RTLinux에서 효율적인 태스크 스케쥴링을 위한 프레임웍 설계

최대수[U] 임종규 구용완
수 원 대 학 교 전 자 계 산 학 과
rtlinux@lycos.co.kr jkim6@hanmail.net ywkoo@mail.suwon.ac.kr

A Design of Framework for Efficient Task Scheduling on RTLinux

Dae-Soo Choi[U] Jong-Kyu Im Yong-Wan Koo
Dept. of Computer Science, The University of Suwon

요 약

실시간 시스템은 설계단계에서 주어진 태스크 집합에 맞게 구성되어 진다. 이러한 상황은 결국 동적인 상황에 시스템을 쉽게 적용할 수 없는 결과를 초래한다. 본 논문에서는 RTLinux환경에서 보다 동적인 상황에 적응하도록 효율적인 태스크 스케쥴링을 위한 프레임웍을 제시한다. Schedulability manager는 실시간 태스크의 특성에 맞게 스케쥴러를 선택하도록 하였으며 Slack manager는 non-critical 태스크를 스케쥴하도록 한다. 이로서 critical 태스크의 마감시한을 준수하면서 non-critical 태스크의 응답시간을 최소화한다.

1. 서론

기존의 UNIX 운영체제의 스케쥴링 방식은 시분할 환경에 적합하도록 고안되었으며 스케쥴러는 프로세스들의 우선순위를 주기적으로 재계산하여 각 프로세스들이 프로세서를 공평하게 공유할 수 있도록 설계하였다. 그러나 실시간 시스템에서 이러한 비실시간 스케쥴링 방법으로는 시간 제약성을 만족시킬 수 없다. 실시간 시스템의 스케쥴링은 시간 제약성과 정확성을 보장하기 위해서 우선순위에 따르는 선점 스케쥴링(preemptive scheduling)이 필요하다[1]. 실시간 태스크들은 마감시한내에 수행되는 것이 보장되고 수행이 될 수 있다는 것이 예측 가능하여야 한다.

일반적으로 실시간 시스템은 설계단계에서 주어진 태스크 집합에 맞게 구성되어진다. 이러한 상황은 결국 동적인 상황에 시스템을 쉽게 적용할 수 없는 결과를 초래한다. 경성 실시간 시스템은 태스크 수행완료를 보장하는데 초점을 맞춘다[2]. 유닉스 계열의 운영체제를 실시간 운영체제에 사용할 경우 많은 장점이 존재한다. 우선 개발 비용과 시간의 절약이다. 새로운 실시간 운영체제의 개발이 아닌 기존 운영체제의 확장으로 실시간 시스템을 구성하므로, 개발자들에게는 익숙한 개발환경을 제공으로 개발 시간을 많이 단축할 수 있다.

POSIX.1b는 실시간 시스템을 위한 표준으로서 Linux는 현재 POSIX.1b 표준 중 일부를 지원한다. 즉, 특정 연성 실시간 처리에 적합한 정도의 지원이 이뤄지고 있다. RTLinux[3]는 Linux에 최소의 변경을 가해서 완벽하게 Linux를 선점가능하게 구현한 작은 실시간 실행부를 구현하였다. 이는 경성 실시간 운영체제를 위해 개발되었으며 몇가지 문제가 있지만 많은 개발자들에 의해 보완되어지고 있다.

현재 RTLinux를 위한 스케쥴러는 RMS와 EDF 두가지의 스케쥴러가 별도로 구현되어 있다. 이 두 가지 스케쥴러 중에서 사용자는 각각의 스케쥴링 알고리즘의 특성을 전혀 고려하지 않고 두가지 방법을 선택하여 사용하고 있다. 이로 인해 실시간 시스템의 스케쥴링 가능성 검사의 미수행으로 마감시한 실패율을 증가시키는 결과를 초래한다. 또한, 현재 RTLinux에서는 스케쥴 불가능한 태스크를 스케쥴함으로 시스템자체가 멈춰버리는 정지현상이 발생한다. 이러한 현상은 실시간 시스템에서는 매우 치명적이다. 또한 실시간 시스템에서 연성 마감시한을 가지는 비주기 태스크는 마감시한보다 응답시한에 초점을 맞춰 스케쥴링한다. 그리고 RTLinux에서 일반 리눅스 프로세스에 최하위 우선순위를 할당한다.

