실시간 운영체제를 위한 시그널 처리 설계 및 구현*

°이재호, 편현범, 이철훈
충남대학교 컴퓨터공학과

Design and Implementation of Signal Handling For the Real-Time Operating System

Jae-Ho Lee°, Hyun-bum Pyun, Cheol-Hoon Lee
Dept. of Computer Engineering, Chungnam National Univ.
{jhlee, hbpyun, chlee}@ce.cnu.ac.kr

요 약 문

본 논문은 실시간 운영체제에서 프로세스들간의 비동기적 통신을 제공하기 위한 시그널 처리를 설계하고 구현 하였다. 일반적으로 실시간 운영체제에서는 태스크간 통신을 위해서 메시지 메일 박스와 큐를 사용하여 정보를 주고 받고, 태스크간의 동기화를 위하여 세마포어를 사용하는데 이들은 모두 특정 이벤트에 관계되어 이벤트를 발생시키거나(POST) 이벤트의 발생을 기다리는(PEND) 방식으로 동작한다. 본 논문에서는 필요한 루틴을 수행시키기 위해 특정 이벤트 동기화에 관계없이 처리가 가능하도록 실시간 운영체제에 적합한 시그널 처리 방법에 대해서 언급한다.

1. 서론

RTS(Real Time System)는 일반적으로 Embedded 시스템과 동일한 의미를 가지고 있으며, 이러한 시스템을 구동하기 위해 정확한 결과를 데드라인에 맞춰 출력할 수 있는 소프트웨어를 탑재하게 된다. 이러한 Embedded 시스템에 탑재된 각종 태스크들을 효율적으로 관리할 수 있는 운영체제가 필요한데 이것이 실시간 운영 체제이다. 최근 여러 분야에서 복잡한 기능을 수행하는 다양한 Embedded 시스템들이 개발되면서 이에 적합한 운영체제의 필요성이 더욱 커지고 있다.

본 논문에서는 CDMA 단말기를 위한 실시간 운영체제'를 개발하면서 실시간 운영 체제가 제공해야 할 태스크간의 통신 및 동기화 도구들 이외에 부가적으로 특정 이벤트에 관계없이 수행 가능하도록 구현된 시그널 처리 방안을 설계 및 구현하였다.[1]

본 논문은 2 장에서 실시간 운영체제의 기본 구조와 커널 서비스들을 간략히 언급하고, 특히 태스크간 통신을 위한 도구(ITC: Inter Task Communication) 및 동기화 도구에 대한 관련연구를 소개하였고, 3 장에서는 태스크간 비동기적 이벤트 처리를 위한 시그널 설계 및 구현 내용을 기술 하였으며, 4 장에서 결론 및 향후 연구 과제를 기술 하였다.

2. 관련연구

2.1 실시간 운영체제의 기본 구조

오늘날의 Embedded 시스템 응용이 다양함에도 불구하고,Embedded 시스템과 상위 응용 태스크들 사이에 존재하는 실시간 운영체제는 대부분 유사한 커널 구조와 기본적 서비스들을 제공하고 있으며, 본 논문에서 구현된 실시간 운영체제는 다음에 열거된 기능과 특징을 갖추고 있다.[1] [2]

- 태스크 관리 정책 - 태스크의 정적, 동적 생성 및 삭제
- 스케줄링 정책 - 태스크들의 우선순위를 기반으로 한 Round-Robin 을 지원하는 선점형 스케줄러
- 태스크간 통신 - 메시지 메일박스, 메시지 큐
- 동기화 및 상호 배제 - 세마포, 임계 영역 보호 제공
- 외부 이벤트 - 중요도에 따른 차별적 인터럽트 처리
- 메모리 관리 - 메모리의 정적, 동적 할당 및 해제
- 타이머 - 일정 시간 후 특정 루틴 수행
- 디바이스 드라이버 인터페이스(DDI)
- 시그널 - 태스크간 비동기적 이벤트 처리

