제목: uC/OS-II Task Programming 기초 지식
작성: 신동하

1. uC/OS-II 실시간 커널에서 태스크 상태(state)

uC/OS-II 실시간 커널은 다중 태스크 수행을 지원하는 커널이다. uC/OS-II의 태스크는 아래와 같이 5개의 상태인 dormant, ready, running, waiting 및 IRS(Interrupt Service Routine) 중 하나를 가진다.

- dormant 상태의 태스크는 메모리 상에 존재하나 아직 커널에 의하여 태스크로 생성되지 않은 상태이다.
- ready 상태의 태스크는 수행 가능하나 태스크 우선순위가 낮아서 아직 수행되지 못하는 태스크이다.
- running 상태의 태스크는 현재 프로세서에 의하여 수행 중인 태스크이다.
- waiting 상태의 태스크는 어떤 사건(event)이 발생하기를 기다리는 태스크이다. 대표적인 사건은 입출력이 끝나는 일, 공유 자원의 사용이 가능하게 되는 일, 주어진 시간이 지나는 일 등이다.
- IRS 상태는 인터럽트가 발생하여 프로세서가 인터럽트 처리 루틴을 수행하고 있는 상태이다.

다음 그림 1은 태스크 상태 이전을 그림으로 나타낸 것이다. 상태 이전을 위하여 필요한 커널 함수가 상태 이전을 의미하는 화살표에 표시되어 있다. (참고 자료 1)

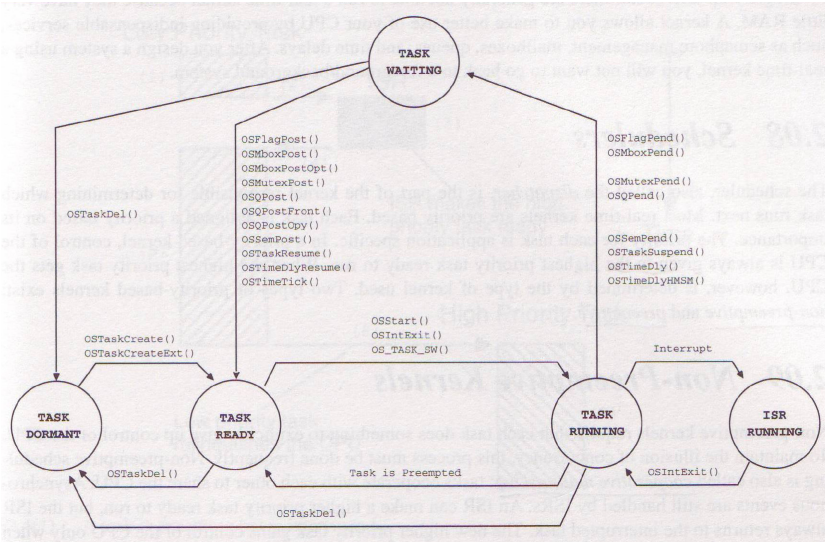

그림 1 uC/OS-II 태스크 상태 이전

각 태스크는 dormant 상태에 있다가 커널이 태스크로 생성하면(예: OSTaskCreate) ready 상태로 된다. 이 상태에서 콘텍스트 스위칭(context switching)이 발생하여 running 상대가 되면 이 태스크는 프로세서(CPU)를 수행한다. 모든 태스크는 프로세서를 수행하다가 공유 자원을 기다리거나(예: OSSemPend), 일정 시간을 기다리거나(예: OSTimeDly) 혹은 수행 상태를 바꾸어서(예: OSTaskSuspend) waiting 상태로 바뀐다. waiting 상태에 있는 태스크는 기다리는 자원이 생기거나(예: OSSemPost), 일정 시간을 기다리다가(예: OSTimeTick) 다시 ready 상태가 된다.

