실시간 프로세스의 최악 응답 시간 예측에 관한 연구

이동식, 김기창
인하대학교 정보공학과
e-mail : zecksmaster@empal.com

A Study on Worst Case Response Time Prediction of RT Process

Dongsik Lee, Kichang Kim
Dept. of Information Technology, Inha University

요 약

본 논문에서는 Ingo Molnar 의 realtime-preempt 패치가 적용된 리눅스를 활용한 실시간 시스템에서 다른 프로세스와 동시에 수행하지 않고도 다른 프로세스에 의한 실시간 프로세스의 최악 응답 시간의 변화를 예측할 수 있도록 최악 응답 시간에 영향을 주는 커널 모드에서 선점 금지 시간을 프로세스 별로 분석을 하기 위한 도구를 커널 모듈로 구현하여 실시간 프로세스의 최악 응답 시간을 예측할 수 있음을 보였다.

1. 서론

범용 운영체제로 개발된 리눅스는 전세계의 많은 개발자들에 의해서 기존의 커널을 수정하는 작업을 통해서 임베디드 시스템에 적용되어 사용되고 있을 뿐만 아니라 실시간 시스템에서도 사용할 수 있도록 스케줄러의 수정, 선점형 커널의 지원 등 각종 지연 시간을 줄이기 위한 연구가 활발히 진행되고 있다.[2]

그러나 현재 리눅스의 실시간 성능은 많이 높아졌지만 여전히 경성 실시간성을 지원하지 못하고 있다. 리눅스는 선점형 커널의 지원으로 커널 모드에서 우선순위가 높은 프로세스에 의한 커널 선점이 가능하지만 여전히 선점이 불가능한 영역이 남아있다.[1][6] 이러한 리눅스에서 스케줄링 가능성 분석을 위해서는 프로그램의 최악실행시간 정보가 필수적인 요소이다.[5] realtime-preempt 패치가 적용된 리눅스에서는 시스템에서 발생하는 이벤트에 대한 프로세스의 응답 시간을 측정하여 최악실행시간에 매우 큰 영향을 줄 수 있는 최악 응답 시간을 예측함으로써 시스템의 실시간 성능 및 시스템에서 실시간 프로세스의 마감시간 준수 가능성을 판단한다.[7]

그러나 여러 프로세스가 동시에 동작 가능한 실시간 시스템에서 실시간 프로세스의 최악 응답 시간 예측은 시스템에서 실행될 모든 프로세스가 실행되는 상황을 만들어 실시간 프로세스의 응답 시간을 측정해야만 한다. 게다가 시스템에 새로운 프로세스를 추가해야 할 경우에도 똑같은 과정을 통해서 예측을 할 수 밖에 없다.

본 논문에서는 이러한 문제점을 해결하기 위해 프로세스가 커널 모드에서 동작할 때 선점을 금지하는 시간을 측정하기 위한 도구를 커널 모듈로 구현하고 이를 이용해 프로세스 별로 선점 금지 시간을 측정하고 새로운 프로세스를 실행시키지 않고도 실시간 프로세스의 최악 응답 시간의 변화를 예측할 수 있음을 보인다.

본 논문의 구성은 다음과 같다. 2 장에서는 최악 응답시간을 예측하기 위해 주로 사용되는 도구들에 대해서 알아보고, 3 장에서는 기존 도구들의 문제점을 보완하기 위한 분석 도구를 구현한다. 4 장에서는 구현한 도구를 이용한 실험을 통해 최악 응답 시간을 예측하고 5 장에서는 결론 및 향후 연구방향을 제시한다.