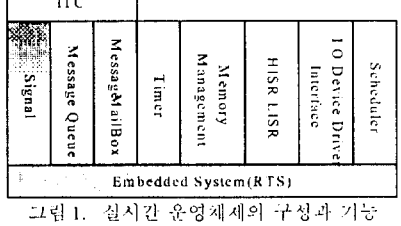

그림 1. 실시간 운영체제의 구성과 기능

2.2 태스크간의 통신 및 동기화 도구

태스크간 동기적 이벤트처리를 위해 다음과 같은 도구들이 있으며 이들은 특정 이벤트가 발생 될 때까지 수행 중이던 태스크를 suspend 상태로 만든다.[3]

- Semaphore
  세마포는 여러 태스크들간의 동작에 접근할 수 없는

*본 연구는 99년 한국전자통신연구소의 위탁과제로 개발된 CDMA용 실시간 운영체제를 기반으로 한다.

영역에 존재하는 공유 자원들을 획득하여 충돌 없이 안전하게 사용하기 위한 기술이다. 어떤 공유 리소스를 사용하고자 하는 태스크는 먼저 리소스에 해당하는 세마포를 요청하게 되고 리소스를 획득하게 되면 세마포 카운트를 1 씩 감소 시킨다. 이때, 만약 세마포 카운터가 0 이 되면 더 이상 자원을 획득 할 수 없기 때문에 해당 리소스가 가용 하게 되거나 타임-아웃될 때 까지 이벤트를 기다리는 상태로 진이하게 된다.

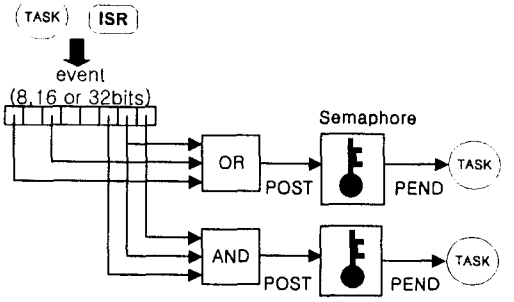

그림 2. 동기화 이벤트를 위한 세마포 사용

- Mail Boxes

  메일박스는 하나의 태스크나 ISR 에서 다른 태스크로 하나의 메시지를 전송하는 방식이다.

- Message Queues

  메시지 큐는 하나의 태스크나 ISR 에서 다른 태스크로 여러 개의 메시지를 전송하는 방식으로 메일박스가 여러 개 모여있는 형태이다.

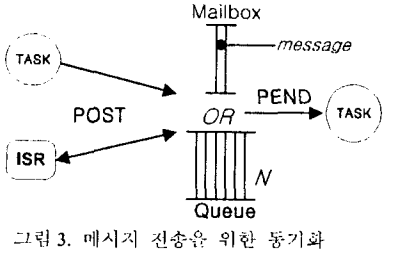

그림 3. 메시지 전송을 위한 동기화

3. 시그널 처리 설계 및 구현

3.1 시그널의 기본개념

시그널은 일반적으로 유닉스 시스템에서 프로세스간 통신을 하는 가장 오래된 방법중의 하나로 하나 이상의 프로세스들에게 비동기적인 이벤트를 알리기 위해 사용된다.[4] 앞서 언급한 세마포, 메일박스, 큐와 같은 태스크간의 통신 도구들은 메시지 또는 세마포를 획득 할 때까지 태스크가 suspend 된 상태에서 특정 이벤트가 발생하기를 기다려야만 하는 반면에, 태스크가 시그널을 통해 특정 시비스를 요청하면 시그널을 받은 태스크는 이벤트 플래그 변화에 관계없이 CPU 를 점유하이 실행권한의 자격이 주어졌을 때 해당 시그널에 대한 처리 루틴을 수행한다. 즉, 수행 중이던 태스크가 시그널을 받으면 바로 처리가 가능하고, 다른 태스크가 CPU 를 점유 있을 때 시그널을 받으면, Ready 상태에서 자신의 실행 권한이 오기를 기다리다가 최우선 순위가 되어 실행권한을 얻었을 때 받았던 시그널을 처리할 수 있다. 이는 동기화 이벤트 도구와는 달리 시그널이 비동기적 이벤트를 제공함을 의미하는 것이다. 이와 같은 특성 이외에 시그널은 한번 받은 시그널을 처리하기 전에 다시 보내진 시그널을 무시한다.즉, 태스크는 어떤 시그널을 받았는지 만을 기억할 뿐, 어떤 시그널이 얼마나 보내졌는지는 기억하지 못한다.

