DVS 적용을 위한 프로세스 모니터링 기법

이준희° 김효승 차호정
연세대학교 컴퓨터과학과
{jhl,hskim,hjcha}@cs.yonsei.ac.kr

A Process Monitoring Strategy towards DVS Applications

Joonhee Lee° Hyoseung Kim Hojung Cha
Dept. of Computer Science, Yonsei University

요 약

본 논문에서는 동적전압변경(DVS)을 적용하기 위해 운영체제의 커널상에서의 프로세스 모니터링 기법을 제안한다. 이상적인 DVS 알고리즘은 응용프로그램의 수정없이 자동으로 수행되어야 하며 프로세스의 QoS를 고려해야 한다. 본 논문에서는 이를 위해 커널상에서 프로세스의 프로세서 수행상태와 QoS와 밀접한 관련이 있는 주기정보를 확인할 수 있는 프로세스 모니터링 기법을 제안한다. 또한 리눅스 운영체제상에서 실제 구현하고 실험을 통해 제안하는 모니터링 기법이 정확하게 프로세스의 상태와 주기정보를 추출할 수 있음을 밝힌다.

1. 서론

최근 프로세스 상태에 따른 동적전압변경(DVS) 기법 연구가 활발히 이루어지고 있다. 프로세스 상태에 따른 동적전압변경 기법이란 커널내부에서 프로세스 상태를 모니터링하여 프로세서의 전압을 자동으로 변경해주는 기술을 말한다. 이는 기존 응용프로그램의 변경이 필요 없고, 사용자에게 별도의 조작을 요구하지 않는 점에서 큰 장점을 지닌다.[1] 이를 위해서는 운영체제의 커널상에서의 적절한 모니터링 기법이 필요하다.

프로세스 모니터링을 기반으로 한 DVS 알고리즘을 제시한 대표적인 연구로는 Vertigo[2]가 있다. Vertigo는 커널 모니터링을 기반으로 수행에 관련된 프로세스들을 에피소드로 나누고 에피소드의 주기성을 찾기 위해 노력하였다. 그러나 커널상에서 프로세스들 간의 수행에 관련된 방법은 직접적인 수행이외에 IPC등의 다양한 수행방법이 존재하여 에피소드를 확인하기 어려우며 오버헤드가 크다. 또한 특정 시스템콜 모니터링에 근거하여 일반적인 프로세스의 주기성을 찾는데는 제한이 있다. 또다른 연구로 Benini[3]가 제안한 모니터링 구조가 있다. Benini는 Observer라는 구조를 작성하여 커널상에서 각 디바이스의 모니터링에 초점을 맞추었다. 그러나, CPU 사용률에 대한 모니터링은 프로세스 단위로 이루어지지 않고 있으며 프로세스의 주기성등 프로세스의 상태 및 QoS와 관련된 모니터링은 이루어지지 않았다. ECOSystem[4]은 파워를 운영체제가 관리해야 하는 가장 중요한 리소스로 인식하고 각 디바이스가 사용하는 파워소비를 운영체제에서 모니터링하여 프로세스의 수행을 제한하는 방법으로 파워소비를 줄이려 노력하였다. 이를 위해서는 프로세스 모니터링이 가장 중요하다. 그러나 ECOSystem은 타이머 인터럽트 기반의 모니터링으로 10ms안의 프로세스의 정확한 모니터링이 어렵다.

기존 연구의 문제점을 보완하기 위해 본 논문에서 제시하는 프로세스 모니터링 기법은 범용운영체제 기반의 시스템에 적용가능하며, 10ms등의 일정간격에 상관없이 프로세스의 프로세서 수행상태를 정확히 모니터링할 수 있다. 또한 프로세스의 QoS관련 정보를 모니터링하기 위해 커널에서 프로세스의 QoS를 처리하기 위해 사용되는 커널흐름 중심의 모니터링 기법을 사용하여 멀티프로세스 수행시에도 다른 프로세스 및 인터럽트등의 영향을 받지 않으면서 프로세스의 주기정보를 정확히 측정할 수 있다. 제시하는 모니터링 기법은 리눅스상에 실제 구현하여 프로세스 상태 및 주기정보를 정확히 판독할 수 있음을 실험을 통해 밝혔다.

