휴대 임베디드 시스템 커널 구현

유 진 호(兪 眞 鎬)
한국전자통신연구원 멀티미디어 연구부
전화 : (042) 860-6610 / 팩스 : (042) 860-6671

Implementation of A Handheld Embedded Kernel

Jin Ho Yoo

Multimedia Technology Department
ETRI
E-mail : yoojh@etri.re.kr

Abstract

In this paper, we implement and construct a kernel on handheld systems. The goal of this project is to develop issues related to the development of small devices: embedded kernel, power management, user interface issues, networking, and the development of applications for small devices. We explain basic system configuration, kernel activity, device drivers and developing environment in this paper. We also explain detail scheduler activity.

I. 서론

최근의 일반적인 목적의 컴퓨터가 놀랄만한 속도로 발전되고 있고 이에따라 사무에서부터 공장에 이르기까지 많은 응용이 이루어지고 있다. 이러한 컴퓨터 기술의 발달은 입을 수 있는 컴퓨터로 까지 발전하고 있으며 원거리에서 무선 데이터 네트워크를 사용하여 편리하게 정보를 공유하고 그 정보를 사용하여 여러 가지 일을 할수있도록 구성이 되고 있다. 이에 대해 컴퓨터도 노트북부터 랩탑, 팜탑까지 그 크기를 달리하여 사용자 업무와 사용편의에 맞게 발달하기에 이르렀다. 이러한 컴퓨터 시스템에 대한 요구로 인해 만들어진 하드웨어 상에 임베디드 시스템으로의 구성은 지금도 그 구현이 많이 이루어지고 있다. 본 논문에서는 이러한 하드웨어 시스템 구성에 맞도록 시스템 소프트웨어를 커널 레벨에서 어떻게 구성, 구현할 것인가의 문제를 다룬다. 이와 함께 이러한 커널이 가져야할 기능이나 그 구현시 어떤 부분으로 구성되는가를 알아본다. 특히 본 논문에서는 휴대를 위한 컴퓨터의 하드웨어 위에 올라가는 시스템 소프트웨어인 커널과 그 구성요소들에 대해 알아본다.

II. 기본 시스템 구성

기본적인 시스템 소프트웨어 구성은 다음과 같다. 예외처리, 인터럽트, 트랩, C run time 환경(crt0.s), VBR, 스택포인터, 타이머, BSS의 초기화, 롬영역의 데이터를 램으로 옮기는 부분, 시스템 초기 레지스터 세팅 등으로 구성이 된다. 이러한 구성요소의 일련의 세팅을 구성으로 하여 시스템의 초기화는 이루어진다. 이러한 초기화 후에 BSP부분의 프로그램이 구성된다. 그리고 이런 프로세스 다음으로 커널의 멀티타스크 운영을 위한 모듈이 올라가고 타스크 간의 통신을 위한 모듈들이 구성된다. 이러한 프로그램 기능구현의 편의를 위해 편의함수에 해당하는 커널 라이브러리들이 같이 제공되어 컴파일이 된다. 각각의 다바이스사용을

