uC/OS-Ⅱ 뛰어넘기(Ⅱ)

uC/OS-II 개관

용어는 이해를 위한 가장 기본적인 정의이다. 따라서, 이번 회에는 RTOS와 관련된 개념 및 용어들을 알아보고 RTOS의 한 종류인 uC/OS를 통해 실질적인 접근을 해보고자 한다. 그리고, 언급된 내용들을 uC/OS의 서비스 함수 및 커널을 통해 개념들을 익히고 활용할 수 있도록 하자.

글: 김대홍/CyberLab 실장, 삼성 첨단기술연수소 RTOS강사
redizi@armkorea.com

RTOS

이제부터는 이전 회에서 언급됐던 OS 중에서 RTOS에 대해서 본격적으로 다루기로 할 것이다. 개인적으로는 RTOS에 대해서 상당한 애착을 가지고 있다. 맨 처음 접한 임베디드 시스템에서의 OS이기도 하지만, 그것만이 가진 매력을 무시할 수가 없어서 이기도 할 것이다.

RTOS도 하나의 완벽한 OS이지만, Kernel의 동작을 봤을 때, 상당히 이해하기 쉬운 시스템이다. 그래서 멀티태스킹, 드라이버, 데드락 등등의 여러 가지 OS의 용어들을 쉽게 이해할 수 있으며, 연습을 통해 쉽게 접근할 수 있는 OS이기도 하다. 그래서 덩치가 큰 Linux나 Windows를 처음부터 무작정 이해하고 응용프로그램을 짜려면 상당히 어려운 점들이 많지만, RTOS를 통해서 OS의 개념을 익히고 들어간다면 훨씬 좋은 능력을 갖추게 될 것이다.

국내에 많은 개발자들이 있지만, 교육여건 및 환경이 갖춰지질 못한 상황에 의해 이러한 OS들을 100% 이해하고 활용하지 못하는데 아쉬움이 많지만, RTOS를 통해서 이러한 접근을 시도해 본다면 좋은 결과가 있으리라 생각한다.

OS의 용어들을 살펴보기 전에 먼저 이전 회에서 언급했던 RTOS의 특징들 중에서 몇 가지 사항들의 내용을 살펴보고 넘어가자.

▶ Multitasking

Linux나 Windows와 같이 멀티태스킹을 지원한다. 뒷부분에서 좀더 자세하게 다루겠지만, 동시에 여러 개의 타스크가 수행될 수 있음을 의미한다. 이것을 잘 이해할 수 있으면 linux나 windows도 좀더 쉽게 생각할 수 있을 것이다.

▶ Preemptive

커널이 수행중인 타스크를 강제적으로 정지시킬 수 있다는 의미이다. 이것은 RTOS에 있어서 아주 중요한 성질이기도 하다.

▶ Deterministic

실행시간을 예측할 수 있다는 의미인데, RTOS의 가장 큰 특징 중 하나이다. 현재 수행중인 타스크들의 실행시간이 예측가능해서 언제 어떠한 일들을 할 수 있는지 예측이 가능하다. 일반적인 OS들은 이러한 성질들이 없다.

이러한 내용들을 머릿속에 염두해 두고 몇 가지 용어들을 정리해보면서 OS를 배워보도록 하자.

■ Task

문제를 해결하기 위한 기본적인 프로그램 덩어리로 일반적으로 OS가 제어하는 기본단위가 된다. 즉, 스케줄링이나 자원 할당의 단위가 된다. 보통 thread라고도 불린다. 여기서 thread

와 task간의 정의에 대해 여러 가지 의견을 제시하시는 분들이 있으리라 생각이 된다. 하지만, task와 thread가 수행되는 환경과 기능을 생각해 보면 두 개가 거의 유사하고 task와 thread의 용어가 혼합되어 사용되는 경우가 거의 없으므로, 여기서는 task와 thread를 같은 개념으로 보고 통일성을 위해 task란 용어만을 사용하도록 하자.