3.2 구현 내용

시그널을 처리하기 위해서 기존에 설계 되었던 태스크 제어 블록 구조에서 <표 1>과 같은 추가적인 필드를 두었다.

표 1.시그널을 처리하기 위한 TCB 구조의 추가 필드

```
Typedef struct os_tcb {
        void            *OSTCBStkPtr;
        void            *OSTCBSavedStkPtr;
        uint            OSTCBStat;
        uint            OSTCBSavedStat;
        /*          중간 생략              */
        uint                    OSTCBSignal;
        uint                    OSTCBEnableSignal;
        uint                    OSTCBActiveSignal;
        void                    (*signal_handler)(uint);
        struct os_tcb           *OSTCBNext;
        struct os_tcb           *OSTCBPrev;
        struct os_tcb           *OSSuspendListNext;
        struct os_tcb           *OSSuspendListPrev;
        OS_EVENT                *OSTCBEventPtr;
} OS_TCB;
```

OSTCBSavedStkPtr 필드는 시그널이 발생했을 때 이를 처리하기 위하여 태스크의 로컬 스택을 사용하므로, 시그널을 받은 태스크가 이전의 작업들을 계속 수행할 수 있도록 문맥의 context 를 가리키는 스택의 포인터 값을 저장하여 시그널 처리를 마친 후 다시 수행 중이었던 작업을 계속할 수 있도록 한다. *OSTCBSavedStat* 필드는 시그널을 받았을 때 수행 중이던 태스크의 상태를 저장 하기 위해 사용된다. 일반적으로 시그널은 수행 중인 태스크에 의해 자기 자신 또는 다른 태스크로 보내지는데, 시그널을 받은 태스크의 상태가 ready 가 아닌 suspend 되어 있을 경우 시그널을 처리하고 나서 시그널을 받기 이전의 원래 상태로 복귀 시키기 위해서 상태를 저장한다. *OSTCBSignal* 필드와 *OSTCBEnableSignal* 필드는 쌍으로 사용하는데 *OSTCBSignal* 은 32bit 변수로서, 각 비트가 하나의 시그널을 나타내므로 최대 32 개의 시그널을 구성할 수 있으며, 시그널을 받으면 해당 비트를 1 로 setting 함으로서 시그널이 도착하였음을 표시하는 역할을 한다. *OSTCBEnableSignal* 은 32bit 변수로서, 만약 어떤 비트가 1 로 setting 되어 있으면 이전 등록된 해당 시그널을 enable 시키는 것이고, 어떤 비트가 0 으로 되어 있다면 그 비트에 해당하는 시그널은 무시된다. *OSTCBActiveSignal* 필드는 현재 태스크가 시그널을 처리 하고 있음을 나타내고, 만약 1 로 setting 되어 있을 때는 다른 시그널에 관련된 작업을 수행하지 않도록 한다. *(*signal_handler)(uint)*필드는 시그널을 받았을 때 실제로 시그널을 처리하는 함수의 주소를 등록한다.