본 논문에서는 이러한 상황을 고려하여 보다 유연성있고 안정적인 스케쥴링 가능성 검사를 통하여 RMS와 EDF 등 태스크의 특성에 맞는 스케쥴링 방법을 선택하도록 한다. 이로서 마감시한을 보장하고 또한 경성 마감시한을 요하지 않는 경우 남는 여유시간동안 수행하여 전체적인 응답시한을 최소화하는 효율적인 태스크 스케쥴링 프레임웍을 제시한다.

전체적인 논문의 구성을 보면 2장 관련연구로서 RTLinux에 대해 기술하며, 3장에서는 RMS와 EDF에 관하여 기술하고, 4장에서는 효율적인 태스크 스케쥴링을 위한 프레임웍을 제시하고, 5장에서는 결론 및 향후 연구방향에 대하여 기술한다.

2. Real-Time Linux

실시간 리눅스(Real-Time Linux)[2][3]는 driver 관리, X 윈도우 시스템, 네트웍과 같은 구성요소를 그대로 가지면서 Linux를 실시간 운영체제로 전환시킨 작고도 효율적인 코드들로 이루어진다. Linux 커널 아래에 즉, 커널과 하드웨어 사이에 실시간 실행부(executive)를 넣는다는 상당히 단순한 아이디어에 기반을 둔다. 이러한 실행부는 기본적으로 커널 수준에서 실시간 태스크의 수행을 지원하며 전체 Linux 프로세스를 이러한 태스크중 하나로 간주한다. 커널위에서 모든 Linux 프로세스들이 수행된다. 그러므로 실시간 어플리케이션은 실시간 태스크들의 모음으로 정의될 수 있다. Linux 프로세스는 수행을 원하는 태스크가 없을 때 수행하는 가장 낮은 우선순위를 갖는 태스크이다. 이런 방법으로, RTLinux는 실시간 태스크와 Linux 프로세스 두가지 독립적인 계산 수준을 갖는다. 실시간 태스크들은 커널 특권을 가지고 커널 수준에서 수행된다. 반면 Linux와 모든 사용자 프로세스는 더 이상 수행할 실시간 태스크가 없을 때 수행한다. 더욱이 Linux 프로세스는 사용자수준 또는 커널수준에 상관없이 보다 높은 우선순위 태스크가 활성화될 때마다 선점을 당한다.

예측할 수 없는 인터럽트 디스패치 지연를 제거하기 위하여 RTLinux는 매크로를 에뮬레이트함으로 원래 함수를 인터럽트 가능 그리고 불가능하도록 변경한다. 이런 메커니즘을 소프트 인터럽트(soft interrupt)라고 부른다. 이것은 Linux 커널이 인터럽트를 불가능하게 할때조차도 실시간 태스크에는 여전히 유용하게 인터럽트를 허용한다.

RTLinux에서 구현된 원래 실시간 지원은 RT_TASK와 RTIME data type과 [그림 1]의 함수집합을 포함한다. RT_TASK 구조는 스케쥴러가 각각의 실시간 태스크에 관한 정보 즉, 우선순위, 주기 등 모든 정보를 숨긴다. 반면에 RTLinux는 시스템을 나타내는 다른 방법을 소개한다. RTIME type은 tic으로 시간을 측정한다. RTLinux는 microsecond마다 1 tic의 정밀도를 이끌어 내기 위해 PC의 Intel 8354 timer chip을 이용한다. rt_get_time 함수를 호출함으로 시스템 시간을 얻을 수 있다. RTLinux 커널내에 내장함수로 추가되어있다.

```
RTIME rt_get_time(void);
int   rt_task_init(RT_TASK task, void (*fn)(int data),
                   int data, int stack_size, int priority);
int   rt_task_make_periodic(RT_TASK, *task,
                   RTIME start_time, RTIME period);
int   rt_task_wait(void);
int   rt_task_suspend(RT_TASK *task);
int   rt_task_wakeup(RT_TASK *task);
int   rt_task_delete(RT_TASK *task);
```