2. uC/OS-II 실시간 커널의 선점형(preemptive) 태스크 스케줄

uC/OS-II 실시간 커널은 선점형 태스크 스케줄을 지원하는 커널이다. 이를 그림으로 설명하면

다음과 같다. (참고 자료 1)

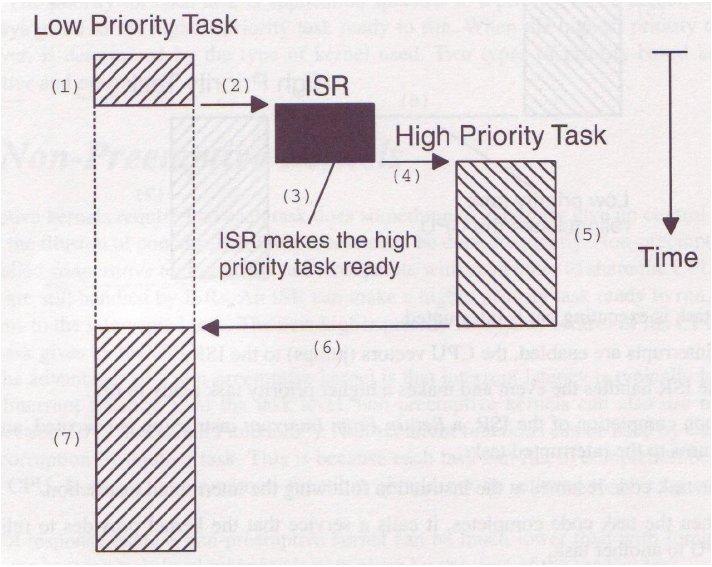

그림 2 선점형(preemptive) 태스크 스케줄

위 그림 2에서 (1) 낮은 우선순위를 가지는 Task A가 수행되는(running) 도중 (2) 인터럽트가 발생하여 (3) ISR(Interrupt Service Routine)이 수행된다. (4) ISR의 수행이 끝나면 제어가 Task A로 넘어가지 않고 (5) 태스크 중에서 가장 우선순위가 높은 태스크인 Task B가 수행된다. (6) Task B가 수행되다가 CPU의 사용을 포기하면 (7) Ready 상태의 태스크 중에서 가장 우선순위가 높은 Task A가 다시 수행된다.

3. uC/OS-II 실시간 커널에서 태스크 생성

u/OS-II 실시간 커널에서 새로운 태스크를 생성하는 함수는 INT8U OSTaskCreate(void (*task)(void *pd), void *pdata, OS_STK *ptos, INT8U prio)이다. 이 함수의 인수의 의미는 다음과 같다.

- task: 수행할 태스크에 대응하는 함수로 이 함수의 return 값의 type은 void이고 인수의 type은 void * 이다.
- pdata: 태스크 함수의 인수이다. type은 void * 이다.
- ptos: 태스크 스택의 top 주소이다. 태스크 스택은 OS_STK 형(type)의 값을 가지는 요소의 array로 정의된다. 만약 태스크 스택이 OS_STK TaskStk[1024]로 정의되어 있고 스택이 메모리 주소 high에서 low로 push된다면(이 경우 매크로 OS_STK_GROWTH 값이 1임) ptos로 &TaskStk[1023]을 넘기면 된다. 만약 스택이 메모리 주소 low에서 high로 push되면(이 경우 매크로 OS_STK_GROWTH 값이 0임) ptos로 &TaskStk[0]을 넘기면 된다.
- prio: 태스크의 우선순위이다. 사용 가능한 우선순위는 4 ~ 59이며 이 수가 작을수록 우선순위가 높다. 모든 태스크는 서로 다른 고유의 우선순위를 가진다.