한편, 태스크가 suspend 상태에서 특정 이벤트를 기다리고 있을 때, 시그널이 도착하면 이에 대한 처리를 하기 위해 *OSTCBSavedStat* 필드에 현재의 태스크 상태가 Suspend 상태였음을 저장하고 시그널 처리를 위해 Ready 상태로 가서 스케줄링 대상이 되며, 최우선 순위가 되었을 때 CPU 를 점유하여 스택에 쌓아두었던 시그널 처리 루틴을 수행하게 된다. 그런데 이 과정에서 태스크가 Ready 상태에서 CPU 실행권한을 기다리면서 바로 이전 suspend 상태에서 기다리고 있던 이벤트가 발생하면 이를 처리 할 수 있도록 해당 이벤트에 Pending 되어 있는 태스크들의 리스트를 유지하여 시그널 처리 및 이벤트의 처리를 동시에 고려해 주어야 하는데 이를 위해 OSSuspendListPrev(Next) 필드를 두어 Ready 상태에서 스케줄링을 되기를 기다리면서 이벤트처리로 가능하도록 하였다.

시그널은 현재 수행중인 태스크에 의해서 자기 자신에게 보내거나, Ready,Suspend 또는 Delayed 상태에 있는 다른 태스크들에게 보내진다.Running 상태에 있는 태스크,즉 수행중인 태스크가 자신에게 보내진 시그널을 수행중인 작업을 잠시 중단

하고 받은 시그널을 바로 처리하며, 다른 상태에 있는 태스크에게 보내진 시그널은 태스크의 로컬 스택에 원래 수행해야 할 context 위에 시그널 처리를 위한 context 를 쌓아주고 원래의 스택포인터를 시그널 처리를 위한 곳으로 스택 포인터를 바꾸어 준다. 이때 시그널을 받은 태스크의 상태가 Ready 였다면 CPU 실행권한을 얻었을 때 시그널을 처리하고,Delayed 또는 Suspend 상태에서 시그널을 받으면 현재 자기가 속한 상태를 저장하고 시그널 처리를 위해 Ready 상태로 바꿔주어 스케쥴링 대상에 포함시킨다.

특히 이때 Suspend 상태의 태스크를 Ready 상태로 바꿀 경우에는 시그널처리를 위한 대기 상태에서도 특정 이벤트가 발생되었을 때는 이에 대한 처리가 가능하도록 해야 하므로 태스크가 Suspend 리스트와 Ready 리스트 양쪽에 모두 존재하는 것처럼 구현하여 동기화 이벤트 처리와 시그널 처리 가 모두 해결 가능하도록 하였다.

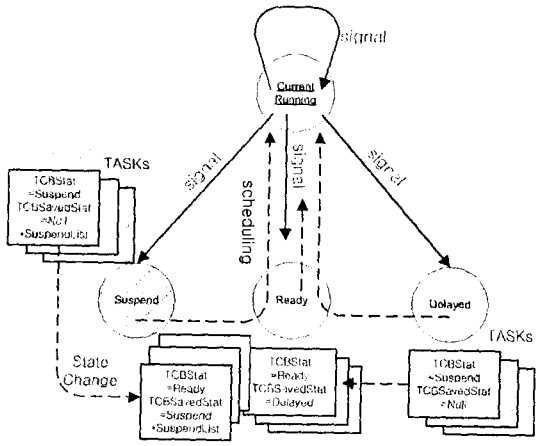

그림 4. 시그널 처리를 위한 태스크의 상태 변화

시그널 처리를 위해 구현한 사용자 서비스 모듈에는 태스크가 시그널을 받았을 때 이를 처리하기 위한 handling routine 을 등록하는 모듈(*RegisterSignalHandler*), 태스크가 특정 시그널에 대해 무시 하거나, 또는 무시된 시그널에 대해서 다시 처리 할 수 있도록 해 주는 모듈(*ControlSignal*), 시그널을 다른 태스크에 보내는 모듈(*SendSignal*), 어떤 시그널이 도착 했는지를 확인하는 모듈(*ReceivedSignal*), 사용자가 실제로 호출 할 수는 없으나 시그널이 발생했을 때 커널이 처리 가능한 시그널을 감사하여 해당 handling routine 을 호출 하도록 만들어진 모듈(*DispatchSignal*)이 있으며 다음과 같이 사용된다.