위한 디바이스 드라이버를 위한 테이블이 커널 내에 삽입이 되고 이러한 디바이스테이블을 통해 디바이스들은 드라이버 함수를 구성하며 응용프로그램은 이들 디바이스 드라이버 함수를 통해 디바이스를 초기화하고 읽고/쓰고, 사용할 수 있다. 예외처리의 구성은 칩 스펙에 나온 것을 근거로 하여 테이블형태로 구성되는데, 여기에는 재설정, 잘못 할당되는 접근, 일반적인 접근 오류, 0으로 나누는 것, 특권화된 명령 위반, 잘못된 명령, 브레이크 포인트, 트레이스, 인터럽트의 인지, 트랩 등과 예약된 부분들로 구성이 된다. 이 예외처리기능은 재설정 기능부터 소프트웨어로 구성이된다. 벡터의 구성 후에 그 벡터가 저장된 주소를 VBR(Vector Base Address)에 저장함으로써 시스템의 예외상황이 발생했을 때, 해당 벡터로 분기하게 된다. 인터럽트는 여러 인터럽트 중 어떤 우선순위에 의해 분기를 하는데 타이머나 DMA 등이 그런 것에 속한다. 타이머는 멀티 프로세스 시스템에서 프리미티브로서 쓰인다. 인터럽트는 하드웨어와 그 위의 시스템 소프트웨어 간의 자원사용을 위한 가장 기본적인 인터페이스를 제공한다. 운영체제 상에서 상호배제등의 알고리즘을 통해 임계구역을 설정할 수가 있으나 이보다 더 기본적인 프리미티브로써 트랩이 제공되고 어떤 소프트웨어적인 메카니즘보다 더 적절하게 수행될 수 있다. 시스템을 멈춘 상태에서 무슨일을 하고자할 때 사용되는데 기본적으로 타스크 교환에서 주로 사용되는 메카니즘이 트랩이다. C run time 환경을 위해 제공되는 부분은 주로 처음에 VBR의 값을 정하고 스택포인터, BSS의 초기화, 데이터 롬을 램으로 옮기는 작업 등을 한다. 스택포인터는 보통 시스템 커널 스택과 응용프로그램 스택으로 구성할 수 있다. 이 스택의 구성은 커널설계자가 정할 수 있다. 안정된 선택을 위해서는 스택을 분리해서 유지하는 것이 좋다. 그리고 처음에 시스템이 시작할 때 초기화되지 않은 BSS부분은 0으로 초기화를 해주고, 데이터 롬에 있는 데이타는 수정되어야 하므로 램으로 복사를 하게된다. BSP(Board Support Package)는 하드웨어 구성의 초기화를 요구하는 부분이나 레지스터, 큐등을 초기화하는 부분으로 어떤 하드웨어가 장착되느냐에 따라 다를 수 있는 보드 의존적인 부분이다. 그리고 CPU의 효율적인 사용을 위해 시스템은 주로 멀티타스크 모델로 구현이 된다. 이를 위해 타스크 스택이 구성되고 이것은 PC, PSR, 모든 레지스터 등의 정보를 가지는 요소로 구성이 된다. 또한 타스크의 포크를 지원하기 위한 포크블록도 제공이 되며, 타스크간의 통신을 위한 함수들로 구성이 된다. 이러한 함수는 임계구역으로 설정되는 임의의 공간을 통한 동기/비동기 통신을 하게된다. 이러한 프로세스간

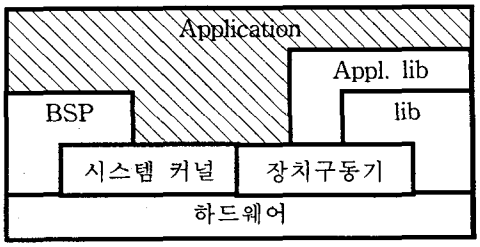

그림1 시스템 스택

의 통신은 메시지 큐, 메일박스, 이벤트 큐등으로 그특성을 갖는다. 메시지 큐나 메일박스는 프로세스간의 데이터 통신을 위해 지원이 되고 이벤트 플래그 등은 시간제약이 많이 따르는 인터럽트 등의 루틴과 통신을 위해 플래그정도의 데이터 교환을 위한 자원으로써 제공된다. 이러한 구현을 기본적으로 뒷받침하는 세마포어 등이 제공되는데, 세마포어와 같은 프리미티브는 인터럽트 등의 발생으로 인한 개입을 막는 방법으로 지원이 된다. 이러한 소극적인 알고리즘에 의해 세마포어는 지원이 된다. 세마포어를 위해 특정한 명령세트를 준비해 두기도 한다. 이 명령세트의 기능은 발생한 인터럽트를 지연시킴으로써 인터럽트의 개입을 막는 방법을 주로 사용한다.

III. 스케줄러

스케줄러는 멀티타스크 시스템에서 콘트롤을 분배하는 정책과 각 타스크에 콘드롤을 주는 두가지 기능을 가진다. 타스크에 콘드롤을 할당하는 스케줄러는 그 정책에 따라 문맥교환을 수행하는 아주 중요한 부분이다. 클럭이 발생할 때마다 우선순위를 고려하여 데이터 구조상의 문맥교환이 이루어지고 스케줄러는 이러한 문맥교환을 클럭틱이 발생했을 때 수행한다. 문맥교환 모듈은 클럭틱마다 호출되는 모듈이므로 아주 간결해야하고 그 수행을 빨리 할 수 있어야 한다. 스케줄러는 그 빠르고 간섭이 없는 수행을 위해서 주로 트랩에 의해서 분기가 일어난다. 타스크의 수행 여부는 시스템의 스케줄링 정책에 의해서 결정된다. 시스템은 우선순위를 가지고 그 순위에 따라 수행이 되며, 그러한 우선순위는 스케줄링정책을 반영한다. 이 시스템에서 스케줄링정책은 Rate Monotonic 정책을 사용한다. 우선순위와 비례하여 프로세서를 할당받는 시간이 정해진다. 우선순위가 높은 타스크일수록 그 수행시간이 길어지고 우선순위가 낮을수록 수행되는 시간이 작게 결정이 된다. 그러므로 우선순위가 높을수록 프로세서

