the task in manner of semaphore or the task keeps searching the variable's value). In this case, CooCox CoOS supplies the mailboxes and the message queues to avoid the problems above.

5.2.1 Mailboxes

System or the user code can send a message by the core services. A typical mail message, also known as the exchange of information, refers to a task or an ISR using a pointer variable, through the core services to put a message (that is, a pointer) into the mailbox. Similarly, one or more tasks can receive this message by the core services. The tasks sending and receiving the message promise that the content which the pointer points to is just that piece of message.

MAILBOX

Post → Pend →

Task ISR   Task

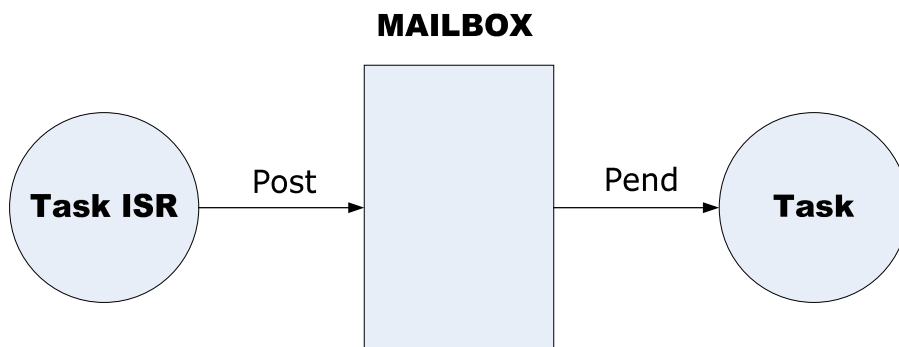Figure 5.2.1 Mailbox

The mailbox of CooCox CoOS is a typical message mailbox which is composed of two parts: one is the information which expressed by a pointer of void; the other is the waiting list which composed of the tasks waiting for this mailbox. The waiting list supports two kinds of sorting: FIFO and preemptive priority. The sorting mode is determined by the user when creating the mailbox.

You can create a mailbox by calling CoCreateMbox () in CooCox CoOS. After being created successfully, there won't be any message inside. You can send a message to the mailbox by calling CoPostMail () or isr_PostMail () respectively in a task or the ISR. You can also get a message from the mailbox by calling CoPendMail () or CoAcceptMail ().

Code 6   The use of the mailbox

```
void myTaskA(void* pdata)
{
    void* pmail;
    StatusType err;
    ..........
    mboxID = CoCreateMbox(EVENT_SORT_TYPE_PRIO); //Sort by preemptive priority
    pmail = CoPendMail(mboxID,0,&err);
    ..........
}
void myTaskB(void* pdata)
{
    ......
    CoPostMail(mboxID,"hello,world");
    ......
}
void myISR(void)
{
    CoEnterISR();
    ......
    isr_PostMail(mboxID,"hello,CooCox");
    CoExitISR();
}
```

5.2.2 Message Queues

Message queue is just an array of mailboxes used to send messages to the task in fact. The task or the ISR can put multiple messages (that is, the pointers of the message) to the message queue through the core services. Similarly, one or more tasks can receive this message by the core services. The tasks sending and receiving the message promise that the content that the pointer points to is just that piece of message.

The difference between the mailbox and the message queue is that the former can store only one piece of message while the latter can store multiple of it. The maximum pieces of message stored in a queue are determined by the user when creating the queue in CooCox CoOS.

In CooCox CoOS, message queue is composed of two parts: one is the struct which pointed to the message queue; the other is the waiting list which composed of the tasks waiting for this message queue. The waiting list supports two kinds of sorting: FIFO and preemptive priority. The sorting mode is determined by the user when creating the message queue.

You can create a message queue by calling CoCreateQueue() in CooCox CoOS. After being created successfully, there won't be any message inside. You can send a message to the message queue by calling CoPostQueueMail() or isr_PostQueueMaill() respectively in a task or the ISR. Similarly, you can also obtain a message from the message queue by calling CoPendQueueMail() or CoAcceptQueueMail().

Code 7   The use of the message queue

```
void myTaskA(void* pdata)
{
    void* pmail;
    Void* queue[5];
    StatusType err;
    ..........
    queueID = CoCreateQueue(queue,5,EVENT_SORT_TYPE_PRIO);
                              //5 grade, sorting by preemptive priority
    pmail = CoPendQueueMail(queueID ,0,&err);
    ..........
}
void myTaskB(void* pdata)
{
    ......
    CoPostQueueMail(queueID ,"hello,world");
    ......
}
void myISR(void)
{
    CoEnterISR();
    ......
    isr_PostQueueMail(queueID ,"hello,CooCox");
    CoExitISR();
}
```

6 API Reference

6.1 System Management

6.1.1 CoInitOS()

Function Prototype:
   void   CoInitOS (void);
Descriptions:
   Initialize the system.
Parameters:
   None
Returns:
   None
Example usage:

```
#include "CCRTOS.h"
#define TASK0PRIO    10
OS_STK Task0Stk[100];
OS_TID Task0Id;
void Task0 (void *pdata);
int main(void)
{
  System_init ();
  CoInitOS ();                // Initialize CoOS

   …
Task0Id = CoCreateTask (Task0, (void *)0, TASK0PRIO , &Task0Stk[99], 100);

   …
  CoStartOS();               // Start CoOS
  }
void Task0 (void *pdata)
{
  …
  for(;;)
  {
      …
  }
}
```

Note:
   1) A requirement of CooCox CoOS is that you call CoInitOS() before you
      call any of its other services.
   2) You need to set the CPU clock and configure the OS well before

calling CoInitOS().

3) When initializing the OS, the system will close the OS scheduler and create the first task-IdleTask.

6.1.2 CoStartOS()

Function Prototype:
    void   CoStartOS(void);
Descriptions:
    System start running.
Parameters:
    None
Returns:
    None
Example usage:

```
#include "CCRTOS.h"
#define TASK0PRIO    10
OS_STK Task0Stk[100];
OS_TID Task0Id;
void Task0 (void *pdata);
int main(void)
{
  System_init();
  CoInitOS();                // Initialize CoOS
   …
Task0Id = CoCreateTask (Task0, (void *)0, TASK0PRIO , &Task0Stk[99], 100);
   …
  CoStartOS();               // Start CoOS
  }
void Task0 (void *pdata)
{
  …
  for(;;)
  {
      …
  }
}
```

Note:

1) Before calling CoStartOS(), you must create at least one of your application tasks, or OS would stay in CoIdleTask () all the time.

2) When the OS starts, the task scheduler is unlocked and then start the first task scheduling.

6.1.3 CoEnterISR()

Function Prototype:
   void   CoEnterISR(void);
Descriptions:
   System enters the interrupts.
Parameters:
   None
Returns:
   None
Example usage:

```
#include "CCRTOS.h"
void EXTI0_IRQHandler(void)
{
  CoEnterISR();         // Enter ISR

      …

  /* Process interrupt here */

      …

  CoExitISR();          // Exit ISR
}
```

Note:
   1) When the system enters the interrupts, increase the interrupt nesting counter- OSIntNesting by one.
   2) CoEnterISR() and CoExitISR() must be used in pairs.

6.1.4 CoExitISR()

Function Prototype:
   void   CoExitISR(void);
Descriptions:
   System exits the interrupts.
Returns:
   None
Returns:
   None

Example usage:

```
#include "CCRTOS.h"
void EXTI0_IRQHandler(void)
{
  CoEnterISR();          // Enter ISR

      …

  /* Process interrupt here */

      …

  CoExitISR();          // Exit ISR
}
```

Note:

1) When the system API functions are called in the ISR, you need to start a task scheduling by calling CoExitISR().
2) When the system exits the interrupts, decrease the interrupt nesting counter- OSIntNesting by one. When OSIntNesting reaches 0, start task scheduling.
3) CoExitISR() and CoEnterISR() must be used in pairs.

6.1.5 CoSchedLock()

Function Prototype:
   void   CoSchedLock(void);
Descriptions:
   Lock the scheduler.
Parameters:
   None
Returns:
   None
Example usage:

```
#include "CCRTOS.c"
void Task0 (void *pdata)
{
    .....
  CoSchedLock();

    …..

   /* Process critical resources  here */

    …..
  CoSchedUnlock();

    .....
}
```

Note:

1) Increase OSSchedLock by one. When the task scheduler is locked, the system ensure that the current task can't be preempted by any other

tasks so as to protect the critical resources.

2) CoSchedUnlock() and CoSchedUnlock() must be used in pairs.

6.1.6 CoSchedUnlock()

Function Prototype:
    void   CoSchedUnlock(void);
Descriptions:
    Unlock the scheduler.
Parameters:
    None
Returns:
    None
Example usage:

```
#include "CCRTOS.c"
void Task0 (void *pdata)
{
    .....
  CoSchedLock();

    .....

   /* Process critical resources  here */

    .....
  CoSchedUnlock();

    .....
}
```

Note:

1) Decrease OSSchedLock by one. When it reaches 0, start task scheduling.

2) CoSchedUnlock() and CoSchedUnlock() must be used in pairs.

6.1.7 CoGetOSVersion()

Function Prototype:
    OS_VER   CoGetOSVersion(void);
Descriptions:
    Obtain CooCox CoOS's version.
Parameters:
    None
Returns:
    CoOS's version

Example usage:

```
#include "CCRTOS.H"
void TaskN (void *pdata)
{
    U16 version;
    U8 Major, Minor;
    ....
    version = CoGetOSVersion();
    // Get Major Version
    Major = ((version>>12)&0xF) * 10 + (version>>8)&0xF;
    // Get Minor Version
    Minor = ((version>>4)&0xF) * 10 + version&0xF;
    printf("Current OS Version: %d.%02d\n",Major, Minor);
    ....
}
```

Note:

CoGetOsVersion() returns a 16-bit binary number. Shift it 8-bit to right and you can get the actual version. In other words, version 1.01 would be returned as 0x0101.