2. 관련 연구

실시간 프로세스의 마감시간 준수 가능성을 알아보고자 할 때 주로 최악 응답 시간을 측정하는 방법을 주로 사용한다. 최악 응답 시간은 대기 중인 프로세스가 어떠한 특정한 이벤트에 의해서 깨어나 다시 실행되는 한계 시간을 의미한다.[7]

Realfeel 은 mlockall() 시스템 콜을 이용하여 프로세스의 페이징을 금지시키고 스케줄링 정책과 우선 순위를 지정한 후 /dev/rtc 를 이용하여 주기적인 인터럽트를 발생시켜 주기적인 인터럽트에 대해 프로세스가 깨어나는 시간으로 최악 응답 시간을 예측한다.[3] 그리고 cyclictest 의 경우 우선 순위가 다른 여러 개의 쓰레드를 생성하고 소프트웨어 타이머를 이용하여 시그널을 발생시켜 시그널에 대한 응답시간을 측정하거나 nanosleep()을 이용하여 해당 쓰레드가 깨어나 다시 실행되는데 걸리는 시간을 측정함으로써 최악 응답 시간을 예측한다.[4]

이러한 도구들을 이용하여 보다 정확하게 실시간 프로세스의 최악 응답 시간을 예측하고자 하는 경우 다른 프로세스에 의한 선점 지연은 실시간 프로세스의 최악 응답 시간에 영향을 줄 수 있으므로 시스템에서 실행할 모든 프로세스가 동시에 작동하는 상황을 만들어서 실험할 수 밖에 없으며 이러한 상황을

만드는 것 또한 어렵다.

3. 선점 금지 영역 및 분석 도구 구현

현재 리눅스를 실시간 시스템에 활용하기에 가장 문제가 되는 부분은 커널과 디바이스 드라이버에서의 선점이 금지되는 영역이다.[2] 이 영역들은 실시간성을 요구하지 않는 프로세스도 사용할 수 있다. 만약 실시간성을 요구하지 않는 프로세스가 커널 모드의 선점이 금지되는 영역의 코드를 수행하고 있는 경우 실시간 프로세스는 선점이 지연된다. 커널 모드에서의 실시간성을 요구하는 프로세스의 선점 지연은 응답 시간에 영향을 주게 되므로 실시간 프로세스의 최악 응답 시간은 늘어나게 된다. 리눅스에서 선점 금지 영역은 <표 1>의 함수에 의해서 보호되며 preempt_disable()함수는 여러 번 호출될 수 있으며, preempt_enable()함수가 호출되어 선점 카운터가 0 이 되면 비로소 커널 모드에서 우선 순위가 높은 프로세스에 의해서 선점이 가능해진다. [1]

<표 1> 커널 모드에서 선점 금지와 관련된 함수

```
#define preempt_disable() \
do { \
   inc_preempt_count(); \
   barrier(); \
} while (0)

#define preempt_enable() \
do { \
   __preempt_enable_no_resched(); \
   barrier(); \
   preempt_check_resched(); \
} while (0)

#define __preempt_enable_no_resched() \
do { \
    barrier(); \
    dec_preempt_count(); \
} while (0)
```

커널 모드에서 선점이 금지되는 시간을 측정하기 위해서 커널의 선점 금지와 관련된 함수를 <표 2>와 같이 수정했다. 수정된 코드에 추가된 두 개의 함수는 선점 카운터가 0 인 경우에만 시간을 측정하여 커널 모드에서 선점이 금지되는 시간을 메모리 상에 저장한다.

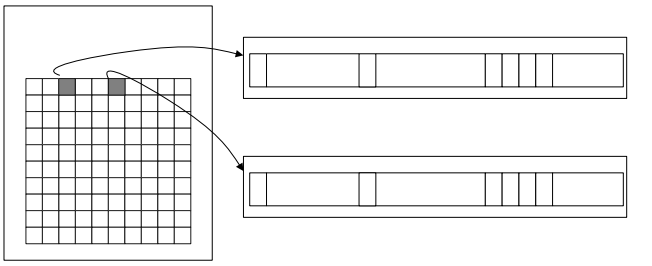
<표 2> 선점 금지 시간 분석을 위해 수정된 커널 코드

```
#define preempt_disable() \
do { \
   kernel_disable_logging_fn(); \
   inc_preempt_count(); \
   barrier(); \
} while (0)

#define __preempt_enable_no_resched() \
do { \
    barrier(); \
    dec_preempt_count(); \
    kernel_enable_logging_fn(); \
} while (0)
```