표 2. 시그널 사용 예

```
void Task1(void) /* This task send signal to Task2 */
{
        while(1) {
                OSTimeDly(20);
                SendSignal(&Task2, 1);
        }
}

void SignalHandler()
{
        /* This is signal handling routine */
}

void Task2(void) /* This task receive signal from Task1 */
{
        ControlSignal(1);
        RegisterSignalHandler(SigHandler);
        while(1) {
                OSTimeDly(500);
        }
}
```

Task1 은 SendSignal 함수를 이용하여 Task2 에 1 번 시그널을 보내면 보내면, Task2 에서는 시그널 처리를 위해 자신의 로컬 스택에 시그널 처리를 위한 context 를 쌓아두고, RegisterSignalHanler 함수를 이용해 실제 수행해야 할 함수를 등록하고, ControlSingnal 함수를 이용하여 수행하고자 하는 시그널의 해당 비트를 set 시키면, 커널에서 DispatchSignal 루틴을 실행하여 실제 시그널 처리가 이루어 지도록 구현했다.

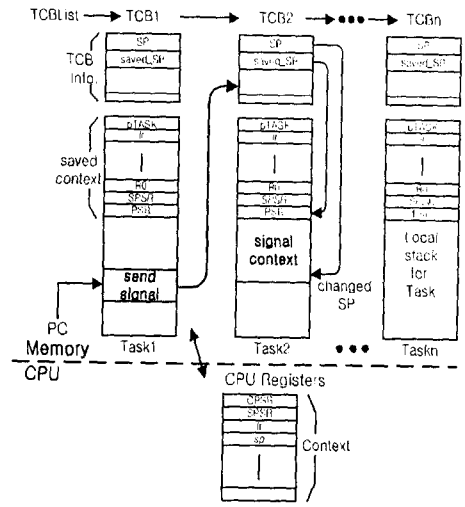

그림 5. 태스크간 비동기적 이벤트 처리

4. 결론 및 향후 연구 과제

본 논문에서는 실시간 운영 체제를 위한 태스크간 비동기적 이벤트들을 처리하기 위한 시그널에 관련된 모듈을 설계하고 구현하였다. 개발된 실시간 운영체제는 ARM7TDI CPU 를 탑재하고 있는 PID7T 평가 보드에 porting 되어 검증 되었으며, 현재는 Intel 의 StrognArm1100 CPU 를 탑재하고 있는 GPS 수신기 보드에 porting 중에 있다. 차후 이들 외의 다양한 CPU 를 탑재한 여러 가지 환경에 응용하여 운영체제의 정확성 및 신뢰성의 향상을 위한 성능 평가가 수반 되어야 할 것이다. 이러한 성능 평가를 위해 실시간 운영체제의 여러가지 성능 평가 요소들(Context Switching Time, Interrupt Response, Determinism 등)을 정의하고 각각의 CPU 의 상에서 정확한 벤치마킹이 이루어져야 할 것이다.

5. 참고문헌

[1] 류석,편현범,이재호,윤기현,이재경,이철훈,"CDMA 시스템용 실시간 운영체제의 설계 및 구현", 한국정보과학회 학술발표논문집 Vol.26.No.2, pp140~142

[2] Klein Embedded Systems Conference West at the San Jose Convention Center,San Jose, CA on September 29-October 23,1997.

[3] Jean J.Labrosse, "μC/OS The Real-Time Kernel", R&D Publications Lawrence,Kansas 66046

[4] David A Rusling , "The Linux Kernel(verion 0.8-3)",1998

[5] Accelerated Technology, Inc "Nucleus PLUS Internal Manual", 1996.

[6] Advanced RISC Machine Ltd(ARM), "ARM Software development Toolkit User Guide", 1996.