6.2 Task Management

6.2.1 CoCreateTask() & CoCreateTaskEx()

CoCreateTask()

Function Prototype:

OS_TID CoCreateTask

(
FUNCPtr   task,
void*   argv,
U8   prio,
OS_STK*   stk,
U16   stkSz,
);

Description:

Create a task and return its ID.

Parameters:

[IN] task

The function which create the task

[IN] argv

The parameter list of the function

[IN] prio

The priority of the task

[IN] stk

The starting address of the task stack

[IN] stkSz

The size of the task stack(its unit is word)

Returns:

Task ID,   Create successfully.

-1,   Failed to create.

Example Usage:

```
#include "CCRTOS.h"
#define TASKM_PRIO        11
#define TASKM_STK_SIZE      100
OS_STK TaskMStk[TASKM_STK_SIZE];
OS_TID TaskMId;
void TaskM (void *pdata);
void TaskN (void *pdata)
{
    ...
    TaskMId = CoCreateTask (TaskM,
                            (void *)0,
                            TASKM_PRIO,
                            &TaskMStk[TASKM_STK_SIZE-1],
                            TASKM_STK_SIZE);
    if (TaskMID==E_CREATE_FAIL)
    {
        printf("Task Create Failed !\n");
    }
    else
    {
        printf("Task ID : %d\n",TaskMId);
    }
    ...
}
void TaskM (void *pdata)
{
    ...
}
```

Note:
1) A relevant PCB is distributed when a task is created.
2) A task is in the ready state once being successfully created.
3) If the priority of the current created task is higher than the running task, the system start a task scheduling and distribute the execution time to the current task.
4) The maximum stkSz is 0xfff.

CoCreateTaskEx()

Function Prototype:

OS_TID CoCreateTaskEx

(
FUNCPtr   task,
void*   argv,
U8   prio,
OS_STK*   stk,
U16   stkSz,
U16   timeSlice,
BOOL   isWaitting
);

Description:

Create a task and return its ID.

Parameters:

[IN] task

The function which create the task

[IN] argv

The parameter list of the function

[IN] prio

The priority of the task

[IN] stk

The starting address of the task stack

[IN] stkSz

The size of the task stack(its unit is word)

[IN] timeSlice

The time slice that the task runs (Once 0 is transferred, set it to the default length of the system.)

[IN] isWaitting

Task's initial state when being created. If TRUE, task is in TASK_WAITING.

Returns:

Task ID, Create successfully.
-1,   Failed to create.

Example Usage:

```c
#include "CCRTOS.h"
#define TASKM_PRIO        11
#define TASKM_STK_SIZE      100
#define TASKM_TIME_SLICE    10
OS_STK TaskMStk[TASKMSTKSIZE];
OS_TID TaskMId;
void TaskM (void *pdata);
void TaskN (void *pdata)
{
    ...
    TaskMId = CoCreateTaskEx(TaskM,
                             (void *)0,
                             TASKM_PRIO,
                             &TaskMStk[TASKM_STK_SIZE-1],
                             TASKM_STK_SIZE,
                             TASKM_TIME_SLICE,
                             FASLE);
     if (TaskMID==E_CREATE_FAIL)
    {
        printf("Task Create Failed !\n");
     }
    else
    {
        printf("Task ID %d\n",TaskMId);
    }
    ...
}
void TaskM (void *pdata)
{
    ...
}
```

Note:

1) A relevant PCB is distributed when a task is created.
2) A task is in the ready state once being successfully created.
3) If the priority of the current created task is higher than the running task, the system start a task scheduling and distribute the execution time to the current task.

4) The maximum stkSz is 0xfff, the maximum timeSlice is 0xfff.

6.2.2 CoExitTask()

Function Prototype:
    void   CoExitTask(void);
Description:
    Exit current task.
Parameters:
    None
Returns:
    None
Example Usage:

```
#include "CCRTOS.h"
void TaskN (void *pdata)
{
    ...
    CoExitTask();   // Exit current task
}
```

Note:
1) When the task exits, If the system is in Dynamic task scheduling mode, it will recover its TCB resources automatically so as to redistribute them. Else, holding TCB resources and waiting to be actived.
2) When a task exits, the task scheduler runs and assign the system runtime to the next task.
3) This function can be called only inside the task.

6.2.3 CoDelTask()

Function Prototype:
    StatusType CoDelTask
                    (
                     OS_TID task ID
                    );
Description:
    Delete a task.
Parameters:
    [in] taskID
        ID of the task to be deleted
Returns:
    E_INVALID_ID,   Invalid task ID.
    E_PROTECTED_TASK,   Protected by the system and can not be deleted.
    E_OK,   Delete successfully.

Example Usage:

```c
#include "CCRTOS.h"
OS_TID TaskMId;
void TaskM (void *pdata);
void TaskN (void *pdata)
{
    StatusType result;
    ...
    result = CoDelTask(TaskMId);
    if (result != E_OK)
    {
        if (result==E_INVALID_ID)
        {
            printf("Invalid task ID !\n");
        }
        else if (result==E_PROTECTED_TASK)
        {
            printf("Protected task in OS cannot be deleted !\n");
        }
    }
    ...
}
void TaskM (void *pdata)
{
    ...
}
```

Note:

When the task exits, If the system is in Dynamic task scheduling mode, it will recover its TCB resources automatically so as to redistribute them. Else, holding TCB resources and waiting to be actived.

6.2.4 CoGetCurTaskID()

Function Prototype:

CoGetCurTaskID(void);

Description:

Obtain the ID of current task.

Parameters:

None

Returns:

ID of current task

Example Usage:

```
#include "CCRTOS.h"
void TaskN (void *pdata)
{
    OS_TID tid;
    ...
    tid = CoGetCurTaskID();
    printf("Current task ID is %d !\n",tid);
    ...
}
```

Note:

None

6.2.5 CoSetPriority()

Function Prototype:

StatusType   CoSetPriority

(

OS_TID   taskID,

U8   priority

);

Description:

Set the priority of the designated task and return the executing state of the function.

Parameters:

[in] task ID

ID of the designated task

[in] Priority

The priority to which the task will be set

Returns:

E_INVALID_ID,   The task ID is invalid.

E_OK,   Reset successfully.

Example Usage:

```
#include "CCRTOS"
#define NEW_PRIO        10
void TaskN (void *pdata)
{
    ...
  SetPriority (TaskMId, NEW_PRIO);
    ...
}
void TaskM (void *pdata)
{
    ...
}
```

Note:

1) If the task which to be set is in the waiting list and the priority of this task being reset to is higher than the current running one, run a task scheduling.

2) The changing of the priority will influence the order of the lists that relevant to this priority, such as the mutex list, the event waiting list and so on.

6.2.6 CoSuspendTask()

Function Prototype:
StatusType CoSuspendTask
                        (
                        OS_TID task ID
                        );

Description:
Suspend the designated task.

Parameters:
[in] task ID
     ID of the designated task

Returns:

E_INVALID_ID,	The task ID is invalid.
E_ALREADY_IN_WAITING,	The designated task has been in the waiting state.
E_PROTECTED_TASK,	It couldn't suspend idle task.
E_OK,	Suspend the task successfully.

Example Usage:

```
#include "CCRTOS.h"
void TaskN (void *pdata)
{
  StatusType result;
  result= SuspendTask (TaskMId);   // Suspend TaskM
  if (result!= E_OK)
  {
     if (result==E_INVALID_ID)
     {
         printf ("TaskM does not exist !\n");
     }
     else if (result==E_ALREADY_IN_WAITING)
     {
         printf ("TaskM is not ready !\n");
     }
  }
}
void TaskM (void *pdata)
{
    ...
}
```

Note:

1) The task being suspended turns to the waiting state (TASK_WAITING).
2) CoSuspendTask() and CoAwakeTask() must be used in pairs.

6.2.7 CoAwakeTask()

Function Prototype:
   StatusType CoAwakeTask
                         (
                          OS_TID task ID
                         );

Description:
   Awake the designated task.

Parameters:
   [in] task ID
       ID of the designated task

Returns:
   E_INVALID_ID,   The task ID is invalid.
   E_TASK_WAIT_OTHER,   The task now is waiting other awake event.
   E_TASK_NOT_WAITING,   The task is not in waiting state.

E_PROTECTED_TASK,   Idle task mustn't be awaked.

E_OK,   Awake the task successfully.

Example Usage:

```
#include "CCRTOS.h"
void TaskI (void *pdata)
{
    ...
  SuspendTask (TaskMId);        // Suspend TaskM
    ...
}
void TaskN (void *pdata)
{
  StatusType result;
    ...
  result= AwakeTask (TaskMId);      // Wakeup TaskM
  if (result==E_OK) printf("TaskM is awake!\n");
    ...
}
void TaskM (void *pdata)
{
    ...
}
```

Note:

1) The task will remain in the waiting state (TASK_WAITING) if the task is still waiting for other events. Otherwise, the task will return to the ready (TASK_READY) state.

2) CoSuspendTask() and CoAwakeTask() must be used in pairs.

6.2.8 **CoActivateTask**()

Function Prototype:

StatusType CoActivateTask

(

OS_TID task ID,

void *argv

);

Description:

Awake the designated task.

Parameters:

[in] task ID

ID of the designated task

[in] argv

The parameter list of the function

Returns:

    E_INVALID_ID,   The task ID is invalid.