를 오랜시간 할당받아서 긴시간 동안 처리가 이루어지므로 우선순위가 높을수록 빠른 수행결과를 얻을수가 있다. 스케줄러의 우선순위할당이나 다음 우선순위의 타스크를 찾는 것은 빠른 수행을 위해 그 자료구조와 탐색 알고리즘을 제공해야 한다. 다음 우선순위의 타스크를 빨리 찾을 수 있는 자료구조와 알고리즘이 있어야 한다. 스케줄러의 행동은 다음과 같이 진행된다. 포크큐에 어떤 블록이 있으면 그 큐에 있는 모든 리스트들을 수행한다. 그 다음 현재의 타스크의 상태를 검사한다. 현재의 타스크가 활동상태(TCBST_ACTIVE)이고 대기 우선순위 큐(tmg_rdyprio)보다 그 우선 순위가 높으면, 원래의 타스크를 수행을 계속한다. 그러나 만약, 현재의 타스크가 활동상태(TMS_ACTIVE)이고 대기 우선순위 큐(tmg_rdyprio)보다 그 우선 순위가 낮으면, 원래의 타스크를 준비상태(TCBST_READY)로 만들어야 한다. 그리고 난 후 현재의 타스크를 대기큐(tmg_rdyqueue[])에 준비상태로 집어넣고 실행될 타스크로 지정된 대기큐에있는 타스크를 현재 수행될 타스크로 지정한다. 대기큐가 비어있지않으면 우리는 단순히 헤드노드에 삽입을 한다. 그렇지않다면, 우리는 반드시 우선순위 테이블을 갱신시켜 주어야 한다. 그 다음으로 현재 대기큐의 우선순위가 0인지 비교한다. 만약 0이라면 이벤트가 발생했는지(tmg_event == 1)를 확인해야 한다. 확인 후 유휴상태인지 아닌지를 표시해야 한다. 만약, 현재 대기큐의 우선순위가 0이 아니라면, 현재의 우선순위를 가지는 대기큐에 있는 타스크를 수행 타스크로 지정한다. 수행 타스크로 지정된 타스크외에 다른 타스크가 있으면 그 타스크를 대기큐의 헤더로 옮긴다. 만약 같은 우선순위의 타스크가 자신 혼자이면, 그 타스크를 옮긴후 아무것도 없으므로 대기큐의 우선순위 테이블을 새로 계산해야 한다. 여기서 우선순위가 높은 것을 찾아내기 위해 좀더 빨리 찾을 수 있는 방법을 찾아야 하는데 여기서는 계산에 의한 연산의 수를 줄이기 위해 미리 계산된 배열(tmk_log2[])을 사용한다. 이 배열은 그 인덱스의 값이 주어졌을 때, 인덱스값의 가장 왼쪽에 위치한 1로 된 비트의 순번을 그 배열의 값으로 가진다. 이렇게 해서 쉬프트 등의 연산 오버헤드없이 몇번째 비트가 1로 세트되어 있나를 알수가 있다. 이러한 검사는 대기큐에 어떤 타스크가 기다리고 있는지의 세트정보를 손쉽게 알수있도록 해준다.