■ Multi-tasking

단어 그대로 Task를 여러 개 수행시키는 것을 의미한다. 그림 1에서 보듯이 각각의 Task는 서로의 영향을 받지 않고, 독립적으로 각자 수행이 된다. **그림 1**의 오른쪽 그림을 보면 3개의 task가 동작이 되는데, 이것은 마치 일반적인 시퀀스 프로그램에서 main 함수 3개가 동시에 수행되는 것처럼 동작을 하게 된다. Windows나 Linux를 사용해온 독자들이라면 이것이 별로 대수롭지 않을 것이다. 하지만, 이것을 이용해 프로그램을 짜야 하는 프로그래머라면 이 대수롭지 않은 multitasking의 동작 때문에 잠 못드는 나날들을 보내고 있을 것이다.

> **시퀀스 프로그램**
>
> 보통 main( ) 함수에서 한 단계씩 순차적으로(위에서 아래로) 실행해 나가는 방식의 프로그램을 말한다. DOS 시절에 프로그램을 하셨던 분들이나 linux 또는 windows에서 처음 프로그램을 하시는 분들도 대게 이러한 방식으로 프로그램을 하게 된다. windows나 linux에서 프로그램을 짠다고 mutitasking 방식의 프로그램을 짜는 것이 아니라는 점을 명심해야 한다.

Multitasking은 한 개의 CPU에서 여러 개의 Task를 빠른 속도로 조금씩 잘라서 처리함으로써, 동시에 여러 개의 Task가 수행되는 것처럼 보이게 하는 효과이다.

Multi-processor의 경우는 약간 다르다. Multi-processor의 경우, CPU가 여러 개이기 때문에 실제로 동시에 여러 개의 명령을 수행할 수 있다. 하지만, 앞으로 우리가 다루는 범위를 CPU가 한 개인 경우로 한정을 하도록 하자.

Multitasking의 눈속임을 확인하기 위해서 옆에 있는 볼펜을 하나씩 집어보자. 그 다음 볼펜을 수평으로 놓고 끝을 살짝

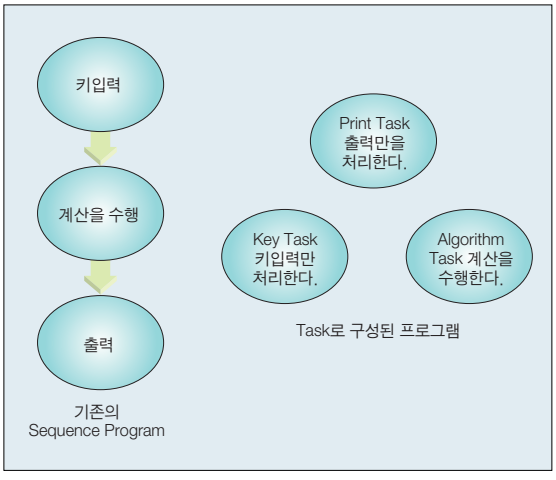

그림 1. Sequence Program과 Multitasking Program

잡고 위아래로 살살 흔들어보자. 어렸을 때 이런 장난을 많이 했던 기억이 나면서, 볼펜이 여러 개로 보일 것이다. 즉 CPU는 하나인데, CPU가 여러 개인 것처럼 여러 개의 task를 동시에 수행되게 하는 것이 multitasking이다.

Multitasking은 다음과 같은 여러 가지 장점들을 제공한다.

장점

1) CPU의 효율을 높여준다.

CPU의 효율이 높다는 것은 CPU가 놀지 않고 계속 연산을 한다는 의미이다. 어떻게 해서 multitasking이 CPU의 효율을 높여줄까? 우리가 일반적인 프로그램 방식을 생각해보자. 그림 1의 sequence program 방식으로 프로그램을 짰을 때, 키입력 부분을 생각해보자. 일반적으로 키입력이 들어올 때까지 CPU는 무의미한 무한루프를 돌게 될 것이다. 또한, 타이밍을 위한 delay문도 마찬가지로 쓸모없는 CPU 시간을 증가시킬 것이다. 하지만, Multitasking에서는 이러한 시간들을 전부 다른 task가 수행하는 시간으로 돌리게 된다.

2) 분할적 구조를 제공한다.

Task의 개념을 다시 살펴보자. Task는 문제를 해결하는 단위이기 때문에 작업시 분할적인 구조로 쉽게 나눌 수 있게 된다. 그래서, 여러 명이 참가하는 프로젝트에서 부분에 따라서 독자적인 작업 후에 취합하는 형태로 개발이 가능하게 된다. 또한 디버깅시 문제가 발생한 부분을 손쉽게 발견하여 처리할 수

있게 된다.