    E_OK,   Awake the task successfully.

Example Usage:

```c
#include "CCRTOS.h"
void TaskI (void *pdata)
{
    ...
  CoDelTask(TaskMId);             // Delete TaskM
    ...
}
void TaskN (void *pdata)
{
  StatusType sta;
    ...
  sta = CoActivateTask(TaskMId,NULL);     // ActivateTaskM
  if(sta==E_OK) printf("TaskM is ready !\n");
    ...
}
void TaskM (void *pdata)
{
    ...
}
```

Note:

    None

6.3 Time Management

6.3.1 CoGetOSTime()

Function Prototype:
   U64   CoGetOSTime(void);
Description:
   Obtain the current system time.
Parameters:
   None
Returns:
   The current system tick number.
Example Usage:

```
#include "CCRTOS.h"
void TaskN (void *pdata)
{
  U64 ostime;
     ...
  ostime = CoGetOSTime();
     ...
}
```

Note:
    None

6.3.2 CoTickDelay()

Function Prototype:
   StatusType   CoTickDelay
                        (
                         U32 ticks
                        );
Description:
   Delay the task for a specified system tick number.
Parameters:
   [IN] ticks
         The number of the system tick number to be delayed
Returns:
   E_CALL,   Called in the ISR.
   E_OK,   Execute correctly.

Example Usage:

```
#include "CCRTOS.h"
void TaskN (void *pdata)
{
  StatusType result;
    ...
  result= CoTickDelay(15);          // Delay 15 system ticks
  if(result != E_OK)
  {
     if(result == E_CALL)
     {
         printf("TickDelay cannot been called in ISR !\n");
     }
  }
    ...
}
```

Note:

Once calling TickDelay(), the current task will change from the running state(TASK_RUNNING)to the waiting state (TASK_WAITING). Then it will be inserted into the delay list and delay for the specified system ticks.

6.3.3 CoResetTaskDelayTick()

Function Prototype:

StatusType   CoResetTaskDelayTick
                              (
                               OS_TID   taskID,
                               U32   ticks
                              );

Description:

Reset the delayed system ticks of the designated task.

Parameters:

[IN] taskID

ID of the designated task

[IN] ticks

The delayed system ticks to be reset to

Returns:

E_INVALID_ID,   ID is invalid.

E_NOT_IN_DELAY_LIST,   The designated task isn't in the delay list.

E_OK,   Execute correctly.

Example Usage:

```
#include "CCRTOS.h"
OS_TID TaskMId;
OS_TID TaskNId;
void TaskM (void *pdata)
{
    …
    TickDelay (30);
    …
}
void TaskN (void *pdata)
{
    StatusType result;
    …
    /* Reset TaskM delay time */
    result = ResetTaskDelayTick (TaskMId, 61);
    if(result!=E_OK)
    {
        if(result == E_INVALID_ID)
        {
            printf("Invalid task id !\n");
        }
        else if (result == E_NOT_IN_DELAY_LIST)
        {
            printf("TaskM is not in delay list !\n");
        }
    }
    …
}
```

Note:

If the tick number is reset to 0, move the specified task out of the delay list.

6.3.4 CoTimeDelay()

Function Prototype:

```
StatusType   CoTimeDelay
                    (
                     U8      hour,
                     U8      minute,
                     U8      sec,
                     U16      millsec
                    );
```

Description:

Delay a task for a specified time.

Parameters:

[IN] Hour

The number of hours to be delayed

[IN] Minute

The number of minutes to be delayed

[IN] Sec

The number of seconds to be delayed

[IN] Millsec

The number of milliseconds to be delayed

Returns:

E_CALL,	Called in the ISR.
E_INVALID_PARAMETER,	Parameter passed is invalid.
E_OK,	Execute correctly.

Example Usage:

```c
#include "CCRTOS.h"
void TaskN (void *pdata)
{
  StatusType result;
    ...
  result= TimeDelay(0,0,1,0);        // Delay 1 second
  if(result!=E_OK)
  {
    if(result==E_CALL)
    {
        printf("TimeDelaycannot been called in ISR !\n");
    }
    else if (result== E_INVALID_PARAMETER)
    {
        printf("Invalid parameter !\n");
    }
  }
    ...
}
```

Note:

If the time input is not up to snuff, return errors.

6.4 Software Timer

6.4.1 CoCreateTmr()

Function Prototype:

    OS_TCID   CoCreateTmr

                        (

                        U8   tmrType,

                        U32   tmrCnt,

                        U32   tmrReload,

                        vFUNCPtr   func

                        );

Description:

    Create a timer.

Parameters:

    [IN] Tmrtype

        The type of the timer

    [IN] tmrCnt

        The initial counter value (system ticks)

    [IN] tmrReload,

        The reloaded value of the timer

    [IN] func

        The call-back function of the timer

Returns:

    The ID of the timer,   Create successfully.

    -1,   Failed to create.

Example Usage:

```
#include "CCRTOS.h"
OS_TCID sftmr;
void SftTmrCallBack(void)
{
    ...
}
void TaskN (void *pdata)
{
    ...
  sftmr = CoCreateTmr(TMR_TYPE_PERIODIC,
                      100,
                      100,
                      SftTmrCallBack);
  if (sftmr == E_CREATE_FAIL)
  {
    printf("Failed to create the timer!\n");
  }
  else
  {
    printf("Create the timer successfully, Time ID is %d\n", sftmr);
  }
    ...
}
```

Note:

1) CooCox CoOS provides two kinds of timers: the periodic timer and the disposable timer. For the periodic timer, the time first-calling the callback function is determined by the tmrCnt, and then it is determined by the tmrReload. For the disposable timer, the calling time is determined completely by the tmrCnt and it can call only once.
2) Once a timer is created, it is in the stopping state. You need call CoStartTmr() to startup it.

6.4.2 CoStartTmr()

Function Prototype:

    StatusType   CoStartTmr
                        (
                         OS_TCID tmrID
                        );

Description:

    Start a specified timer to work normally.

Parameters:

[IN] tmrID

The ID of the specified timer

Returns:

E_INVALID_ID,   The timer ID passed is invalid.

E_OK,   Start the specified timer successfully.

Example Usage:

```
#include "CCRTOS.h"
OS_TCID sftmr;
void TaskN (void *pdata)
{
  StatusType result;
     …
  /* Create Software Timer */
  sftmr = CoCreateTmr(TMR_TYPE_PERIODIC,
                        100,
                        100,
                         SftTmrCallBack);
     …
  /* Start Software Timer */
  result= CoStartTmr (sftmr);
  if (result != E_OK)
  {
     if (result == E_INVALID_ID)
     {
         printf("The timer id passed is invalid, can't start the timer. \n");
     }
  }
     …
}
```

Note:

None

6.4.3 CoStopTmr()

Function Prototype:

StatusType CoStopTmr

(

OS_TCID tmrID

);

Description:

Stop a specified timer.

Parameters:

[IN] tmrID

The ID of the specified timer

Returns:

E_INVALID_ID,   The timer ID passed is invalid.

E_OK,   Stop the specified timer successfully.

Example Usage:

```c
#include "CCRTOS.h"
OS_TCID sftmr;
void TaskN (void *pdata)
{
  StatusType sta;
    ...
  /* Stop Software Timer */
  sta = CoStopTmr (sftmr);
  if (sta != E_OK)
  {
    if (sta == E_INVALID_ID)
    {
        printf("The timer id passed is invalid, failed to stop. \n");
    }
  }
    ...
}
```

Note:

When a software timer stops running, the system will retain its current counter value in order to re-activate it.

6.4.4 CoDelTmr()

Function Prototype:

StatusType CoDelTmr

(

 OS_TCID tmrID

);

Description:

Delete a specified timer and release the resources it had occupied.

Parameters:

[IN] tmrID

The ID of the specified timer

Returns:

E_INVALID_ID,   The timer ID passed is invalid.

E_OK,   Delete the specified timer successfully.

Example Usage:

```
#include "CCRTOS.h"
OS_TCID sftmr;
void TaskN (void *pdata)
{
  StatusType result;
     ...
  /* Create Software Timer */
  sftmr = CoCreateTmr(TMR_TYPE_PERIODIC,
                      100,
                      100,
                      SftTmrCallBack);
     ...
  /* Delete Software Timer */
  result= CoDelTmr (sftmr);
  if (result != E_OK)
  {
    if (result== E_INVALID_ID)
    {
        printf("The timer id passed is invalid, filed to delete!\n");
    }
  }
     ...
}
```

Note:
　　None

6.4.5 CoGetCurTmrCnt()

Function Prototype:
　　U32   CoGetCurTmrCnt
　　　　　　　　　(
　　　　　　　　　 OS_TCID   tmrID,
　　　　　　　　　 StatusType*   perr
　　　　　　　　　);
Description:
　　Obtain the current counter of a specified software timer.
Parameters:
　　[IN] tmrID
　　　　The ID of the specified timer
　　[OUT] Perr
　　　　　The types of the error returned
　　　　　E_INVALID_ID,   The timer ID passed is invalid.

E_OK,   Get the current counter of the timer successfully.

Returns:

The current counter of the specified software timer

Example Usage:

```
#include "CCRTOS.h"
OS_TCID sftmr;
void TaskN (void *pdata)
{
  StatusType result;
  U32    sftcnt;
     ...
  sftcnt = CoGetCurTimerCnt (sftmr, &result);
  if (result != E_OK)
  {
     if (result == E_INVALID_ID)
    {
         printf("The timer id passed is invalid, failed to get. \n");
    }
  }
  else
  {
    printf("Current Timer Counter : %ld", sftcnt);
  }
     ...
}
```

Note:

   None

6.4.6 CoSetTmrCnt()

Function Prototype:

StatusType CoSetTmrCnt

                    (
                     OS_TCID   tmrID,
                     U32   tmrCnt,
                     U32   tmrReload
                    );

Description:

Set the timer counter and reload value.