[그림 1] 실시간 태스크를 관리하는 함수들

[그림 1]의 나머지 함수들은 rt_prio_sched.o라고 불리는 kernel-compiled, loadable module인 RTLinux의 스케쥴러에 정의된다. 이것은 RTLinux의 또 다른 중요한 아이디어를 소개한다. Linux 커널 모듈의 동적인 적재는 많은 잇점을 갖게 한다. 커널을 재컴파일 하지 않고 또한 시스템을 재부팅하지 않고 일부 RTLinux 모듈을 적재하고 제거할 수 있다. RTLinux 스케쥴러와 사용자 실시간 어플리케이션은 커널 모듈로서 컴파일된다. 이러한 모듈은 Linux 쉘에서 적당한 명령을 입력함으로서 시작하고 멈출 수 있다.

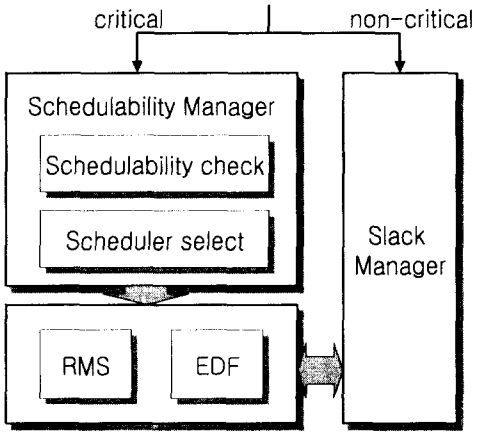
3. RMS와 EDF 알고리즘의 비교

비율단조 스케쥴링(Rate-Monotonic Scheduling:RMS)은 실시간 스케쥴링에 관한 연구 중 가장 많이 연구되고, 사용되는 방법 중의 하나이다[4]. 본 알고리즘은 최적의 정적 알고리즘으로 알려졌으며, 주기를 기초로 태스크에 정적 우선순위를 할당하고 짧은 주기의 태스크에 높은 우선순위를 할당하며 태스크가 도착할 때 우선순위가 할당되므로 우선순위는 재계산 될 필요가 없다. 마감시한이 주기와 같은 태스크 모델에서 최적이며 주기보다 작은 경우에는 최적이지 못하다. 또한 88%의 실용성을 갖으며 실시간 시스템 구현시 간단한 특성으로 많이 이용되고 있다. 이는 다음과 같은 가정을 하고 있다. 서로 독립적인 주기 태스크만을 고려하였고 태스크간의 선행관계가 없으며 태스크들은 선점될 수 있고 문맥교환과 태스크 스케쥴링에 드는 비용을 무시한다. 또한 모든 태스크들은 서로 독립적이며 선행 제약은 존재하지 않는다.

마감시한 우선 스케쥴링(Earliest-Deadline First: EDF) 알고리즘은 동적 우선순위 알고리즘 방법으로 선점 알고리즘들 중에서 최적인 것으로 알려져 있다[5]. 태스크의 우선순위는 마감시한에 따라서 할당된다. 즉, 마감시한이 짧을수록 높은 우선순위가 할당되므로 임의의 순간에 실행되는 태스크는 실행이 완료되지 않은 태스크들 중에서 마감시한이 가까운 것이 선택된다. 또 정적 알고리즘과 달리 태스크의 우선순위가 시간에 따라 변하게 된다. 이는 마감시한이 주기와 동일하거나 주기에 비례하여 작을 경우 RMS와 같다. 마감시한이 주기보다 작은 경우 최적이다. EDF는 스케쥴링 오버헤드가 발생한다.

4. 효율적 태스크 스케쥴링을 위한 프레임웍

현재까지 RTLinux를 위한 스케쥴러는 RMS와 EDF 두가지의 스케쥴러가 별도로 구현되어 있다. 이 두가지 스케쥴러 중에서 사용자는 각각의 스케쥴링 알고리즘의 특성을 고려하지 않고 두가지 방법중 하나를 선택하여 사용하고 있다. 이로 인해 실시간 스케쥴링 가능성 검사의 미수행으로 마감시한 실패율을 증가시키는 결과를 초래하고 있다. 또한 현재 RTLinux에서는 스케쥴 불가능한 태스크에 대한 스케쥴로 인하여 시스템이 정지되는 현상이 발생된다. 이러한 현상은 경성 실시간 시스템에서 매우 치명적이다. RTLinux에서 효율적인 태스크 스케쥴을 하기 위한 프레임웍은 [그림 2]와 같다.