이렇게 이야기하면 multitasking이 최선의 방법처럼 보이지만, 아래와 같은 여러 가지 단점도 가지고 있다.

단점

1) Scheduling의 Overhead

multitasking을 구현하기 위해서는 task를 관리해주는 커널이 수행되어야 하는데, 커널의 수행시간은 응용프로그램의 수행시간을 뺏기 때문에 될 수 있는 대로 짧아야한다.

2) Task간의 상호작용 예측의 어려움

Task는 별개로 동작한다. 그래서 버그가 생겼을 때, 각 task의 상태가 항상 다를 수 있고 상호간의 영향에 의해 다양한 상황이 발생할 수가 있어서 디버깅시 상당한 어려움을 겪게 된다.

3) 간단한 시스템에 적용은 무리

OS라는 것 자체가 어느 정도의 시스템 사양 및 환경을 요구하는 것이기 때문에 OS의 적용에 의해 오히려 시스템의 성능을 떨어뜨릴 수도 있다는 점을 명심해야 한다.

uC/OS-II

uC/OS는 RTOS를 배우고자 하는 분들에게는 교과서와 같은 OS이다.

물론, 이 글을 읽고 있는 독자라면 한번씩은 들어보거나 이미 책을 구입해서 접해 본적이 있으리라 생각이 된다. uC/OS는 가장 기본적인 Kernel의 기능을 가지고 있는 OS이고 소스가 오픈되어 공부할 때 많은 도움이 된다. 또한, 저자가 커널을 분석한 책까지 출판을 했기 때문에 이해하기 더욱 쉽다. 이 책에는 일반적인 OS의 개념들과 포팅 부분에 대한 언급도 되어있어서 공부하기를 원하는 독자들에게 많은 도움이 될 것이다. 현재 한글 번역본도 있으니 아까워하지 마시고 구입해보시기를 권장한다(참고로, uC/OS는 무료가 아니다. 책을 구입한 사람에 한해서 교육용 목적으로 사용할 경우에만 자유롭게 사용이 가능하며, 상업적 목적으로 사용할 경우 별도의 라이선스를 받아야 한다.)

현재 최신버전은 2.7x이지만, 책구입시 제공되는 버전은 2.52버전이다. 초기의 2.0버전에 비해 많은 기능들이 들어가면서 기능상으로 좋아지긴 했지만, 그만큼 복잡해진 면들도 있어서 처음 접하는 분이라면 모든 부분을 처음부터 한번에 다루기보다는 조금씩 발췌를 해서 본다면 좀더 쉽게 접근이 가능하다.

책과 같이 들어있는 uC/OS의 배포본은 x86을 기본으로 한다. 그리고 컴파일러는 borland사의 BC45 컴파일러를 사용할 수 있도록 설정이 되어있다(정식 명칭은 Borland C++ 4.5이지만, 일반적으로 BC45라고 부른다). 가능하다면 uC/OS를 처음 접하는 독자들은 PC에서 충분히 연습을 해서 RTOS의 개념과 사용법을 숙지하고 임베디드 시스템에 적용하기 바란다.

여기서 다루는 예제들은 강좌 뒷부분에서 다룰 ARM 포팅 부분을 제외한다면 배포본을 기준으로 PC에서 제작하고 실행하도록 할 것이다.

> uC/OS의 공식홈페이지
> *http://www.ucos-ii.com*

uC/OS 구성

uC/OS는 그림 2와 같은 트리 구조를 갖는다. 각 폴더의 내용은 다음과 같다.

+DOC
  uC/OS와 관련된 문서들이 있다.
+EX1_x86L~EX4_x86L
  예제 파일들
+Ix86L
  x86용 Large model 포팅 파일들
+Ix86L-FP