Parameters:

[IN] tmrID

     The ID of the specified timer

[IN] tmrCnt

     The count value to be reset to

[IN] tmrReload

　　The reloaded value of the timer to be reset to

Returns:

E_INVALID_ID,　The timer ID passed is invalid.

E_OK,　　　　Get the count value of the specified timer successfully.

Example Usage:

```
#include "CCRTOS.h"
OS_TCID sftmr;
void TaskN (void *pdata)
{
  StatusType result;
    ...
  result= CoSetTimerCnt (sftmr, 200, 200);
  if (result != E_OK)
  {
    if (result== E_INVALID_ID)
    {
        printf("The timer id passed is invalid, failed to stop. \n");
    }
  }
    ...
}
```

Note:

　None

6.5 Memory Management

6.5.1 CoKmalloc()

Function Prototype:
    void*   CoKmalloc
                    (
                       U32   size
                    );
Description:
    Allocate the size of memory block.
Parameters:
    [IN] size
        The length of the memory block which need to be allocated with the
        unit bytes
Returns:
    NULL,   Failed to allocate.
    Others,   Allocate successfully and return the initial address pointer of
                    the memory block.
Example usage:

```
#include "CCRTOS.h"
void TaskN (void *pdata)
{
    unsigned int *ptr;
    ...
    /* Allocate 20 words of memory block from kernel heap */
    ptr = (unsigned int *)CoKmalloc(20*4);

    /* process ptr here */
    ...
    /* Release memory block to kernel heap */
    CoKfree(ptr);
    ...
}
```

Note:
    Once you called CoKmalloc(), it will cost 8 bytes to manage this memory
    block.

6.5.2 CoKfree()

Function Prototype:

 void CoKfree

    (

     void* memBuf

    );

Description:

 Release the memory block with the initial address memBuf.

Parameters:

 [IN] memBuf

  The initial address of the memory block which needs to be released

Returns:

 None

Example usage:

```
#include "CCRTOS.h"
void TaskN (void *pdata)
{
    unsigned int *ptr;
    ...
    /* Allocate 20 words of memory block from kernel heap */
    ptr = (unsigned int *)CoKmalloc(20*4);

    /* process ptr here */
    ...
    /* Release memory block to kernel heap */
    CoKfree(ptr);
    ...
}
```

Note:

 MemBuf must be obtained by CoKmalloc(), or it will do nothing and return immediately.

6.5.3 CoCreateMemPartition()

Function Prototype:

 OS_MMID CoCreateMemPartition

        (

         U8* memBuf,

         U32 blockSize,

         U32 blockNum

        );

Description:

Create a memory partition with a certain length.

Parameters:

[IN] memBuf

The initial address of the partition

[IN] blockSize

The size of each memory block in the partition

[IN] blockNum

The number of the memory block in the partition

Returns:

The ID of the memory partition,   Create the partition successfully and
return the partition ID.

-1,   Failed to create.

Example usage:

```
#include "CCRTOS.h"
#define MEM_BLOCK_NUM   10
OS_MMID mmc;
unsigned int MemoryBlock[100];
void TaskN (void *pdata)
{
    ...
    /* Create a memory partition          */
    /* Memory size: 100*4 bytes,          */
    /* Block num:    10                    */
    /* Block size    100*4/10 = 40 bytes */
    mmc=CoCreateMemPartition((U8*)MemoryBlock,
                            100*4/MEM_BLOCK_NUM,
                            MEM_BLOCK_NUM);
    if (mmc == E_CREATE_FAIL)
    {
        printf("Create memory partition fail !\n");
    }
    else
    {
        printf("Memory Partition ID : %d \n", mmc);
    }
    ...
}
```

Note:

1) Once a memory partition is created successfully, the system will distribute a memory control block to manage the memory blocks.

2) blockSize cannot be 0 and blockNum must larger than 2.

6.5.4 CoDelMemoryPartition()

Function Prototype:
StatusType CoDelMemoryPartition
(
 OS_MMID   mmID
);
Description:
Delete a certain memory partition.
Parameters:
[IN] mmID
The ID of the memory partition
Returns:
E_INVALID_ID,   The ID of the memory partition is invalid.
E_OK,   Delete successfully.
Example usage:

```
#include "CCRTOS.h"
OS_MMID mmc;
void TaskN (void *pdata)
{
    StatusType result;
    ...
    /* Delete a memory partition   */
    /* mmc: Created by other task */
    result = CoDelMemoryPartition(mmc);
    if (result != E_OK)
    {
        if (result == E_INVALID_ID)
        {
            printf("Invalid memory partition !\n");
        }
    }
    ...
}
```

Note:
1) Once deleting a memory partition, the resources of the memory control block which is occupied by this partition will be released.
2) When deleting a memory partition, the system will not check whether it is idle, you can control it in your own applications.

6.5.5 CoGetMemoryBuffer()

Function Prototype:
    void*   CoGetMemoryBuffer
                        (
                         OS_MMID   mmID
                        );

Description:
    Obtain a memory block from a certain memory partition.

Parameters:
    [IN] mmID
        The ID of the memory partition

Returns:
    NULL,   Failed to allocate.
    Others,   Allocate successfully and return the initial address pointer of the
                memory block.

Example usage:

```
#include "CCRTOS.h"
OS_MMID mmc;
void TaskN (void *pdata)
{
    int *ptr;
    ...
    /* Get a memory block from memory partition */
    /* mmc: Created by other task */
    ptr = (int* )CoGetMemoryBuffer(mmc);
    if (ptr == NULL)
    {
        printf("Assign buffer fail !\n");
    }
    else
    {
        ...
        /* Process assigned buffer here */
        ...
        /* Free assigned buffer to memory partition */
        CoFreeMemoryBuffer(mmc, (void* )ptr);
    }
    ...
}
```

Note:
    None

6.5.6 CoFreeMemoryBuffer()

Function Prototype:
StatusType   CoFreeMemoryBuffer
                                                  (
                                                  OS_MMID   mmID,
                                                  void*   buf
                                                  );

Description:
Release the memory block which has the initial address buf to a certain memory partition.

Parameters:
[IN] mmID
        The ID of the memory partition
[IN] buf
        The initial address of the memory block which needs to be released

Returns:
E_INVALID_ID,	The ID of the memory partition is invalid.
E_INVALID_PARAMETER,	Invalid parameter buf.
E_OK,	Release the memory block successfully.

Example usage:

```
#include "CCRTOS.h"
OS_MMID mmc;
void TaskN (void *pdata)
{
    int *ptr;

    ...
    /* Get a memory block from memory partition */
    /* mmc: Created by other task */
    ptr = (int* )CoGetMemoryBuffer(mmc);
    if (ptr == NULL)
    {
        printf("Assign buffer fail !\n");
    }
    else
    {
        ...
        /* Process assigned buffer here */
        ...
        /* Free assigned buffer to memory partition */
        CoFreeMemoryBuffer(mmc, (void* )ptr);
    }
    ...
}
```

Note:

MemBuf must be obtained by CoKmalloc(), or it will error return.

6.5.7 CoGetFreeBlockNum()

Function Prototype:

U32 CoGetFreeBlockNum

(
OS_MMID   mmID,
StatusType*   perr
)

Description:

Obtain the number of the free block in a certain memory partition.

Parameters:

[IN]   mmID

The ID of the memory partition

[OUT] perr

Errors returned:

E_INVALID_ID,   The ID is invalid.

E_OK,   Obtain the number successfully.

Returns:

fbNum,   The number of the free block in the memory partition

Example usage:

```
#include "CCRTOS.h"
OS_MMID mmc;
void TaskN (void *pdata)
{
    U32 blocknum;
    StatusType result;
    ...
    /* Get free blocks' number */
    /* mmc: Created by other task */
    blocknum = CoGetFreeBlockNum(mmc, &result);
    if (result != E_OK)
    {
        if (result == E_INVALID_ID) {
            printf("Invalid ID !\n");
        }
    }
    ...
}
```

Note:

None

6.6 Mutex Section

6.6.1 CoCreateMutex()

Function Prototype:
   OS_MutexID   CoCreateMutex(void);
Descriptions:
   Create a Mutex section.
Parameters:
   None
Returns:

   Mutex section ID,   create successfully

   -1,   create failed
Example usage:

```
#include "CCRTOS.h"
OS_MutexID mutex;
void TaskN (void *pdata)
{
    …
  /* Create a mutex */
  mutex = CoCreateMutex ();
  if (mutex == E_CREATE_FAIL)
  {
     printf("Create mutex fail. \n");
  }
  else
  {
    printf("Mutex ID : %d \n", mutex);
  }
    …
}
```

Note:
   None

6.6.2 CoEnterMutexSection()

Function Prototype:
   StatusType CoEnterMutexSection(

                              OS_MutexID   mutexID
                              );
Descriptions:
   Enter the mutex section whose ID was designated.

Parameters:

    [in] muterID

        the designated mutex section ID

Returns:

    E_CALL,   be called in the interrupt service program

    E_INVALID_ID,   the mutex section ID that was incomed is invalid

    E_OK,   enter the mutex section successfully

Example usage:

```
#include "CCRTOS.h"
OS_MutexID mutex;
void TaskN (void *pdata)
{
    ...
   /* Create a mutex */
  mutex = CoCreateMutex ();
   /* Enter critical region */
  CoEnterMutexSection(mutex);

    …
   /*Process here */

    …
   /* Exit critical region */
  CoLeaveMutexSection(mutex);

    ...
}
```

Note:

    1) If high-priority task A attempts to enter the mutex region,the system will promote the priority of task B which has entered the mutex region currently to the same level of task A, and change task A' state to the waiting state (TASK_WAITING),then do a task scheduling so that the task B which has entered the mutex region could leave the mutex region as soon as possible. Task B' s priority will be back to the original priority.