IV. 디바이스의 연결

디바이스는 하드웨어 메모리주소의 연결선상(memory-mapped I/O)으로 접근한다. 보통 디바이스는 읽고, 쓰고, 상태를 보고 할 수 있는 환경으로 제공된다. 입력출력이 필요한 포트와 그 상태를 알 수 있는 레지스터등으로 구성된다. 커널은 디바이스의 수행을 위한 함수 테이블을 가지고 있고 그런 테이블의 형태로 디바이스 함수를 정의한다음 등록을 시키고 그 디바이스를 구동시킬수가 있다. 이러한 함수의 내용은 주로 일정주소공간을 읽는 함수, 일정주소공간에 쓰는 함수, 레지스터 세팅 등을 통해 초기화하고 타스크에 의해서 이용된다. 이런 디바이스 드라이버는 두가지 방식으로 구성될 수가 있다. 첫째는 화일시스템을 접근하는 것처럼 디바이스 리스트 디렉토리 내에 두어 접근하면서 쓰는 것이고 또 하나는 폴링방식을 이용하여 살아있는 타스크가 일정 주기대로 수행이 될 때, 그 주기에 디바이스의 상태를 검사하여 수행이 이루어지도록 하는 것이다. 화일시스템방식을 쓰는 것은 사용자가 사운드나 여러가지 필요한 디바이스들, 즉, 사용자 편의에 의해서 사용되는 디바이스에 접근하기 위해서 사용할 때가 많다. 폴링방식을 이용하는 경우는 입력이 들어와서 읽어 주기를 기다린다거나 하는 이벤트방식의 디바이스를 다룰 때 주로 사용한다. 이런 이벤트방식의 디바이스는 주로 인터럽트를 사용하는 경우가 많다. 하지만, 인터럽트라는 자원을 쓸만한 것이 아니거나 그런 신속한 감지가 없어도 되는 경우에 사용할 수가 있다. 그리고 타스크자체도 충분히 빠르게 동작하기 때문에 그런 이벤트 상태의 감지를 충분히 할 수가 있다. 보통 이런 경우 이벤트 플래그라는 대기큐가 허용되는 시스템 플래그를 두어 인터럽트 발생시 이벤트 플래그를 세팅을 하고 이벤튜 플래그를 감시하는 타스크를 두어 상태를 감시하게 되는 경우가 보통이다. 이런 디바이스의 특성에 맞게 타스크의 우선순위가 구성이 되어 디바이스의 활동을 원활하게 지원해 주어야 한다.

V. 개발환경 - 타겟 모니터 디버거

어느정도의 코드가 완성되면 타겟을 구성한 후 그곳에 다운로드를 해서 테스트를 해봐야 한다. 타겟에 프로그램 오브젝트를 올려서 테스트를 하기 위해서는 타겟 프로그램과 호스트 상에 있는 디버거의 통신을 위한 인터페이스가 필요하다. 이러한 통신을 위해서 RS 232 등의 시리얼 인터페이스를 사용한다. 타겟에는 통신과 몇가지 기본적인 기능을 위한 스텁이 올라가고 호스트 쪽에는 이러한 인터페이스 기능을 사용할 간단한 시리얼 통신모듈이 필요하다. 호스트 컴퓨터에는 디버거기능을 수행하는 gdb등과 같은 디버거가 설치되어야 한다. 물론 이때 gdb는 타겟에 있는 프로세서의 명령어

세트를 만들어낸 것을 해석하고 인식할 수 있는 것이어야 한다. 이를 위해 프로세서 선택에 맞게 gdb를 다시 컴파일해야 한다. 그 프로세서의 명령어 세트를 생성해낼 수 있는 컴파일러로 디버그하고자하는 프로그램을 컴파일한후 이 컴파일된 오브젝트 파일을 디버거의 입력으로 하여 디버깅을 시작하게 된다. 이때 디버거로 입력한 모든 커맨드는 통신 모듈을 통해 타겟에 있는 스텁에 전달되어 타겟에서의 움직임을 통제하게 된다. 스텁은 다음과 같은 기능을 가진다. 첫째는 우선 부트모듈이어야 한다. 둘째는 다운로드기능, 셋째는 메모리 조사기능 즉, 메모리 읽고/쓰기 기능, breakpoint 설정 기능, 역어셈블기능 등을 가져야 한다. 이러한 기능을 가지고 디버거와 통신하면서 프로그램을 수행하면서 추적 해볼수가 있다. 그리고 프로그램 중에 잠깐 멈추게 한후 프로세서의 상태를 볼 수가 있어야 한다. 이러한 기능들이 제공되는 가운데 통신스텁은 간단하지만 아주 튼튼하게 구성이 되어야 한다. 이러한 스텁 모듈은 부분적으로 시스템 초기화모듈을 포함하고 있어야한다.