커널 내부에서 선점이 금지되는 시간을 프로세스별로 보다 정확하게 측정하기 위해서는 실행에 대한 오버헤드가 최대한 적어야 한다. 현재 시간을 측정하는 오버헤드를 제외하고 프로세스 별로 선점 금지 시간을 측정해서 저장하는 오버헤드를 최대한 줄이기 위해서 커널 모듈은 로드될 때 <표 3>에서와 같이 리눅스에서 실행 가능한 프로세스 개수만큼의 포인터 배열을 선언하고 NULL 값으로 미리 초기화 시켜놓음으로써 로깅할 프로세스인지를 판단할 때 해당 프로세스 id 위치에 해당하는 배열에 저장된 값이 NULL 값인지만 확인해보는 과정만으로 최대한 오버헤드를 줄이면서 판단 가능하도록 하였다.

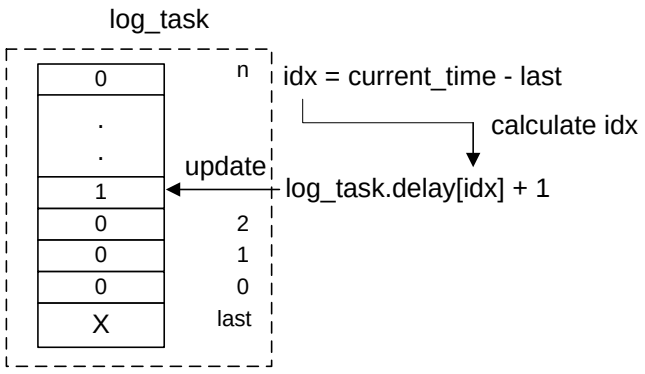
<표 3> 프로세스 별 선점 금지 시간 저장을 위한 구조체

```
typedef struct log_task {
   unsigned int delay[MAX_PREEMPT_DELAY];
   u64 last;
} LOG_TASK;

LOG_TASK *logging_task[MAX_PID];
```



(그림 1) 모듈의 자료구조의 예

그리고 선점 금지 시간을 저장할 때에도 최대한 저장할 위치를 찾는 오버헤드를 줄이기 위해서 (그림 2)와 같이 log_task 구조체의 last 변수에 저장된 값과 현재 시간의 차를 log_task 구조체의 delay 배열의 인덱스로 사용하여 해당 위치의 값을 1 증가 시킴으로써 선점 금지 시간을 한번에 저장하도록 하였다.



(그림 2) 선점 금지 시간 저장 방법

4. 실험 및 결과 분석

본 논문의 제안된 선점 금지 시간 분석 모듈은 리눅스 커널 2.6.18 버전에 Ingo Molnar 의 realtime-preempt 패치가 적용된 시스템에서 gcc 4.1.2 버전을 이용하고 구현되었다. 실험에는 Intel x86 기반의 Pentium VI 1.5GHz, 256MB PC 를 사용하였다.

실시간성을 요구하지 않는 프로세스에 의한 실시간 프로세스의 최악 응답 지연 시간의 변화를 예측할 수 있음을 보여주기 위해 먼저 4 개의 프로세스 A, B, C,

D 가 각자 실행될 때 커널 모드에서의 선점을 금지시키는 시간을 측정해 보았다. 그리고 이 프로세스가 실시간 프로세스와 동시에 실행되는 여러 상황에서 실시간 프로세스의 응답 시간을 측정해 보았다.

<표 4> Non-RT 프로세스의 선점 금지 시간

프로세스	최소 선점 지연 시간	최대 선점 지연 시간
A	5 us	13 us
B	12 us	24 us
C	11 us	39 us
D	8 us	21 us

<표 4>는 실시간성을 요구하지 않는 각기 다른 일을 수행하는 동안 커널 모드에서 선점 금지 시간을 측정한 경과이다.