    2)This function can not be used in interrupt service routine.

    3)It should be used in pairs with CoLeaveMutexSection().

6.6.3 CoLeaveMutexSection()

Function Prototype:

    StatusType   CoLeaveMutexSection(

                              OS_MutexID mutexID

                              );

Descriptions:

    Leave the mutex section whose ID was designated.

Parameters:

[in] muterID

the designated mutex section ID

Returns:

E_CALL,　　　　　be called in the interrupt service program

E_INVALID_ID,　the mutex section ID that was incomed is invalid

E_OK,　　　　　enter the mutex section successfully

Example usage:

```
#include "CCRTOS.h"
OS_MutexID mutex;
void TaskN (void *pdata)
{
     ...
    /* Create a mutex */
  mutex = CoCreateMutex ();
    /* Enter critical region */
  CoEnterMutexSection(mutex);

    …
    /*Process here */
    …
    /* Exit critical region */
  CoLeaveMutexSection(mutex);

    ...
}
```

Note:

1) If there are tasks waiting to enter the mutex section currently,it need to run a task scheduling　and the priority of the currently task will be back to the original priority when leaving the mutex section.
2) This function can not be used in interrupt service routine.
3) It should be used in pairs with CoEnterMutexSection().

6.7 Semaphores

6.7.1 CoCreateSem()

Function Prototype:
    OS_EventID   CoCreateSem(
                                U16 initCnt,
                                U16 maxCnt,
                                U8 sortType
                                );
Descriptions:
    Create a semaphore.
Parameters:
     [in] initCnt

        the initial volume of the effective number of semaphores
    [in] maxCnt
        The maximum value of the semaphore
    [in] sortType
        arrangement types:
            EVENT_SORT_TYPE_FIFO,   FIFO order
            EVENT_SORT_TYPE_PRIO,   Preemptive Priority order
Returns:
    Semaphore ID,   create successfully
    -1,   create failed
Example usage:

```
#include "CCRTOS.h"
OS_EventID semaphore;
void TaskN (void *pdata)
{
     ...
   /* Create a semaphore */
  semaphore = CoCreateSem (1, 10, EVENT_SORT_TYPE_FIFO);
  if (semaphore == E_CREATE_FAIL)
  {
    printf("Create semaphore failed !\n");
  }
  else
  {
    printf("Semaphore ID : %d \n", semaphore);
  }
    ...
}
```

Note:

None

6.7.2 CoDelSem()

Function Prototype:

StatusType CoDelSem(

                     OS_EventID   id,

                     U8   opt

                     );

Descriptions:

Delete the semaphore whose ID was designated.

Parameters:

[in] id

    the designated semaphore ID

[in] opt

    the ways to delete the designated semaphore：

        EVENT_DEL_NO_PEND，delete when the waiting list is

                       empty

        EVENT_DEL_ANYWAY，delete unconditionally

Returns:

E_INVALID_ID，   the semaphore ID that was incomed is invalid

E_INVALID_PARAMETER，invalid parameters, that is ,the

                     corresponding control block is empty

E_TASK_WAITING，   the waiting list is not empty

E_OK，   delete successfully

Example usage:

```c
#include "CCRTOS.h"
OS_EventID semaphore;
void TaskN (void *pdata)
{
  StatusType result;
   ...
    /* Create a semaphore */
  result = CoDeleteSem (semaphore, OPT_DEL_ANYWAY);
  if (result != E_OK)
  {
    if (result == E_INVALID_ID)
    {
      printf("Invalid event ID !\n");
    }
  }
    ...
}
```

Note:

Release the resources that the semaphore occupied If delete it successfully,then run a task sheduling and give the system execution time to the highest priority task.

6.7.3 CoAcceptSem()

Function Prototype:

StatusType   CoAcceptSem(

OS_EventID   id

);

Descriptions:

To obtain semaphore resources whose ID was designated.

Parameters:

[in] id

the designated semaphore ID

Returns:

E_INVALID_ID,   the semaphore ID that was incomed is invalid

E_SEM_EMPTY,   the semaphore resources whose ID was designated is empty

E_OK,   to obtain the resources successfully

Example usage:

```
#include "CCRTOS.h"
OS_EventID semaphore;;
void TaskN (void *pdata)
{
  StatusType result;
     ...
    /* Accept a semaphore without waitting */
  result = CoAcceptSem (semaphore);
  if (result != E_OK)
  {
     if (result == E_INVALID_ID)
     {
         printf("Invalid event ID !\n");
     }
     else if (result == E_SEM_EMPTY)
     {
         printf("No semaphore exist !\n");
     }
  }
  else
  {
     …
     /* Process semaphore here */
     …
  }
    ...
}
```

Note:

1) Running AcceptSem() program is an incomplete P operation. There are some difference between them,when the number of the resources is 0,complete P operation will wait until other task release the corresponding resource or finished for timeout,but this program will return error immediately.

2) The number of the semaphore resources substract 1 after getting the resources successfully.

6.7.4 CoPendSem()

Function Prototype:

```
StatusType   CoPendSem(
                        OS_EventID   id,
                        U32          timeout
                        );
```

Descriptions:

To wait the semaphore.

Parameters:

[in] id

the designated semaphore ID

[in] timeout

overtime time,0 means waiting indefinitely

Returns:

E_CALL,   be called in the interrupt service

program

E_INVALID_ID,   the semaphore ID that was

incomed is invalid

E_TIMEOUT,   time is out for waiting resources

E_OK,   to obtain the resources successfully

Example usage:

```
#include "CCRTOS.h"
OS_EventID semaphore;;
void TaskN (void *pdata)
{
  StatusType result;
     ...
     /* Pend a semaphore, Time out: 20 */
  result = CoPendSem (semaphore, 20);
  if (result != E_OK)
  {
     if (result == E_INVALID_ID)
     {
        printf("Invalid event ID !\n");
     }
     else if (result == E_CALL)
     {
        printf("Error call in ISR !\n");
     }
     else if (result == E_TIMEOUT)
     {
        printf("Semaphore was not received within the specified timeout time !\n");
     }
  }
  else
  {
     ...
     /* Process semaphore here */
     ...
  }
     ...
}
```

Note:

    1)This function runs a task sheduling while waiting and give the system execution time to the highest priority task.

    2)The number of the semaphore resources substract 1 after getting the resources successfully.

    3)The function cann't be used in the interrupt service program.

6.7.5 CoPostSem()

Function Prototype:
    StatusType CoPostSem(
                              OS_EventID   id
                              );
Descriptions:
    To release semaphore resources whose ID was designated.
Parameters:
    [in] id
        the designated semaphore ID
Returns:
    E_INVALID_ID,   the semaphore ID that was
                            incomed is invalid
    E_SEM_FULL,   the designated semaphore ID has
                            been reached it maxmum value
    E_OK,   release a semaphore resource successfully
Example usage:

```
#include "CCRTOS.h"
OS_EventID semaphore;
void TaskN (void *pdata)
{
  StatusType result;
      ...
    /* Post a semaphore */
  result = CoPostSem (semaphore);
  if (result != E_OK)
  {
     if (result == E_INVALID_ID)
     {
         printf("Invalid event ID !\n");
     }
     else if (result == E_SEM_FULL)
     {
         printf("Semaphore is full !\n");
     }
  }
    ...
}
```

Note:
    1)The function cann't be used in the interrupt service program.
    2)It is used in pairs with CoAcceptSem() or CoPendSem().

6.7.6 isr_PostSem()

Function Prototype:
StatusType isr_PostSem(OS_EventID id)
Descriptions:
To release semaphore resources whose ID was designated in the interrupt service program.
Parameters:
[IN] id
the designated semaphore ID
Returns:

E_SEV_REQ_FULL	interrupt service requests is full
E_INVALID_ID	the semaphore ID that was incomed is invalid
E_SEM_FULL	the designated semaphore ID has been reached it maximal value
E_OK	release a semaphore resource successfully

Example usage:

```
#include "CCRTOS.h"
OS_EventID semaphore;
void XXX_IRQHandler(void)
{
    StatusType result;
    EnterISR();        // Enter ISR
        ...
    /* Post a semaphore */
    result = isr_PostSem (seamaphore);
    if (result != E_OK) {
        if (result == E_SEV_REQ_FULL) {
            printf("Service requst queue is full !\n");
        }
    }
        ...
    ExitISR();       // Exit ISR
}
```

Note:
Interrupt service routine can not call CoPostSem () to release the semaphore resources ,otherwise the system will lead to confusion.

6.8 Mailboxes

6.8.1 CoCreateMbox()

Function Prototype:
    OS_EventID CoCreateMbox(
                                    U8 sortType
                                    );

Descriptions:
    Create a mailbox.