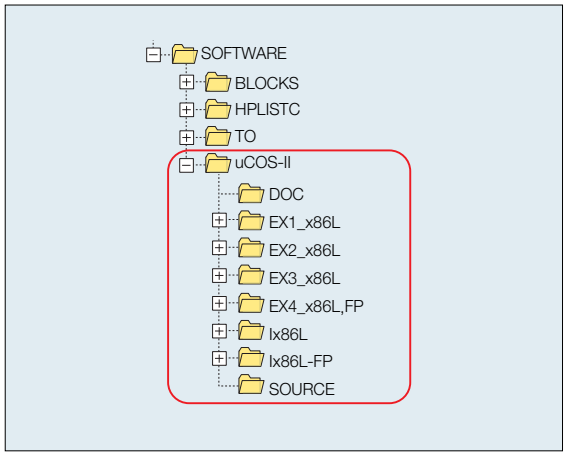

그림 2. uC/OS 폴더 구조

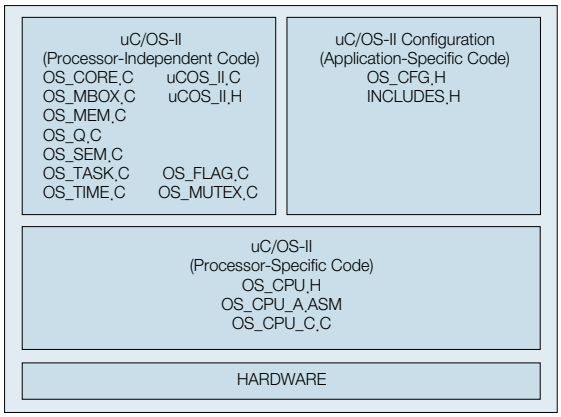

그림 3.uC/OS 커널 파일 구조

x86용 Large Model Floating point 지원 포팅 파일들
+SOURCE
uC/OS 소스파일들(프로세서에 독립적)

uC/OS가 컴파일 되서 동작되기 위해서는 크게 세부분의 구성파일들이 필요하다.

1) 프로세서에 의존적인 커널파일

프로그램이 동작하는 CPU가 무엇인가에 따라서 달라진다. uC/OS가 지원하는 기본 프로세서는 x86이기 때문에 모든 x86 계열에서 동작이 가능하다. 그래서, 임베디드쪽에서 아직까지 생산이 되는 186~486계열에서도 거의 수정없이 바로 동작을 할 수 있게 된다. 우리가 사용하는 PC도 x86이기 때문에 컴파일만 되면 바로 사용이 가능하다. uC/OS는 위에서 설명된 것과 같이 x86용 두 가지 버전을 지원한다. 만약 다른 프로세서를 지원해야하는 경우라면 사용자가 직접 수정을 해주어야 한다.

uC/OS에서의 포팅 과정은 이 부분을 CPU에 맞게 수정해주는 과정을 말한다.

2) 프로세서에 독립적인 커널파일

프로세서와 상관없이 동작을 하게 된다. 그래서 C컴파일러가 지원하는 프로세서라면 어디서든지 컴파일이 된다. 사실, 요즘 생산되는 거의 모든 CPU들이 C를 지원하기 때문에 걱정할 필요는 없다.

3) 애플리케이션

사용자 프로그램 부분으로 애플리케이션에 따라 다르게 작성이 된다.

uC/OS 커널 파일

그림 3은 uC/OS의 커널 구조와 관련된 파일들을 보여준다. 위에서 언급된 1)과 2)에 해당하는 부분으로 구체적인 파일명들이 나열되어 있다.

OS_CPU.H	header file
OS_CPU_A.ASM	assembler file
OS_CPU_C.C	C file

프로세서에 의존적인 커널 파일들

프로세서에 의존적인 커널 파일들은 모두 3개인데, 포팅을 한다는 것은 이 파일들을 CPU에 맞게 옮겨준다는 의미가 된다. 더 자세한 내용은 ARM 포팅 과정에서 다루도록 하자.

OS_CORE.C	내부적으로 사용되는 공통 커널 함수들
OS_MBOX.C	MailBox 서비스 함수
OS_MEM.C	Memory 서비스 함수
OS_Q.C	Queue 서비스 함수
OS_SEM.C	Semaphore 서비스 함수
OS_TASK.C	Task 서비스 함수
OS_TIME.C	Time 서비스 함수
OS_FLAG.C	Flag 서비스 함수
OS_MUTEX.C	Mutex 서비스 함수
uCOS_II.C	위의 C파일들을 include한 파일
uCOS_II.H	커널 관련 header file

프로세서에 독립적인 커널 파일들

OS_CORE.C

uC/OS 서비스 함수들이 공통적으로 사용하는 함수들이 들어있다. 그래서 이 부분을 전부 이해한다면 나머지 서비스 함수들을 이해하는데 많은 도움이 된다.