* 참고: uC/OS-II ARM 포팅 버전에서 사용하는 주요 type들(파일 os_cpu.h에 정의되어 있음)

```
typedef unsigned char BOOLEAN;          /* Unsigned 8 bit quantity */
typedef unsigned char INT8U;            /* Unsigned 8 bit quantity */
typedef signed char INT8S;              /* Signed 8 bit quantity */
typedef unsigned short INT16U;          /* Unsigned 16 bit quantity */
typedef signed short INT16S;            /* Signed 16 bit quantity */
```

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

다음은 태스크 Task1을 생성하고 수행시키는 uC/OS-II 응용 프로그램의 보기이다.

```
#include "includes.h"

OS_STK TaskStk1[1024];

void main(void)
{
  OSInit();                            /* uC/OS-II 초기화 */
  ...
                                       /* 태스크 Task1 생성 */
  OSTaskCreate(Task1, (void *) 0, &TaskStk1[1023], 25);
  ...
  OSStart();                           /* 다중 태스크(multitasking) 수행 시작 */
}

void Task1(void *pdata)
{
  TIMER_enable();                      /* 첫 번째 task에서 항상 call해야 함 */

  for(;;){
    ...
  }
}
```

4. uC/OS-II 실시간 커널에서 태스크 수행 지연

uC/OS-II 실시간 커널은 일정 시간 태스크의 수행을 지연시키는 함수 void OSTimeDly(INT16U ticks) 및 void OSTimeDlyHMSM(INT8U hours, INT8U minutes, INT8U seconds, INT8U milli)를 제공한다. 이 함수는 인수로 주어진 tick 수 혹은 인수로 주어진 시간만큼 현재 수행 중인 태스크를 지연시킨다. 다음은 함수 OSTimeDly() 및 OSTimeDlyHMSM()을 사용한 태스크 프로그램의 보기이다.

```
void Task1(void *pdata)
{
  for(;;){
    ...
    OSTimeDly(10);                     /* 이 태스크를 10 ticks 만큼 지연시킴 */
    ...
  }
```

```
}

void Task2(void *pdata)
{
  for(;;){
    ...
    OSTimeDlyHMSM(0, 0, 1, 0);          /* 이 태스크를 1초 지연시킴 */
    ...
  }
}
```

수행 지연된 태스크는 함수 INT8U OSTimeDlyResume(INT8U prio)에 의하여 다시 수행될 수 있다. 여기서 인수 prio는 수행이 지연된 태스크의 우선순위이다. 이 함수는 아래와 같은 의미를 가지는 값을 넘겨준다.

OS_NO_ERROR	함수 부름이 성공적임
OS_PRIO_INVALID	OS_LOWEST_PRIO 보다 큰 우선순위를 지정했을 때
OS_TIME_NOT_DLY	태스크가 시간이 지나기를 기다리고 있지 않을 경우
OS_TASK_NOT_EXIST	주어진 우선순위의 태스크가 생성되지 않았을 경우

다음은 함수 OSTimeDlyResume()의 사용 보기이다.

```
void Task3(void *pdata)
{
  INT8U err;
  for(;;){
    ...
    err = OSTimeDlyResume(10);          /* 우선순위 10의 태스크를 시작시킴*/
    if(err == OS_NO_ERR) {
      ...                               /* 태스크가 시작됨 */
    }
    ...
  }
}
```

5. uC/OS-II 실시간 커널에서 태스크 수행 순서 조정

uC/OS-II 실시간 커널은 가장 높은 우선순위의 태스크를 항상 수행시킨다. 어떤 태스크가 수행 중일 때 인터럽트가 발생하지 않고 이 태스크가 스스로 프로세서의 사용을 중단하지 않는 경우 이 태스크는 계속 수행된다. 어떤 응용 프로그램에서 여러 태스크를 순서적으로 동시에 수행시키고 싶을 때는 앞에서 설명한 태스크 지연 함수를 사용하여 이를 구현할 수 있다. 어떤 태스크의 수행이 지연되면 ready 상태의 태스크 중 가장 우선순위가 높은 태스크가 수행된다. 다음은 10개의 태스크를 순서적으로 수행시키는 프로그램의 보기이다.

```
  ...
  for (i = 0; i < 10; i++) {{              /* 10개의 태스크를 생성함 */
    OSTaskCreate(Task, (void *) &TaskData[i],
```

```
                    &TaskStk[i][TASK_STK_SIZE - 1], 11 + i);
  }
  ...

void Task(void *pdata)
{
  for (;;) {
    ...                                    /* 태스크가 주어진 일을 함 */
    OSTimeDly(1);                          /* 1 tick 만큼 태스크 수행을 지연시킴 */
  }
}
```