Parameters:
    [in] sortType:
        arrangement types:
            EVENT_SORT_TYPE_FIFO,   FIFO order
            EVENT_SORT_TYPE_PRIO,   Preemptive Priority order

Returns:
    Mailbox ID,   create successfully
    -1,   create failed

Example usage:

```
#include "CCRTOS.h"
OS_EventID mailbox;
void TaskN (void *pdata)
{
     ...
    /* Create a mailbox */
  mailbox = CoCreateMbox (EVENT_SORT_TYPE_FIFO);
  if (mailbox != E_OK)
  {
     if (mailbox == E_CREATE_FAIL)
    {
        printf("Create mailbox failed !\n");
    }
  }
  else
  {
     /* Process mail here */
     printf("MailBox ID : %d \n", mailbox);
  }
    ...
}
```

Note:
    None

6.8.2 CoDelMbox()

Function Prototype:
    StatusType CoDelMbox(
                                OS_EventID   id,
                                U8   opt
                                );
Descriptions:
    Delete the mailbox whose ID was designated.
Parameters:
    [in] id
        the designated mailbox ID
    [in] opt
        the ways to delete the designated mailbox :
            EVENT_DEL_NO_PEND, delete when the waiting list is
                                    empty
            EVENT_DEL_ANYWAY, delete unconditionally
Returns:
    E_INVALID_ID,   the mailbox ID that was incomed is invalid
    E_INVALID_PARAMETER, invalid parameters, that is ,the
                                corresponding control block is empty
    E_TASK_WAITING,   the waiting list is not empty
    E_OK,   delete successfully
Example usage:

```
#include "CCRTOS.h"
OS_EventID mailbox;
void TaskN (void *pdata)
{
  StatusType result;
      ...
    /* Create a mailbox */
  result = CoDelMbox (mailbox, OPT_DEL_ANYWAY);
  if (result != E_OK)
  {
     if (result == E_INVALID_ID)
    {
        printf("Invalid event ID !\n");
    }
  }
    ...
}
```

Note:

Release the resources that the mailbox occupied if delete it successfully,then run a task sheduling and give the system execution time to the highest priority task.

6.8.3 CoAcceptMail()

Function Prototype:

void*   CoAcceptMail(

                    OS_EventID   id,

                    StatusType*   perr

                    );

Descriptions:

To obtain mailbox message whose ID was designated.

Parameters:

[in] id

    the designated mailbox ID

[in] perr

    error returned types:

        E_INVALID_ID,   the mailbox ID that was

                    incomed is invalid

        E_MBOX_EMPTY,   the mailbox message whose ID

                    was designated is empty

        E_OK,   to obtain the message successfully

Returns:

To get the pointer of the mailbox message.

Example usage:

```
#include "CCRTOS.h"
OS_EventID mailbox;
void TaskN (void *pdata)
{
  StatusType result;
  void *msg;
      …
    /* Create a mailbox */
  msg = CoAcceptMail (mailbox, &result);
  if (result != E_OK)
  {
     if (result == E_INVALID_ID)
     {
         printf("Invalid event ID !\n");
     }
  }
  else
  {
       …
       /* Process mail here */
       …
  }
    …
}
```

Note:

If the message pointer of the designated mailbox is not empty while calling AcceptSem(),this function will get the message pointer and empty the mailbox;otherwise,return error immediately.

6.8.4 CoPendMail()

Function Prototype:

```
void*   CoPendMail(
                OS_EventID    id,
                U32           timeout,
                StatusType*    perr
                );
```

Descriptions:

To wait the message of the designated mailbox.

Parameters:

[in]   id

the designated mailbox ID

[in] timeout

overtime time,0 means waiting indefinitely

[out] perr

error returned types:

E_CALL, be called in the interrupt service program

E_INVALID_ID, the mailbox ID that was incomed is invalid

E_TIMEOUT, time is out for waiting resources

E_OK, to obtain the resources successfully

Returns:

To get the pointer of the mailbox message.

Example usage:

```c
#include "CCRTOS.h"
OS_EventID mailbox;
void TaskN (void *pdata)
{
  StatusType result;
  void *msg;
      ...
    /* Create a mailbox */
  msg = CoPendMail (mailbox, 20, &result);
  if (result != E_OK)
  {
      if (result == E_INVALID_ID)
    {
          printf("Invalid event ID !\n");
    }
  }
  else
  {
      /* Process mail here */
  }
    ...
}
void TaskM (void *pdata)
{
  StatusType result;
  unsigned int pridat;
    ...
  pridat = 0x49;
    ...
  result = CoPostMail (mailbox, &pridat);
  if (result != E_OK)
  {
      if (result == E_INVALID_ID)
    {
          printf("Invalid event ID !\n");
    }
  }
    ....
}
```

Note:

1)This function runs a task sheduling while waiting and give the system execution time to the highest priority task.

2) It will empty the mailbox after getting the mailbox message successfully.

3) The function cann't be used in the interrupt service program.

6.8.5 CoPostMail()

Function Prototype:

StatusType   CoPostMail(

OS_EventID   id,

void*   pmail

);

Descriptions:

Fill the message to the mailbox whose ID was   designated.

Parameters:

[in] id

the designated mailbox ID

[in] pmail

the message pointer that is filling

Returns:

E_INVALID_ID,   the mailbox ID that was incomed

is invalid

E_MBOX_FULL,   the mailbox has been full

E_OK,   fill message to mailbox successfully

Example usage:

```c
#include "CCRTOS.h"
OS_EventID mailbox;
void TaskN (void *pdata)
{
  StatusType result;
  void *msg;
     ...
    /* Create a mailbox */
  msg = CoPendMail (mailbox, 20, &result);
  if (result != E_OK)
  {
     if (result == E_INVALID_ID)
    {
        printf("Invalid event ID !\n");
    }
  }
  else
  {
    /* Process mail here */
  }
    ...
}
void TaskM (void *pdata)
{
  StatusType result;
  unsigned int pridat;
     ...
  pridat = 0x49;
     ...
  result = CoPostMail (mailbox, &pridat);
  if (result != E_OK)
  {
     if (result == E_INVALID_ID)
    {
        printf("Invalid event ID !\n");
    }
  }
    ....
}
```

Note:

1) The function cann't be used in the interrupt service program.

2) It is used in pairs with CoAcceptMail() or CoPendMail().

6.8.6 isr_PostMail()

Function Prototype:
StatusType isr_PostMail(

OS_EventID   id,

void*   pmail

);

Descriptions:
To send message to a mailbox whose ID was designated in the interrupt service program.

Parameters:
[IN] id

the designated mailbox ID

[IN] pmail

message pointer

Returns:
E_SEV_REQ_FULL,   interrupt service requests is full

E_INVALID_ID,   the semaphore ID that was incomed is invalid

E_MBOX_FULL,   the mailbox has been full

E_OK,   send message to mailbox successfully

Example usage:

```
#include "CCRTOS.h"
OS_EventID mailbox;
int IsrDat;
void XXX_IRQHandler(void)
{
    StatusType result;
    EnterISR();        // Enter ISR
        ...
    IsrDat = 0x90;
    /* Post a mail to Mailbox that created by other tase */
    result = isr_PostMail (mailbox, &IsrDat);
    if (result != E_OK) {
        if (result == E_SEV_REQ_FULL) {
            printf("Service requst queue is full !\n");
        }
    }
        ...
    ExitISR();        // Exit ISR
}
```

Note:
1)It is used in the interrupt service program.
2)Interrupt service routine can not call CoPostMail()to send mailbox message ,otherwise the system will lead to confusion.

6.9 Message Queues

6.9.1 CoCreateQueue()

Function Prototype:

OS_EventID CoCreateQueue(

        void** qStart,

        U16   size ,

        U8   sortType

        );

Descriptions:

Create a message queue.

Parameters:

[in] qStart

message pointer storage address

[in] size

the maximum number of message in message queue

[in] sortType

waiting list arrangement types:

    EVENT_SORT_TYPE_FIFO,   FIFO order

    EVENT_SORT_TYPE_PRIO,   Preemptive Priority order

Returns:

Message Queue ID

Example usage:

```
#include "CCRTOS.h"
#define  MAIL_QUEUE_SIZE  8
OS_EventID queue;
void *MailQueue[MAIL_QUEUE_SIZE];
void TaskN (void *pdata)
{
 ...
 queue = CoCreateQueue (MailQueue, MAIL_QUEUE_SIZE,
                         EVENT_SORT_TYPE_PRIO);
 if (queue == E_CREATE_FAIL)
 {
    printf("Create a queue fail !\n");
 }
 else
 {
    printf("Queue ID : %d \n", queue);
 }
 ...
}
```

Note:

None

6.9.2 CoDelQueue()

Function Prototype:

StatusType CoDelQueue(

OS_EventID   id,

U8   opt

);

Descriptions:

Delete the message queue whose ID was designated.

Parameters:

[in] id

the designated message queue ID

[in] opt

the ways to delete the designated message queue：

EVENT_DEL_NO_PEND，delete when the waiting list is

empty

EVENT_DEL_ANYWAY，delete unconditionally

Returns:

E_INVALID_ID,   the message queue ID that was incomed

is invalid

	E_INVALID_PARAMETER,	invalid parameters, that is ,the corresponding control block is empty
	E_TASK_WAITING,	the waiting list is not empty
	E_OK,	delete successfully

Example usage:

```
#include "CCRTOS.h"
OS_EventID queue;
void *MailQueue[8];
void TaskN (void *pdata)
{
   StatusType result;
      ...
     /* Create a queue */
   result = CoDelQueue (MailQueue, OPT_DEL_ANYWAY);
   if (result != E_OK)
   {
      if (result == E_INVALID_ID)
      {
          printf("Invalid Queue ID !\n");
      }
   }
      ...
}
```

Note:

Release the resources that the message queue occupied if delete it successfully,then run a task sheduling and give the system execution time to the highest priority task.

6.9.3 CoAcceptQueueMail()

Function Prototype:

void*   CoAcceptQueueMail(
                                OS_EventID   id,
                                StatusType*   perr
                                );

Descriptions:

Request the message of the designated message queue without waiting.