uCOS_II.C

특별한 기능이나 내용은 없다. 다른 C파일들을 include 해서 이 파일들을 각각 컴파일하는 것이 아니고 uCOS_II.C 파일 하

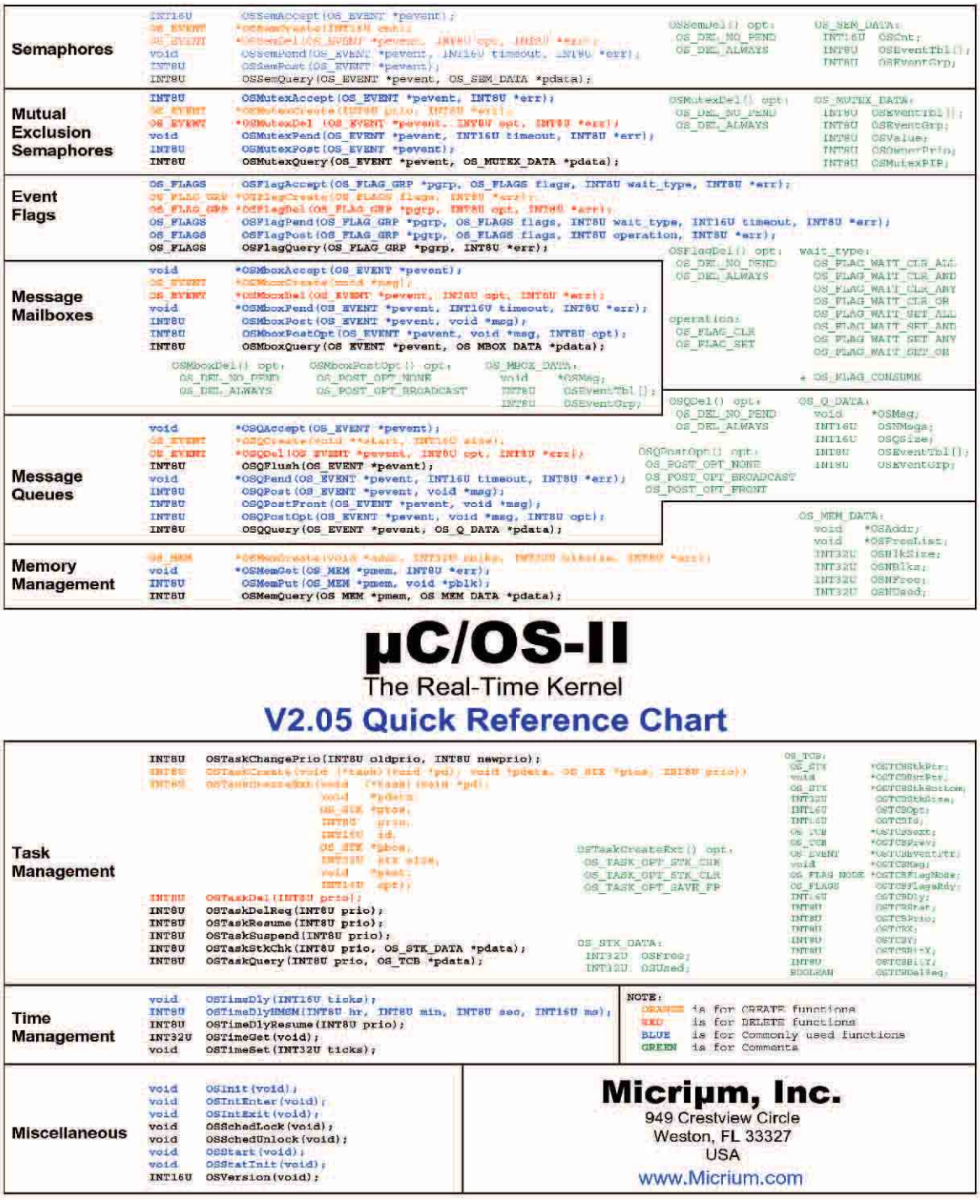

그림 4. uC/OS Quick Reference

나만 컴파일하면 모든 함수들이 컴파일되도록 해주는 파일이다. 결과적으로 여러 개의 obj 파일이 생성되지 않고, uCOS_II.OBJ 라는 커널 object 파일이 생성된다.