6. uC/OS-II 실시간 커널에서 태스크 수행 통계 파악

uC/OS-II는 수행시간 통계(statistics) 제공을 위한 커널 태스크를 수행시킬 수 있다. 이 태스크는 uC/OS-II의 매크로 OS_TASK_STAT_EN이 1일 때 매초 한번씩 자동 수행되어 CPU usage(%)를 계산한다. uC/OS-II에서 이 태스크의 수행을 위해서는 단 하나의 첫(first and only) 사용자 태스크를 생성하여 수행시킨 다음 다른 사용자 태스크를 수행시키기 전에 함수 OSStatInit()을 불러 주어야 한다. 다음은 첫 사용자 태스크에서 함수 OSStatInit()을 부르는 보기 프로그램이다.

```
void FirstAndOnlyTask (void *pdata)
{
  ...
  OSStatInit();
  ...
  OSTaskCreate(...);                       /* 다른 사용자 태스크 생성 */
  OSTaskCreate(...);
  ...
  for (;;) {
    ...
  }
}
```

아래 프로그램은 몇 가지 통계 값을 출력하는 사용자 태스크의 보기이다.

```
void TaskStatDisp(void)
{
  UART_clear();                            /* 화면을 clear 함 */
  UART_goto(0, 23);                        /* 화면의 (0, 23)으로 커서를 옮김 */

  UART_puts("#Tasks: 0x");                 /* 태스크 수 출력 */
  UART_puthex(OSTaskCtr);

  UART_puts(", CPU Usage(%): 0x");         /* CPU usage 축력 */
  UART_puthex(OSCPUUsage);

  UART_puts(", #CtxSW/Sec: 0x"); /* 초당 context switch 수 출력 */
  UART_puthex(OSCtxSwCtr);
```

```
    OSCtxSwCtr = 0;
}
```

7. uC/OS-II 실시간 커널에서 세마포를 사용

uC/OS-II 실시간 커널에서는 사용자가 세마포(semaphore) 변수를 선언하고, 세마포의 초기 값을 정의하고, 세마포를 pending 혹은 posting 할 수 있다. 다음은 세마포 변수 Sem1을 선언하고 초기 값을 1로 정의(함수 OSSemCreate를 인수 1으로 부름)하는 프로그램의 보기이다. 세마포 변수는 OS_EVENT * 로 선언한다.

```
OS_EVENT *Sem1;
...
int main(void)
{
    OSInit();
    ...
    Sem1 = OSSemCreate(1);
    ...
    OSTaskCreate(Task1, (void *) 0, &TaskStk1[TASK_STK_SIZE - 1], 11);
    ...
    OSStart();
}
```

세마포는 공유 자원을 여러 태스크가 사용하기 위하여, 여러 태스크가 서로 동기화되어(synchronized) 일을 수행할 때 혹은 주어진 이벤트가 발생하였음을 알릴 때 사용된다. 세마포에 대한 동작은 PEND와 POST가 있는데 uC/OS-II에서는 각각 함수 OSSemPend() 및 OSSemPost()가 이 동작을 수행한다. 아래는 이들 함수의 설명이다.

- void OSSEMPend(OS_EVENT *pevent, INT16U timeout, INT8U *err)
  - pevent: 기다리는(pending) 세마포를 가리키는 세마포 변수 포인터이다.
  - timeout: clock tick 수로서 최대 기다리는 시간을 나타낸다. 이 값이 0이면 세마포 pevent가 사용가능할 때 까지 영원히 기다린다.
  - err: 오류 code에 대한 포인터이다. 이 값은 아래와 같은 의미를 가지고 있다.
    - OS_NO_ERR: 원하는 세마포가 사용가능할 경우
    - OS_TIME_OUT: 주어진 timeout 내에 세마포가 사용 불가능할 경우
    - OS_ERR_EVENT_TYPE: pevent가 세마포를 가리키지 않을 경우
    - OS_ERR_PEND_ISR: ISR 내에서 이 함수를 부른 경우 (오류임)
    - OS_ERR_PEND_NULL: pevent 변수가 NULL인 경우
- void OSSEMPend(OS_EVENT *pevent, INT16U timeout, INT8U *err)
- INT8U void OSSEMPost(OS_EVENT)
  - pevent: 넘겨주는(post) 세마포를 가리키는 세마포 변수 포인터이다.
  - 복귀 값(return value)
    - OS_NO_ERR: 세마포가 성공적으로 POST된 경우

