실시간 운영체제를 위한 프로세스의 효율적인 스케줄링 알고리즘

정선아[0] 이지영
세명대학교 전산정보학과
sunahori[0]@freechal.com, ljy409@semyung.ac.kr

Effective Scheduling Algorithm of Process for Real Time Operating System

Sun-Ah-Jeong[0] ,Jie-Young Lee
Dept. of computer information, Semyung University

요 약

본 논문은 실시간 운영체제에서 프로세스의 효율적인 관리를 위한 스케줄링 알고리즘을 제안한다. 따라서 CPU의 활용도를 높이고 스케줄링 시간과 인터럽트 시간을 줄임으로서 자원을 효율적으로 관리할 수 있다.

본 논문에서 제안하는 방법으로는 다중 큐에 PIT(Process Information Table)를 두어 각각의 큐에 프로세스가 들어오면 우선순위에 따라 CPU를 할당하는 방법이다. 기존의 다중 큐와는 달리 우선순위 프로세스를 보다 정확하고 빨리 찾아내어 외부 또는 내부의 인터럽트에 응답 할 수 있게 하였다. 또한 우선순위에 밀려 실행하지 못하는 프로세스는 일정 시간이 경과하면 CPU를 선점할 수 있다. 그러므로 CPU는 활용도가 높아지고 유휴 시간은 짧아지게 된다.

본 논문은 일반 펜티엄 PC에서 실험하였으며 현재 사용되는 RTOS(VxWorks, QNX)와 비교하여 다소 우수함을 보였다.

1. 서 론

현대 기술의 발달과 사용자의 요구사항은 더욱 복잡하고 시간에 대한 비중은 무엇보다도 중요하게 되었다. 휴대용 통신 기기의 발달로 어디서나 인터넷을 할 수 있으며 개인 정보와 금융서비스, 증권 서비스, 개인 Email 확인 등을 할 수 있게 되었다. 또한 집이나 사무실에서나 가능했던 일들을 기술의 발달로 인해 시간과 공간의 제약을 받지 않고 할수 있게 되었다.

실시간 시스템은 기존의 시스템과 달리 시스템 동작의 정확성이 논리적 정확성뿐만 아니라 시간적 정확성에도 좌우되는 시스템을 말한다[1-2]. 이러한 시스템의 구현은 프로세스의 작업에 좌우된다. 실시간 프로세스는 외부에서 발생하는 사건(event)에 매우 빨리 반응해야 하므로 스케줄러는 일반 사용자 프로세스와는 구분하여 처리한다. 프로세스는 주변 상황에 따라 상태를 변경한다[3]. 프로세스가 실행중이거나 언제든지 실행할 수 있는 준비단계 즉 시스템의 CPU 중 하나에 할당되는 것을 기다리고 있는 것이다. 프로세스가 이벤트나 자원이 할당되기를 기다리는 것이 대기(Waiting)상태이다. 대기상태에 있으면서 인터럽트를 허용하는 프로세스는 시그널에 의해 인터럽트 될 수 있고, 인터럽트가 금지된 대기상태의 프로세스는 하드웨어를 직접 기다리면서 어떤 사건에도 인터럽트 되지 않는다. 인터럽트를 허용하지 않는 시스템은 실시간 시스템으로 부적합하게 된다. 즉 외부 환경에 즉시 대응할 수 있어야 실시간 시스템으로서의 기능을 발휘할 수 있는 것이다. 그리고 기술의 발달로 실시간 운영체제는 우리 생활에 필수품이 될 휴대용 전자제품과 밀접한 관계를 가지고 있기 때문에 개발에 필요성이 절실하다[4-5].

본 논문에서는 실시간 다중 큐(Real Time Multi Queue)를 이용한 프로세스 스케줄링 알고리즘을 적용하였다. 실험 결과 기존의 스케줄링 알고리즘과 비교하여 프로세스들이 CPU 활용도에서 우수함을 보였다.