<표 5> RT 프로세스의 최악 응답 지연 시간 변화

프로세스	최대 응답 지연 시간	변화량
R	79 us	
R+A	101 us	22 us
R+B	110 us	31 us
R+C	121 us	42 us
R+D	109 us	30 us
R+A+B	111 us	32 us
R+A+C	122 us	43 us
R+A+D	108 us	29 us
R+B+C	123 us	44 us
R+B+D	110 us	31 us
R+C+D	123 us	44 us
R+A+B+C	122 us	43 us
R+A+B+D	112 us	33 us
R+A+C+D	121 us	42 us
R+B+C+D	122 us	43 us
R+A+B+C+D	123 us	44 us

<표 5>는 실시간 프로세스와 실시간성을 요구하지 않는 프로세스가 동시에 수행하면서 측정된 실시간 프로세스의 최대 응답 지연 시간 측정 결과이다. 결과를 보면 실시간 프로세스 R 의 최악 응답 시간은 실시간 프로세스를 제외한 프로세스들에 의한 커널 모드에서의 최대 선점 금지 시간만큼 지연됨을 알 수 있다. 즉, 실시간 시스템에 새로운 프로세스를 추가하거나 제거하는 경우 해당 프로세스에 의한 커널 모드에서의 선점 금지 시간만 분석하면 실시간 프로세스의 최악 응답 지연 시간의 변화를 예측할 수 있음을 알 수 있다.

5. 결론 및 향후 연구

리눅스는 소스 코드가 공개되어 있으며 누구나 수정하고 재 배포할 수 있는 운영체제이다. 이러한 특징으로 인해 전세계 개발자들에 의해서 빠르게 발전해왔으며 실시간 시스템에서도 활용하기 위해서 많은 노력을 기울이고 있다. 그러나 리눅스의 실시간 성능은 많이 높아졌지만 여전히 실시간 시스템으로 활용하기에는 문제점들을 가지고 있다. 그리고 여러 프로세스가 동시에 수행될 수 있으므로 실시간 프로세스의 최악실행시간에 대한 예측이 필요하다.

이에 본 논문에서는 리눅스를 실시간 시스템에서 활용하고자 할 때 시스템에서 동작하게 될 모든 프로세스를 동시에 실행시키는 상황을 만들어서 실시간 프로세스에 최악 응답 지연 시간을 예측하는 방법을 사용하지 않고도 실시간 프로세스의 최악 응답 지연 시간의 변화를 예측할 수 있도록 각 프로세스 별로 커널 모드에서의 선점 금지 시간 측정을 하는 모듈을 구현했다. 그리고 실험을 통해서 실시간 프로세스의 최악 응답 지연 시간의 변화를 예측할 수 있음을 보였다.

그러나 본 논문에서 구현된 프로세스 별 선점 금지 분석 커널 모듈은 보다 정확한 선점 금지 측정을 위해서 가능한 한 오버헤드를 줄이고자 선점 금지 시간을 요청한 프로세스의 pid 값만 이용하여 분석을 수행하는 이유로 인해서 프로세스가 새로운 프로세스를 생성하는 경우에 대해서는 분석을 수행하지 못한다. 향후 연구에서는 보다 정확한 분석을 위해서 오버헤드가 적으면서도 프로세스가 새로운 프로세스를 생성하는 경우까지 커널 모드에서 선점 금지 시간을 분석할 수 있도록 개선할 필요가 있다.

참고문헌

[1] Linus Tovalds, http://www.kernel.org
[2] Ingo Molnar, http://people.redhat.com/mingo/realtime-preempt/
[3] Mark hahn. http://brain.mcmaster.ca/hahn/realfeel
[4] Thomas. Gleixner. http://rt.wiki.kernel.org/index.php/Cyclictest
[5] 박현희, 최명수, 양승민, 최용훈, 임형택, „XScale 프로세서 기반의 임베디드 소프트웨어를 위한 최악실행시간 분석도구의 구현“ 한국정보처리학회 정보처리학회논문지 A, Vol 12, No. 5, 365-374
[6] Robert M, Love, Linux Preemption-patch Kernel, http://www.tech9.net/rml/linux, 2002
[7] Brosky. S, Rotolo. S, “Shielded processors : guaranteening sub-millisecond response in standard linux” Parallel and Distributed Processing Symposium, 2003. proceddings