Ⅵ. 결론 및 추후 연구

본 논문에서는 구현된 임베디드 시스템의 스케줄러 구성과 그 개발환경을 알아보았다. 스케줄러는 단순하되 안전하고 완전하게 수행이 이루어져야 한다. 스케줄러는 타스크의 생성되고, 수행되고, 대기하고, 수행을 마치는 것을 관장한다. 수행을 마치기전까지 프로세서를 할당하고 대기시키고 하는 것을 효율적으로 함으로써 시스템의 효율을 높이고 그 기능을 향상시킨다. 스케줄러는 수행시간도 되도록 간결해야 하고 에러가 없어야 하고 오버헤드를 갖지 않아야 한다. 또한 프로세서의 계산 시간들을 모든 타스크에게 우선순위대로 공정하게 나누어 줄 수 있도록 스케줄이 될 수 있어야 한다. 이렇게 슬림화되면서 공정한 콘트롤을 타스크에게 나누어 줄 수 있어야 한다. 이러한 일련의 타스크의 수행이 보장될 때 이 타스크에 의해서 운영되는 리소스들은 효율적으로 사용될 수가 있다. 이런 예상되는 리소스의 사용을 위해 타스크들도 분류되어야 한다. 예를 들어 대화형 리소스이면 그에 맞는 타스크의 구조를 갖는다든가 이벤트의 처리과정이 효율적으로 구현될 수 있도록 그 구조를 가져야 할 것이다. 임베디드 시스템의 경우 프로그램이 타겟에 올라가기 때문에 개발환경은 아주 필수적인 것이다. 이러한 개발환경은 개발시간을 단축할 수 있을 뿐만아니라 오류정정을 바르게 할 수 있고 더 빨리 오류를 정정할 수 있게 하는 기능을 한다. 이를 위한 타겟모드 디버깅을 지원하기 위해 리모트 디버깅을 할 수 있는 환경을 구성해야 한다. 멀티 타스크 시스템의 경우 각각의 타스크별로 디버깅을 할 필요가 있다. 본 시스템에서는 타스크가 절대적인 주소공간에 존재하므로 브레이크 포인트를 설정함으로 해서 각각의 타스크를 디버깅할 수 있는 환경이다. 더 발전적인 디버거는 각각의 타스크별로 윈도우 소스창을 열어서 각각의 타스크별로 수행공간을 감시하면서 수행할 수 있으면 더없이 좋은 디버거 환경이 될 것이다. 또한 이러한 프로그램 디버거 환경의 편의는 프로그램을 만드는데 아주 중요하며, 수행공간이 타겟인 시스템에서는 아주 필수적인 것이다. 본 시스템에서는 타스크마다 절대 주소 공간을 확보함으로 해서 순차적인 디버거 수행이 가능하다. 태스크별로 윈도우 소스창을 연다든지, 오가는 메시지의 내용을 윈도우를 통해 확인한다든지 하는 환경을 개발하면 더 좋을 것이다. 본 시스템에서는 인터넷상에서 무료로 사용할 수 있는 gdb라는 디버거를 사용함으로 순차적인 디버깅을 한다. gdb의 통신스텁 뿐만아니라 gdb의 디버깅 모듈을 수정하는것에 의해 타스크별 디버거라든지 상태창을 더 가질 수 있으면 좋을 것이다.

참고문헌(또는 Reference)

[1] E.G. Coffman(ed.), "Computer and Job/Shop Scheduling Theory", Wiley, New Yorl, 1976.

[2] L. Sha et al, "Generalized Rate-monotonic Scheduling Theory: A Framework for Developing Real-Time Systems", Proceedings of the IEEE,(this issue).

[3] C. Hou and K. Shin, "Load Sharing with Considerations of Future Task Arrivals in Heterogeneous Distributed Real-Time Systems," Proceedings IEEE Real-Time Systems Symposium, December 1991.

[4] K. Ramamritham, J. Stankovic, and P. Shiah, "Efficient Scheduling Algorithms for Real-Time Multiprocessor Systems," IEEE Transactions on Parallel and Distributed Computing, Vol. 1, No.2, pp.184-194,April 1990.

[5] W. Horn, "Some Simple Scheduling Algorithm," Naval Research Logistics Quarterly, 21, pp.177-185, 1974.

[6] J. Schoeffler, "Distributed Computer systems for Industrial Process Control," IEEE Computer, Vol. 17, No.2, pp. 11-18, February 1984.