2. 스케줄링

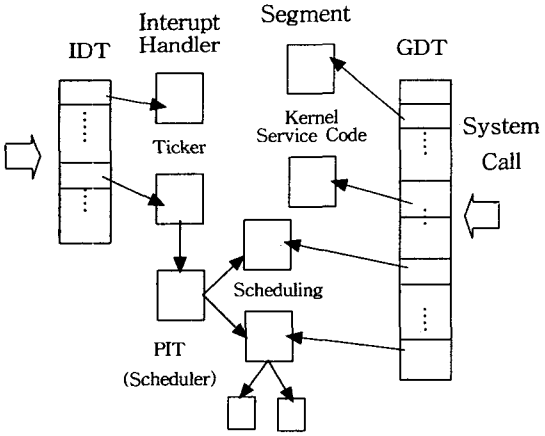
스케줄링 방식으로는 선점 스케줄링(Preemptive scheduling)과 비선점 스케줄링(Non-preemptive scheduling)으로 나눌 수 있다. 전자는 우선순위가 높은 프로세스 즉 테스크가 실행될 준비가 되면 CPU의 선점권을 즉시 받게 된다. 이것은 우선적으로 처리해야할 작업이 있다는 것을 의미하고, 실시간 시스템에서는 이러한 방법을 사용한다. 후자는 하나의 테스크가 CPU를 선점하면 다른 테스크는 현재 CPU를 선점하고 있는 테스크의 작업이 끝날 때까지 기다려야한다. 일반적으로 프로세스의 상태는 몇 단계로 변하게된다. 프로세스가 CPU를 점유하고 있으면 실행(Running)상태가 되고, 현재 실행중은 아니지만 언제든지 CPU를 사용할 수 있는 상태이면 준비(Ready)상태에 있다. 만일 프로세스가 어떤 사건이 발생하기를 기다리고 있는 상태이면 봉쇄(Block) 상태가 되는 것이다. 스케줄링 기법으로는 대표적으로 라운드로빈 스케줄링(Round-Robin)스케줄링과 FIFO 스케줄링이 있다. 라운드로빈 스케줄링 기법은 프로세스가 중앙처리장치에서 배당된 시간이 끝날 때까지 작업을 끝내지 못하면 중앙처리장치는 다음 대기중인 프로세스에게 넘어가고 현재 대기 중이던 프로세스는 CPU를 할당받기 위해서 있는 프로세스의 끝으로 가게된다. 라운드로빈 방법은 대화식 사용자들에게 적절한 응답시간을 보장해 주어야하는 시분할시스템에 효과적이다. 그

리고 FIFO 스케줄링은 프로세스들이 대기 Q(Queue)에서 프로세스들이 대기 큐에 들어오는 순서대로 CPU를 사용하게 된다. 즉 FIFO 방식은 비선점 스케줄링 방법으로 대화식 사용자들에게는 부적합하다. 하지만 본 논문에서는 다중 큐를 이용하여 프로세스들에게 우선순위를 부여하여 우선 순위가 높은 프로세스를 먼저 처리하게 설계하였다.

스케줄링 알고리즘은 각각의 프로세스에게 공정하게 CPU사용 시간을 할당해야하고, CPU는 할당된 시간의 100%를 사용하도록 해야 한다. 그리고 상호 작용하는 사용자에 대해 응답시간을 최소화해야하고, 일괄처리 사용자들이 출력을 기다려야하는 시간을 최소화해야하며 시간당 처리되는 작업의 수를 최대화한다.

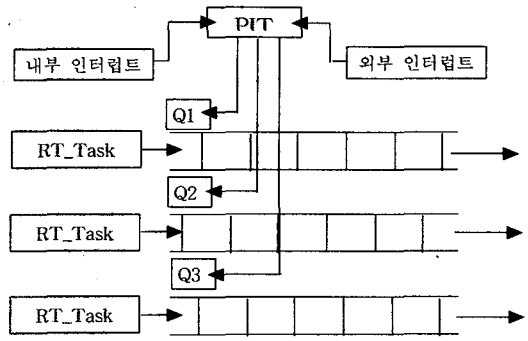
3. 본 논문에서 제안하는 스케줄링 알고리즘

제안하는 알고리즘의 핵심은 다중 큐에서 실행 가능한 프로세스들의 정보(Process Information)를 갖는 테이블을 작성하여 테스크 구조에 유지한다. 프로세스 정보 테이블(PIT)는 우선 순위에 따라 스케줄링 하지만 다른 프로세스들 보다 좀더 오래 CPU를 할당받기를 기다리는 프로세스를 위하여 CPU활용도를 측정하여 어느 기간이 지나면 우선 순위가 낮은 프로세스를 실행하게 하였다. [그림 1]은 실시간 운영체제에서 외부 또는 내부의 인터럽트로부터 시스템이 실행되는 제어 흐름도이다.