[그림 2] 태스크 스케쥴 프레임웍

슬랙(slack)은 주기 태스크가 실행을 종료했을 때 마감시한까지 사용되지 않는 부분을 말한다. 슬랙 스틸링 알고리즘[6]은 엄격한 제한시간을 가지는 주기 태스크의 마감시한을 준수하면서 비주기 태스크의 응답시간을 최소화하기 위한, 프로세서 시간을 태스크로부터 스틸링(stealing)하는 방식이다.

프레임웍 구조를 살펴보면 크게 Schedulability Manager와 Slack Manager로 분류해 볼 수 있다. 각각의 역할을 나눠 보면 다음과 같다.

• Schedulability Manager

스케쥴가능성 검사를 수행하여 주어진 태스크에 대한 프로세서 이용률을 구한다. 그 값에 의하여 현재 태스크 집합이 어떠한 스케쥴러가 적당한지 또는 스케쥴이 불가능한지 정보를 얻을 수 있다. 스케쥴러 선택기를 통하여 스케쥴러 가능성 검사결과를 토대로 RMS와 EDF 스케쥴러를 선택하여 이용한다. 또한 스케쥴이 불가능한 경우는 스케쥴링을 하지 않음으로 시스템의 정지현상을 막을 수 있다. 또한 Linux의 Lodable kernel Module기능을 이용해 새로운 스케쥴러를 이용할 수 있는 확장성이 있다.

• Slack Manager

RTLinux에서 Linux 프로세스는 최하위 우선순위를 할당 받는다. 또한 비주기 태스크가 들어오면 슬랙을 확인하여 할당함으로 non-critical 태스크들의 전체적인 마감시한을 보장하고자 한다. 그러므로 Slack Manager는 Schedulability Manager와 달리 부가적인 기능으로 non-critical 태스크를 다룬다. 즉, 가용한 여유 slack이 있을 때 동작하게 된다. non-cirtical 태스크 발생시 슬랙 시간을 계산하고 실시간 태스크로부터 시간을 얻어낸다. Slack manager가 스케쥴하기 위해 그 시간을 이용한다.

5. 결론 및 향후 연구방향

RTLinux는 보다 유연성있는 경성 실시간 시스템을 구현하기 위한 좋은 환경을 제공한다. 예측가능하고 마감시한을 보장하기 위하여 본 논문에서는 Schedulability manager와 Slack manager를 구분하여 설계하였다.

본 논문에서 RTLinux가 보다 동적인 상황에 적응하도록 하여 특정 어플리케이션 특성에 맞게 실시간 스케쥴러를 선택하여 이용할 수 있는 프레임웍을 제시하였다. 이는 critical 태스크의 마감시한을 준수하면서 non-cirtical 태스크의 응답시간을 최소화할 수 있다. 향후 연구과제는 본 논문에서 제시한 프레임웍을 실제 여러 하드웨어구조에서 스케쥴링을 분석, 검증하고 이외에도 다른 스케쥴링 알고리즘을 모듈화하여 프레임웍을 보다 확장시키는 것이다.

참고문헌

[1] W. Zhao, K. Ramamritham, John. A. Stankovic, "Preemptive Scheduling Under Time and Resource Constraints", IEEE Computers, Vol C-36, No.8, pp.949-960, Aug. 1987.

[2] Victor Yodaiken, "The RTLinux Manifesto", http://www.rtlinux.org

[3] Michael Barabanov and Victor yodaiken. "Real-Time Linux", Linux journal, February 1997.

[4] John Lehoczky, Lui Sha and Ye Ding, "The Rate Monotonic Scheduling Algorithm:Exact Characterization And Average Case Behavior", Proc IEEE Real-Time Systems Symposium, pp.166-171, Dec. 1989.

[5] Chetto. H and Chetto. M, "Some Results of the Earliest Deadline Scheduling Algorithm", IEEE Transactions on Software Engineering Vol. 15, No. 10, pp.1261-1269, Oct. 1989.

[6] Davis, R. I. "Approximate Slack Stealing Algorithms for Fixed Priority Preemptive Systems", Dept. of Computer Science University of York, UK, Report Number YCS-93-217