Parameters:

[in]   id

Message queue ID

[out] perr

error returned types:

E_INVALID_ID,   the message queue ID

E_MBOX_EMPTY, the mailbox message whose ID
was designated is empty

E_OK,   to obtain the message successfully

Returns:

NULL,   receive failed

Others,   message pointer that received

Example usage:

```c
#include "CCRTOS.h"
OS_EventID queue;
void *MailQueue[8];
unsigned int msgdat;
void TaskN (void *pdata)
{
  StatusType result;
  void *msg;
     ...
    /* Wait for a mail, time-out:20 */
  msg = CoAcceptQueueMail (MailQueue, &result);
  if (result != E_OK)
  {
     if (result == E_INVALID_ID)
     {
         printf("Invalid Queue ID !\n");
     }
     else if (result == E_QUEUE_EMPTY)
     {
         printf("Queue is empty !\n");
     }
  }
  else
  {
     /* Process mail here */
  }
     ...
}
```

Note:

If the number of the messages in the designated message queue is more than 0 while calling AcceptQueueMail() function, the function will get the first message in the message queue, and do substract 1 operation to current message in message queue. Otherwise, if the number of the message is equal to 0, it will return error immediately.

6.9.4 CoPendQueueMail()

Function Prototype:
    void*   CoPendQueueMail(
                            OS_EventID   id,
                            U32   timeout,
                            StatusType*   perr
                            );
Descriptions:
    To obtain message of a message queue whose ID was designated.
Parameters:
    [in]   id
        the designated message queue ID
    [in]   timeout
        Overtime time, 0 means waiting   indefinitely
    [out] perr
        error returned types:
            E_CALL,   be called in the interrupt
                             service program
            E_INVALID_ID, the message queue ID
                            that   was   incomed   is
                            invalid
            E_TIMEOUT,   time is out for waiting resources
            E_OK,   to obtain the message successfully
Returns:
    To get the pointer of the queue message.

Example usage:

```c
#include "CCRTOS.h"
OS_EventID queue;
void *MailQueue[8];
unsigned int msgdat;
void TaskN (void *pdata)
{
  StatusType result;
  void *msg;
      ...
    /* Wait for a mail, time-out:20 */
  msg = CoPendQueueMail (MailQueue, 20, &result);
  if (result != E_OK)
  {
      if (result == E_INVALID_ID)
    {
        printf("Invalid Queue ID !\n");
    }
  }
  else
  {
    /* Process mail here */
  }
      ...
}
void TaskM (void *pdata)
{
  StatusType result;
      ...
  msgdat = 0x61;
  result = CoPostQueueMail (queue, (void *)&msgdat);
  if (result != E_OK)
  {
      if (result == E_INVALID_ID)
    {
        printf("Invalid Queue ID ! \n");
    }
      else if (result == E_MBOX_FULL)
    {
        printf("The Queue is full !\n");
    }
  }
}
```

Note:

1)This function runs a task sheduling while waiting and give the system execution time to the highest priority task.

2) It will do substrct 1 operation to the current message in message queue after get the message in the message queue successfully.

3) The function cann't be used in the interrupt service program.

6.9.5 CoPostQueueMail()

Function Prototype:

StatusType CoPostQueueMail(

                     OS_EventID   id,

                     void*   pmail

                    );

Descriptions:

Send message to the message queue that designated.

Parameters:

[in] id

    Message Queue ID

[in] pmail

    the pointer of the message

Returns:

E_INVALID_ID,   the message queue ID that was incomed is invalid

E_QUEUE_FULL,   the message queue is full

E_OK,   the message is send successfully

Example usage:

```c
#include "CCRTOS.h"
OS_EventID queue;
void *MailQueue[8];
unsigned int msgdat;
void TaskN (void *pdata)
{
  StatusType result;
  void *msg;
      ...
    /* Wait for a mail, time-out:20 */
  msg = CoPendQueueMail (MailQueue, 20, &result);
  if (result != E_OK)
  {
      if (result == E_INVALID_ID)
     {
         printf("Invalid Queue ID !\n");
     }
  }
  else
  {
     /* Process mail here */
  }
     ...
}
void TaskM (void *pdata)
{
  StatusType result;
     ...
  msgdat = 0x61;
  result = CoPostQueueMail (queue, (void *)&msgdat);
  if (result != E_OK)
  {
      if (result == E_INVALID_ID)
     {
         printf("Invalid Queue ID ! \n");
     }
      else if (result == E_MBOX_FULL)
     {
         printf("The Queue is full !\n");
     }
  }
}
```

Note:

If the number of the message in designated message queue is equal to the maximum number of messages, then discard the message and return error.

6.9.6 isr_PostQueueMail()

Function Prototype:

StatusType isr_PostQueueMail(

OS_EventID   id,
void*   pmail
);

Descriptions:

Send message to the message queue that designated in the inturrupt service program.

Parameters:

[IN] id

Message Queue ID

[IN] pmail

the pointer of the message

Returns:

E_SEV_REQ_FULL,   interrupt service requests is full
E_INVALID_ID,   the message queue ID that was incomed is invalid
E_MBOX_FULL,   the message queue has been full
E_OK,   send message to message queue successfully

Example usage:

```
#include "CCRTOS.h"
OS_EventID mailqueue;
int IsrDat;
void XXX_IRQHandler(void)
{
    StatusType result;
    EnterISR();        // Enter ISR
    ...
    IsrDat = 0x12;
    /* Post a mail to MailQueue that created by other tase */
    result = isr_PostQueueMail (mailqueue, &IsrDat);
    if (result != E_OK) {
        if (result == E_SEV_REQ_FULL) {
            printf("Service requst queue is full !\n");
        }
    }
      ...
    ExitISR();        // Exit ISR
}
```

Note:

1)It is used in the interrupt service program.

2) Interrupt service routine can not call CoPostQueueMail()to send message of the queue message ,otherwise the system will lead to confusion.

6.10 Flags

6.10.1 CoCreateFlag()

Function Prototype:

OS_FlagID   CoCreateFlag(

        BOOL bAutoReset,

        BOOL bInitialState

        );

Description:

Create a flag.

Parameters:

[in] bAutoReset

 Reset mode:

   1, Reset automatically

   0, Reset manually

[in] bInitialState

 Initial State:

   1, The ready state

   0, Non-ready state

Returns:

The ID of the flag,   Create successfully.

-1,   Failed to create.

Example usage:

```
#include "CCRTOS.h"
OS_FlagID flag;
void TaskN (void *pdata)
{
    ...
    /* Create a flag with auto reset, initial state: 0 */
  flag = CoCreateFlag (1, 0);
  if (result != E_OK)
  {
     if (result == E_CREATE_FAIL)
     {
         printf("Failed to create a flag!\n");
     }
  }
  else
  {
     printf("Flag ID : %d \n", flag);
  }
    ...
}
```

Note:
   None

6.10.2 CoDelFlag()

Function Prototype:
   StatusType CoDelFlag(
                        OS_FlagID   id,
                        U8   opt
                      );
Description:
   Delete a certain flag.
Parameters:
   [in] id
      The ID of a specified flag
   [in] opt
      The deleting mode:
          EVENT_DEL_NO_PEND,   Delete when the waiting list is empty.
          EVENT_DEL_ANYWAY,   Delete unconditionally.
Returns:
   E_INVALID_ID,   The incoming ID is invalid.
   E_TASK_WAITING,   The waiting list isn't empty.

E_OK,   Delete successfully.

Example usage:

```c
#include "CCRTOS.h"
OS_FlagID flag;
void TaskN (void *pdata)
{
  StatusType result;
      ...
    /* Delete the Flag */
  result = CoDelFlag (flag, OPT_DEL_ANYWAY);
  if (result != E_OK)
  {
     if (result == E_INVALID_ID)
     {
         printf("Invalid Flag ID !\n");
     }
  }
     ...
}
```

Note:

When you have deleted a flag successfully, the resources it occupied will be released. The system will start a task scheduling then so as to give the execution time to the task which has the highest priority.

6.10.3 CoAcceptSingleFlag()

Function Prototype:

StatusType CoAcceptSingleFlag(

                                OS_FlagID   id
                                );

Description:

Request for a single flag without waiting.

Parameters:

[in] id

    The ID of a specified flag

Returns:

E_INVALID_ID,   The ID of the incoming flag is invalid.
ID E_FLAG_NOT_READY,   The flag isn't in the ready state.
E_OK,   Obtain the flag successfully.

Example usage:

```
#include "CCRTOS.h"
OS_FlagID flag;
void TaskN (void *pdata)
{
  StatusType result;
      ...
  result = CoAcceptSingleFlag (flag);
  if (result != E_OK)
  {
     if (result == E_INVALID_ID)
     {
         printf("Invalid flag ID !\n");
     }
     else if (result == E_FLAG_NOT_READY)
     {
          printf("None get !\n");
     }
  }
  else
  {
        /* process here */
  }
     ...
}
```

Note:
    None

6.10.4 CoAcceptMultipleFlags()

Function Prototype:
    U32   CoAcceptMultipleFlags(
                              U32   flags,
                              U8   waitType,
                              StatusType*   perr
                              );
Description:
    Request for multiple flags without waiting.
Parameters:
    [in] flags
        The event flags which need to wait for
    [in] waitType
        The types of waiting:

OPT_WAIT_ALL,   Wait for all the flags

OPT_WAIT_ANY,   Wait for a single flag

[out] Perr

The type of the error returned:

E_INVALID_PARAMETER,   The parameter is invalid.

E_FLAG_NOT_READY,   The flag isn't in the ready state.

E_OK,   Obtain the flags successfully.