[그림 1] RTOS의 제어 흐름도

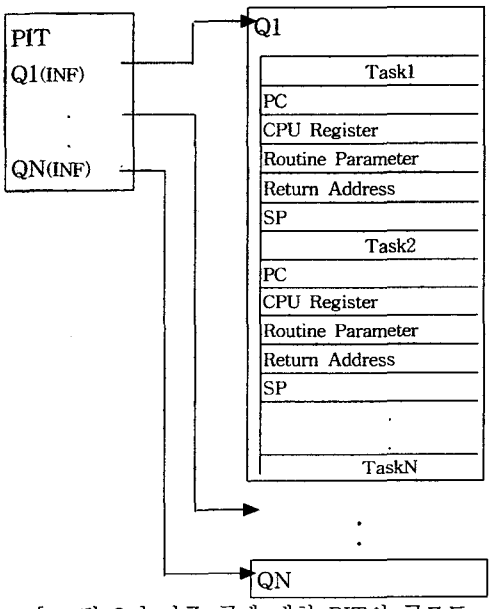
인터럽트 표시테이블(Interrupt Descriptor Table)는 내부에서 인터럽트가 발생된 정보를 저장하고 있다. 그리고 일반적인 시스템 콜에 의한 정보는 GDT(Global Descriptor Table)에 저장된다. 전체적인 흐름은 인터럽트가 발생시 인터럽트 핸들러에 인터럽트 정보를 알리고 PIT에서 인터럽트 발생을 입력받는다. 그리고 인터럽트의 우선순위에 따라 스케줄링을 하고 우선순위가 가장 높은 프로세스 즉 테스크가 CPU를 선점하게 된다. 그리고 시스템 콜에 의한 정보는 GDT에 저장되어 세그먼트, 커널 서비스에 기록되고 직접 스케줄링 되어 작업을 할 수도 있다.



[그림 2] 다중 큐를 위한 프로세스 정보 테이블

[그림 2]에서 프로세스 정보 테이블(PIT)은 각각의 큐에 대한 정보를 가지고 있다. PIT는 TCB(Task Control Block)을 포함하고 각각의 큐에 대하여 컨트롤 한다. 또한 실시간 테스크(RT_Task)들이 우선순위에 의해 Q1과 Q2 그리고 Q3로 들어오고, 내부 인터럽트와 외부인터럽트가 발생하면 요구 조건에 따라 우선순위를 결정하여 최고의 우선순위에 있는 테스크에게 CPU를 할당한다. 그리고 스케줄러는 현재 프로세스를 대기 큐에 넣은 다음이나, 시스템 콜이 끝난 직후, 프로세스가 시스템 모드에서 프로세스 모드로 돌아오기 바로 전에 실행된다. 다른 경우는 시스템의 타이머가 현재 프로세스의 카운터의 값을 0으로 설정한 경우이다.

PIT는 다중 큐의 테스크들에 대해 다음과 같은 정보를 가지게 된다. 이와 같은 정보는 PIT에서 큐에 대한 포인터를 가지고 우선순위에 대한 정보를 가지게 된다. 여기서 각각의 Q들은 프로그램 카운터, CPU 레지스터, 루틴 파라미터, 그리고 리턴 주소와 스택 포인터 등의 정보를 저장한다.



[그림 3] 다중 큐에 대한 PIT의 구조도

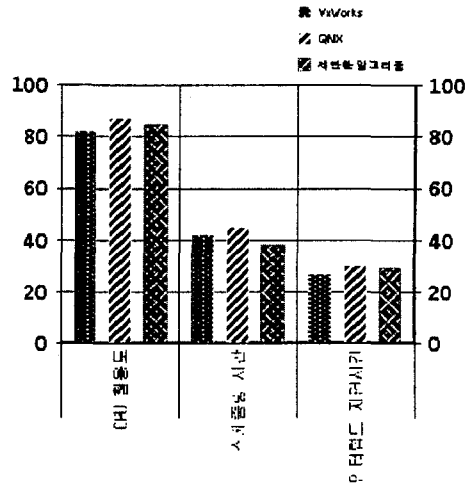
스케줄링 알고리즘

```
RTOS_Scheduling_Algorithm( )
 {
  IF((E_Interrupt || I_Interrupt) == 1)
    {
      mem = PIT;
      reg  = TCB;
    IF(Pre_Task == Task_Running){
      Pre_Task = Task_Ready; }
    Else IF(Current_Task == Highest_Priority){
            PIT = Q_INF;
            TCB = Current_TCB;
            Execute( );
            Exec_Count = Exec_Count +1; }
         Else {
            Task_Pri_Check( );
            PIT = Q_INF;
            TCB = Current_TCB;
            Execute( );
            Exec_Count = Exec_Count +1;  }
    Call Not_Execute_Task( );
 }
Not_Execute_Task( )
 {
   IF(Exec_Count == N)
      Task_Lowest_Pri_Check( );
      PIT = Q_INF;
      TCB = Not_Execute_TCB;
      Execute( );
 }
```