- OS_SEM_OVF: 세마포 카운트에 overflow가 생긴 경우

- OS_ERR_EVENT_TYPE: pevent가 세마포를 가리키지 않을 경우

- OS_ERR_PEND_NULL: pevent 변수가 NULL인 경우

다음 그림 3는 2개의 태스크가 2개의 세마포를 가지고 서로 동기화하여 수행되는 원리를 나타낸 그림이다.

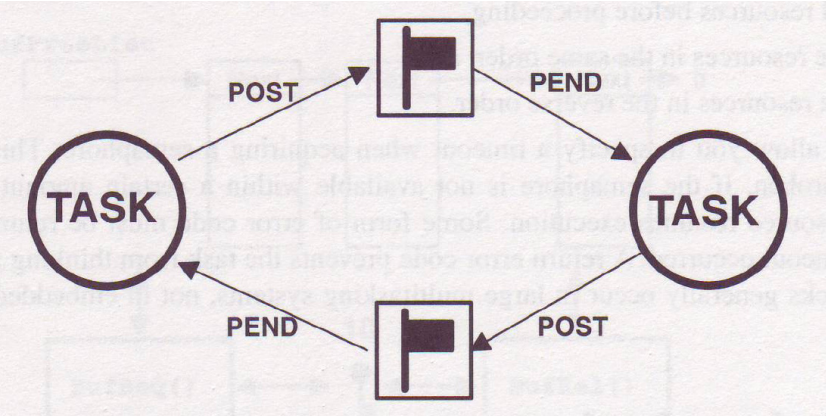

그림 3 세마포를 통한 태스크 동기화

위 그림 2를 바탕으로 작성된 프로그램 보기는 아래와 같다. 이 프로그램에서 세마포 Sem1과 Sem2는 각각 1 및 0으로 초기화되어 있다고 가정한다.

```
void Task1(void *pdata)
{
  INT8U err;

  for (;;) {
    OSSemPend(Sem1, 0, &err);
    /* 여기서 Task1의 주어진 일을 수행한다. */
    OSSemPost(Sem2);
  }
}

void Task2(void *pdata)
{
  INT8U err;

  for (;;) {
    OSSemPend(Sem2, 0, &err);
    /* 여기서 Task2의 주어진 일을 수행한다. */
    OSSemPost(Sem1);
  }
}
```

8. Random 번호 생성 방법

Random 번호를 생성하기 위해서는 아래와 같이 함수 U16 random(U16 n)을 정의하고 이를 자연수 인수 N을 사용해서 부른다. 이 때 복귀 값은 0~(N-1) 사이의 random 번호이다. 함수 random을 부를 때는 아래와 같이 세마포를 사용하여 동시에 여러 태스크가 함수 random을 부르지 않도록 하는 것이 좋다.

```
  ...
  OSSemPend(RandomSem, 0, &err);
  x = random(80);
  y = random(20);
  OSSemPost(RandomSem);
  ...

U16 random(U16 n)
{
  static U32 rand = 0;

  rand = rand * 1103515245 + 12345;
  return (U16)((rand / 65536) % n);
}
```

참고자료

1. Jean J. Labrosse, MicroC/OS-II The Real-Time Kernel Second Edition, CMP Books, 2002.
2. 성원호, MicroC/OS-II 실시간 커널, 에이콘 출판사, 2005.

끝.