OS_CFG.H ····· OS 설정 파일
INCLUDES.H ····· 사용되는 header 파일들을 include

커널 설정 파일들

OS_CFG.H

일반적으로 임베디드 시스템에서 사용되는 OS들은 최적화를 위해서 OS를 설정할 수 있도록 해주는 기능을 가지고 있다. 그래서 필요없는 서비스를 제외시킴으로써 메모리 공간 절약이나 성능 향상과 같은 부수적인 효과를 가져 올 수 있다. uC/OS도 이와 같은 기능을 지원한다.

uC/OS의 함수들

uC/OS에는 많은 서비스 함수들이 존재한다. 그래서 필자가 여기서 직접 정리하기 보다는 **그림 4**와 같이 잘 정리된 문서를 참고하는 것이 훨씬 편리할 것이라 생각이 된다.

uC/OS 사이트에 접속하면 전체 함수들을 정리해둔 PDF가 있다. 이것을 참고하면 아직 uC/OS 함수에 익숙하지 않은 독자들에게도 체계적인 도움이 되리라 생각이 된다.

http://www.ucos-ii.com/contents/support/downloads/QuickRefChartV251-Color.PDF

```
void main (void)
{
  OSInit( );   ----------(1)
  PC_DOSSaveReturn( );
  PC_VectSet(uCOS, OSCtxSw); ---------(2)
  OSTaskCreate(TaskStart, (void *)0, (void *)&TaskStartStk
  [TASK_STK_SIZE - 1], 0); ----------(3)
  OSStart( );  ----------(4)
}

void TaskStart (void *data)
{
  OS_ENTER_CRITICAL( );  ----(1)
  PC_VectSet(0x08, OSTickISR);  ---(2)
  PC_SetTickRate(OS_TICKS_PER_SEC);  ---(3)
  OS_EXIT_CRITICAL( );   ---(4)
  for (;;)
  {
    printf("Task Run");
    OSTimeDly(100);
  }
}
```

프로그램 1. 100 tick마다 메시지가 출력되는 예제

uC/OS 기본 예제

uC/OS에는 기본적으로 PC용 예제가 4개 들어있다.

1) EX1_x86L: Task 생성과 세마포어를 이용한 공유 자원 예제.
2) EX2_x86L: Mailbox 활용 예제
3) EX3_x86L: Queue 활용 예제
4) EX4_x86L.FP: Floating point 활용 예제

한번씩 실행시켜보자!! 아마도 대부분의 독자들은 "이게 뭐야!!"하는 분들이 많으리라 생각이 된다(필자도 처음 RTOS를 접했을 당시 이것보다 더욱 열악한(?) 예제를 보면서 황당해 하던 기억이 난다. 이것이 도대체 뭘까??).

이 예제들을 통해서 RTOS의 기능과 성능을 평가하고 이해하기보다는 OS 함수들의 사용법을 익힌다는 생각을 갖는 것이 더 옳을 것 같다.

OS의 시작

uC/OS에 있는 예제를 참고해도 좋지만, 아직 익숙하지 않은 분들은 복잡하게만 여겨질 것이다. 그래서 좀더 간단한 예제를 준비했다. 이것으로 감을 잡으면서 시작을 해보도록 하자.

uC/OS는 main에서 시작이 된다. 아직까지는 OS가 동작되지 않고 있고, 여기서 OS를 시작하기 위한 준비를 한다.

Main 함수

(1) OS를 초기화 한다.
(2) OS 레벨의 Context Switching 함수를 Interrupt Vector에 등록한다. 이것은 x86의 Software