제안하는 스케줄링 알고리즘은 CPU의 활용을 최대한 높이는데 목적을 두었다. 그리고 우선순위가 낮은 테스크를 최대한 구제하기 위함이다. 외부 또는 내부 인터럽트가 발생하면 현재 상태의 PIT와 TCB를 메모리와 레지스터에 각각 저장하게 된다. 그리고 현재의 테스크(Pre_Task)가 실행중인가를 채크하여 실행 중이면 그 테스크를 레디(Task_Ready) 상태로 보내게 된다. 채크중에 있는 테스크(Current_Task)가 최우선순위(Highest_Priority)를 가지고 있는지를 검사하여, 최우선순위의 테스크이면 그 테스크가 속해있는 큐에대한 정보를 PIT에 저장하고, 그 테스크를 TCB에 저장한다. 이렇게 테스크가 확정되면 CPU를 사용하게된다. 또한 우선순위가 높은 테스크들에 의해 실행되지 못하는 테스크를 위하여 일련의 실행횟수를 정하여 실행될 때마다 카운트를 올리게 된다. 기아현상(Starvation)을 방지하기 위해 정해진 실행 횟수를 초과하면 우선 순위가 가장 낮은 테스크를 실행하게 하였다.

3. 실험결과

본 논문의 실시간 운영체제를 위한 효율적인 스케줄링 알고리즘의 실험을 위해 Intel Pentium IV Processor, CPU 933MHz, RAM 256MB의 사양에서 실험하였다. 실험 방법에 있어서 불안한 점은 있었지만 다소 만족한 결과를 얻을 수 있었다. 비교대상은 VxWorks, QNX v6.1로 하였다.



[그림 4] 제안한 스케줄링 알고리즘과 비교

4.결론

본 논문에서 중점적으로 제안한 것은 실시간 운영체제의 핵심이라고 할 수 있는 프로세스에 대한 스케줄링이다.

스케줄링 알고리즘은 복잡하고 내부 또는 외부의 요구에 대해 실시간으로 응답을 해야 하기 때문에 응답 시간에 대해 매우 민감하다. 또한 프로세스가 생성되어 실행할 때까지는 많은 변수가 발생할 수 있다. 우선순위가 높은 프로세스들이 계속적으로 입력될 경우 우선순위가 낮은 프로세스는 실행 가능성이 희박해진다. CPU 활용면에서도 프로세스에 대한 스케줄링을 효율적으로 하여 유휴시간을 최소화하도록 하였다.

실험에서 볼 수 있듯이 성능 평가에서 VxWorks보다는 다소 우수함을 보였다.

5. 참고문헌

[1] Krithi Ramamritham,John A.Stankovic, "Scheduling Algorithm and Operation Systems Support for Real-Time Systems.", Proceedings of the IEEE. Vol 82, No 1, January pp.55-67, 1994

[2] T. Baker, "A Stack-based Resource Allocation Policy for Real-Time Processes", Proceedings of Real-Time System Symposium, pp.191-200, 1990.

[3] A. Burns, K.Tindell, and A. Wellings, "Effective analysis for engineering real-time fixed priority schedulers," IEEE Trans. on Software Eng., Vol. 21, pp. 475-480, May 1995.

[4] N. Kim, M.Ryu, S. Hong, "Visual assessment of a real-time system design: a case study on a CNC controller," in Proc. IEEE Real-Time Systems Symposium, Dec. 1996

[5] N. Audsley, A. Wellings, "Fixed priority scheduling: An historical perspective,"Real-Time Systems, vol. 8, no. 2/3, pp. 129-154, Mar. 1995.