Returns:

The flag which trigger the function to return successfully

Example usage:

```c
#include "CCRTOS.h"
OS_FlagID flag;
void TaskN (void *pdata)
{
  StatusType result;
  U32 getFlag

     ...
  flagID1 = CoCreateFlag(0,0); // Manually reset, the initial configuration is not ready
  flagID2 = CoCreateFlag(0,0); // Manually reset, the initial configuration is not ready
  flagID3 = CoCreateFlag(0,0); // Manually reset, the initial configuration is not ready
  Flag = 1 << flagID1 | 1 << flagID2 | 1 << flagID3;
  getFlag = CoAcceptMultipleFlags (Flag, OPT_WAIT_ALL, &result);
  if (result != E_OK)
  {
     if (result == E_INVALID_PARAMETER)
     {
         printf("Invalid Parameter !\n");
     }
     else if (result == E_FLAG_NOT_READY)
     {
         printf("Flag not ready !\n");
     }
  }
  else
  {
      /* process Flag here */
  }
     ...
}
```

Note:

None

6.10.5 CoWaitForSingleFlag()

Function Prototype:
StatusType   CoWaitForSingleFlag(
                                   OS_FlagID   id,
                                   U32   timeout
                                   );
Description:
Wait for a certain flag with a specified ID.
Parameters:
[in] id
        The ID of a specified flag
[in] timeout
        Time-out time. 0 means waiting indefinitely.
Returns:
E_CALL,   Called in the ISR.
E_INVALID_ID,   The incoming ID of the flag is invalid.
E_TIMEOUT,   Wait overtime.
E_OK,   Obtain the flag successfully.

Example usage:

```
#include "CCRTOS.h"
OS_FlagID flag;
void TaskN (void *pdata)
{
  StatusType result;
      ...
     /* Waiting for a flag, time-out:20 */
  result = CoWaitForSingleFlag (flag, 20);
  if (result != E_OK)
  {
     if (result == E_INVALID_ID)
     {
         printf("Invalid Flag ID !\n");
     }
     else if (result == E_CALL)
     {
          printf("Error call in ISR !\n");
     }
     else if (result == E_TIMEOUT)
     {
         printf("Time Out !\n");
     }
  }
  else
  {
        /* process Flag here */
  }
     ...
}
```

Note:

  None

6.10.6 CoWaitForMultipleFlags()

Function Prototype:

  U32 CoWaitForMultipleFlags(

  U32   flags,
  U8   waitType,
  U32   timeout,
  StatusType*   perr
  );

Description:

Wait for multiple flags.

Parameters:

[in] flags

The flags which need to wait for

[in] waitType

The type of waiting:

OPT_WAIT_ALL,   Wait for all the flags

OPT_WAIT_ANY,   Wait for a single flag

[in] timeout

Time-out time. 0 means waiting indefinitely.

[out] perr

The type of the errors returned:

E_CALL,   Called in the ISR.

E_INVALID_PARAMETER,   The parameter is invalid.

E_TIMEOUT,   Wait overtime.

E_FLAG_NOT_READY,   The flag isn't in the ready state.

E_OK,   Obtain successfully.

Returns:

The flag which trigger the function to return successfully

Example usage:

```c
#include "CCRTOS.h"
OS_FlagID flag;
void TaskN (void *pdata)
{
  StatusType result;
  U32   getFlag;
     ...
  flagID1 = CoCreateFlag(0,0); // Manually reset, the initial configuration is not ready
  flagID2 = CoCreateFlag(0,0); // Manually reset, the initial configuration is not ready
  flagID3 = CoCreateFlag(0,0); // Manually reset, the initial configuration is not ready
  Flag = 1 << flagID1 | 1 << flagID2 | 1 << flagID3;
  getFlag = CoWaitForMultipleFlags (Flag, OPT_WAIT_ALL, 20, &result);
  if (result != E_OK)
  {
     if (result == E_INVALID_PARAMETER)
    {
        printf("Invalid parameter !\n");
    }
     else if (result == E_CALL)
    {
         printf("Error call in ISR !\n");
    }
     else if (result == E_TIMEOUT)
    {
        printf("Time Out !\n");
    }
  }
  else
  {
        /* process Flag here */
  }
     ...
}
```

Note:

   None

6.10.7 CoClearFlag()

Function Prototype:

    StatusType   CoClearFlag(

                            OS_FlagID   id
                    );

Description:

Set an event to the non-ready state.

Parameters:

[in] id

The ID of a specified event flag

Returns:

E_INVALID_ID,　　The incoming flag ID is invalid.

E_OK,　　　　　Clear successfully.

Example usage:

```
#include "CCRTOS.h"
OS_FlagID flag;
void TaskN (void *pdata)
{
  StatusType result;
     ...
  result = CoClearFlag (flag);
  if (result != E_OK)
  {
     if (result == E_INVALID_ID)
    {
        printf("Invalid flag ID !\n");
    }
  }
     ...
}
```

Note:

This function usually acts on the flag which is reset manually.

6.10.8 CoSetFlag()

Function Prototype:

StatusType   CoSetFlag(

OS_FlagID   id

);

Description:

Set an event to the ready state.

Parameters:

[in] id

The ID of a specified flag

Returns:

E_INVALID_ID,   The incoming flag ID is invalid.

E_OK,   Set successfully.

Example usage:

```
#include "CCRTOS.h"
OS_FlagID flag;
void TaskN (void *pdata)
{
  StatusType result;
      ...
  result = CoSetFlag (flag);
  if (result != E_OK)
  {
      if (result == E_INVALID_ID)
    {
          printf("Invalid flag ID !\n");
    }
  }
      ...
}
```

Note:

If the task is activated successfully, the calling of this function will start a task scheduling.

6.10.9 isr_SetFlag()

Function Prototype:

StatusType isr_SetFlag(

OS_FlagID   id

);

Description:

Set the flag with a certain ID to the ready state in the ISR.

Parameters:

[IN] id

The ID of the flag

Returns:

E_SEV_REQ_FULL,   The Interrupt Service Request is full.

E_INVALID_ID,   Invalid flag ID.

E_OK,   Set the flag successfully.

Example usage:

```c
#include "CCRTOS.h"
OS_FlagID flag;
void XXX_IRQHandler(void)
{
    StatusType result;
    EnterISR();        // Enter ISR

    ...
    /* Set a flag that created by other test */
    result = isr_SetFlag (flag);
    if (result != E_OK) {
        if (result == E_SEV_REQ_FULL) {
            printf("Service request queue is full !\n");
        }
    }
      ...
    ExitISR();        // Exit ISR
}
```

Note:
1) This function is used in the ISR.
2) You can't call CoSetFlag() to set the flags in the ISR, otherwise, the system would be in chaos.

6.11 System Utilities

6.11.1 CoTickToTime()

Function Prototype:
    void TickToTime
                    (
                    U32   ticks,
                    U8*   hour,
                    U8*   minute,
                    U8*   sec,
                    U16*   millsec
                    );
Description:
    Convert the systick number to proper time.
Parameters:
    [in] ticks
            systick number
    [in] hour
            Hours
    [in] minute
            Minutes
    [in] sec
            Seconds
    [in] millsec
            Milliseconds
Returns:
    None

Example usage:

```
#include "CCRTOS.h"
void TaskM (void *pdata)
{
    U8 Hour, Minute, Second;
    U16 Millisecond;
    ...
    CoTickToTime (1949,
                    &Hour,
                    &Minute,
                    &Second,
                    &Millisecond);
    printf ("1949 system ticks = %2d-%02d-%02d-%03d \n",
    Hour, Minute, Second, Millisecond);
    ...
}
```

Note:

None

6.11.2 CoTimeToTick()

Function Prototype:

StatusType CoTimeToTick(

                          U8 hour,
                          U8 minute,
                          U8 sec,
                          U16 millsec,
                          U32* ticks
                        );

Description:

Convert the time to proper systick number.

Parameters:

[in] hour

Hours

[in] minute

Minutes

[in] sec

Seconds

[in] millsec

Milliseconds

[in] ticks

Systick number

Returns:

    E_INVALID_PARAMETER,   The parameter is invalid.

    E_OK,   Convert successfully.

Example usage:

```c
#include "CCRTOS.h"
void TaskM (void *pdata)
{
    U32 tick;
    StatusType result;
    ...
    result = CoTimeToTick (19,
                           49,
                           10,
                           1,
                           &tick);
    if (result != E_OK)
    {
        if (result == E_INVALID_PARAMETER)
        {
            printf("Invalid parameter be passed and convert fail !\n");
        }
    }
    ...
}
```

6.12 Others

6.12.1 CoIdleTask()

Function Prototype:
    void   CoIdleTask
                   (
                    void* pdata
                   );
Description:
    System Idle task code.
Parameters:
    [IN] pdata
        The parameter list passed to IDLE task. The system set it as NULL.
Returns:
    None
Example usage:

```
void CoIdleTask(void* pdata)
{
 /* Add your code here */
 for (;;)
 {
 /* Add your code here */
 }
}
```

Note:
    As a resident function of the system, this function can't be deleted or exit
    automatically. You can call it to achieve the statistics of some system
    parameters.

6.12.2 CoStkOverflowHook()

Function Prototype:
    void   CoStkOverflowHook(
                        OS_TID   taskID
                        );
Description:
    The call-back function when the system stack overflows.
Parameters:
    [IN] taskID
        The task ID which caused the system stack overflows

Returns:

None

Example usage:

```
void CoStkOverflowHook(OS_TID taskID)
{
 /* Process stack overflow here */
}
```

Note:

This function is called to handle the overflow of the system stack. By calling it, you can customize the operation towards the stack overflow. Once you don't do any operation and exit this function, the system will be in chaos.