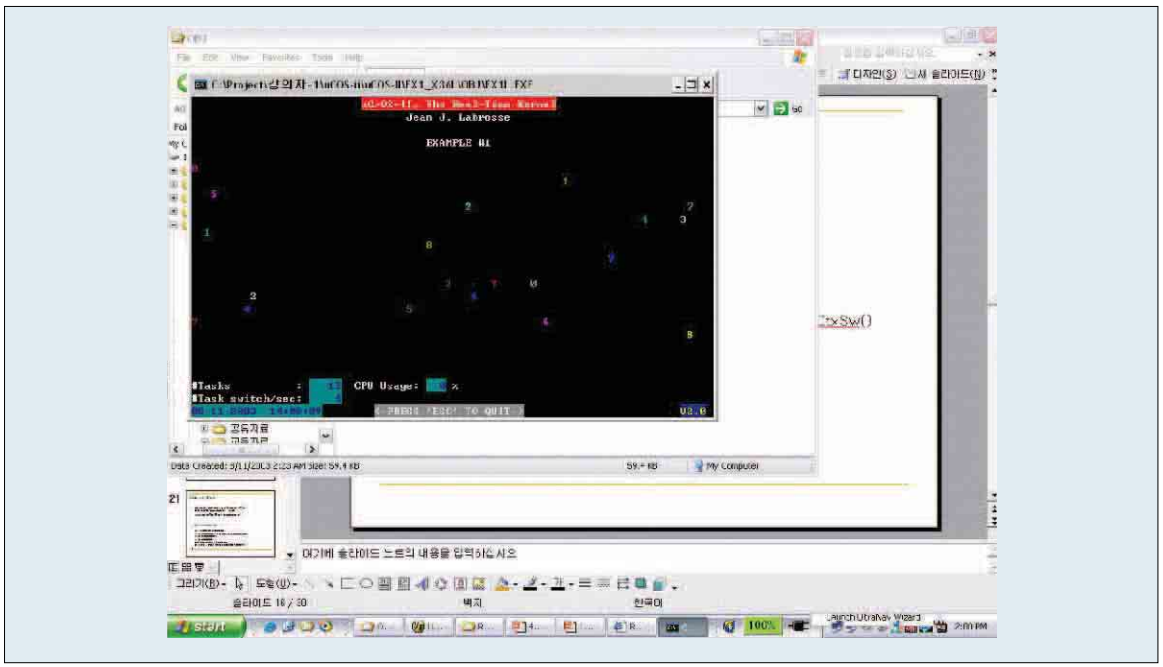

그림 5. EX1_x86L 실행 화면

Interrupt를 사용하여 context switching을 실행시키기 위한 준비이다.

(3) 최소 1개의 Task를 만든다. 여러 개의 Task를 만들어도 되지만 일반적인 경우라면 한 개의 task를 만들고, 이 task에서 시작하는 것이 보편적으로 편리하다.

(4) OS를 시작한다. OS가 시작이 되면, OS는 (3)에서 만든 TaskStart를 수행하게 된다. 만약 (3)의 과정에서 task를 여러 개를 만들었다면, 이 중에서 우선순위가 높은 task를 수행하게 된다. 그리고 영원히 main으로 돌아오지 않는다.

TaskStart 함수

(1) Critical Section으로 들어간다. (4)와 같이 사용되며, 이 두 문장으로 둘러싸인 블록 안에서 수행 중 일 때는 인터럽트나 인터럽트 레벨의 스케줄러도 동작하지 않는 상태가 된다. 시스템에 영향을 줄 수 있는 코드를 수행중 일 때 사용되면, 가능한 한 짧게 코딩을 해야 한다.

(2) 타이머 인터럽트 서비스 루틴을 등록한다. 타이머 인터럽트 서비스 루틴은 OS의 시간관리와 인터럽트 레벨의 스케줄러를 수행하는 심장과 같은 중요한 루틴이다. 만약, 이 부분이 설정이 되지 않으면 OS-TimeDly 함수와 Timeout 기능이 동작되지 않는다.

(3) (2)에서 사용할 타이머의 인터럽트 속도를 설정한다. 일반적으로 1초당 100-200회 정도로 인터럽트가 걸리게 하면 되지만, 애플리케이션에 따라서 알맞게 설정하면 된다. 한 주기를 Tick이라고 부르고 OS의 시간단위가 된다.

여기까지 왔으면 OS가 동작이 되는 것을 볼 수 있을 것이다. 앞에서 말한 것과 같이, 이 예제는 배포판을 기준으로 했으므로 특별히 더 수정을 하지 않았으면 1초에 2번씩 화면에 메시지가 찍히게 될 것이다.

Multitasking이 동작되는 과정이나 RTOS의 기능 확인은 아직까지는 힘들다. 이 예제는 단순히 uC/OS가 동작된다는 점을 확인하기 위한 예제이니 여기서 사용된 서비스 함수들이나 좀 더 많은 기능들은 다음 회에서 살펴보도록 하자. RTime