논문의 구성은 다음과 같다. 2장에서 제안하는 모니터링 구조에 대해 설명하고 3장에서는 실험을 통해 제시한 모니터링 기법이 프로세스의 상태 및 주기정보를 정확히 판독할 수 있음을 밝힌 후, 4장에서 결론을 맺는다.

2. 프로세스 모니터링 기법

제시하는 모니터링 기법은 크게 프로세스 상태 모니터링과 QoS를 지원하기 위한 프로세스의 주기성 모니터링으로 이루어진다.

2.1 프로세스 상태 모니터링

프로세스 상태 모니터링이란 각 프로세스가 프로세서를 사용하는 상태를 측정하는 것이다. 기존 연구에서 프로세스 상태 모니터링은 일정 시간 간격마다 현재 프로세서를 사용하고 있는 프로세스를 확인하는 방법을 이용하였다. 일정 시간 간격을 이용한 확인 방법은 프로세서의 고성능화로 프로세스들의 수행 시간이 짧아지고 스케줄링이 보다 빈번하게 일어남으로 인해 프로세스의 상태를 정확히 측정하기 어렵다. 또한 측정 시간 간격을 줄일 수도 있으나 모니터링으로 인한 오버헤드가 커진다.

• 본 연구는 정보통신연구진흥원에서 지원하는 정보통신기초기술연구지원사업으로 수행하였음 (과제번호 : C1-2003-A1-2000-0240)

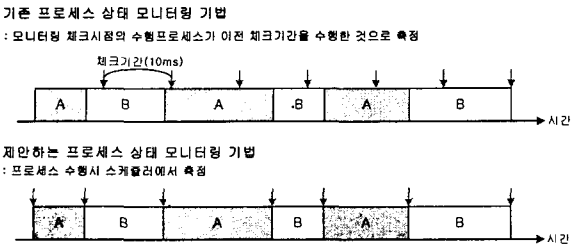

그림 1 : 프로세스 상태 모니터링 기법

그림 1은 기존 프로세스 상태 모니터링 기법과 제안하는 방법을 비교한 것이다. 기존 프로세스의 프로세서 사용상태 모니터링 기법은 일정간격마다 프로세스의 상태를 체크하는 방법임에 반해 제시한 모니터링 구조는 커널내부의 스케쥴링 부분을 모니터링한다. 스케쥴러는 프로세스들을 수행시키는 역할을 하므로 스케쥴러에서 프로세스의 상태변경을 처리하는 커널흐름을 모니터링함으로써 프로세스의 상태를 정확히 확인할 수 있다. 또한 커널에 모니터링을 위해 기능을 추가하여 프로세스의 상태를 측정하는 것이 아니라 기존의 커널코드에 후킹을 통해 체크하므로 오버헤드가 작다.

2.2 프로세스 주기성 모니터링

실시간 운영체제에서 QoS가 필요한 프로세스는 커널에 데드라인을 넘겨준다. 그러나 범용운영체제에서는 이러한 정보를 알기 어렵다. MP3, 동영상 플레이어등은 QoS 보장을 위해 일정간격마다 스케줄링되어 수행함에 따라 수행시간에 있어서 주기성을 띄게 된다. 따라서 본 연구는 이러한 주기와 해당 주기에서의 평균수행시간을 모니터링함으로써 QoS를 필요로 하는 프로세스를 확인한다.

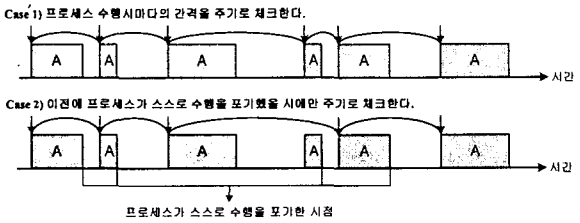

그림 2 : 기존의 주기성 모니터링 기법

그림 2는 기존의 주기성 모니터링 기법을 도식화한 것이다. 기존 연구는 프로세스의 수행시간마다의 간격을 주기로 측정하거나, 프로세스가 스스로 수행을 포기했을 때를 체크하여 주기를 확인하였다. 그러나 멀티프로세스 환경에서는 다른 프로세스의 선점이나 인터럽트 등의 외부요인에 의해 프로세스의 수행시점이 일정치 않다. 따라서 기존 방법으로는 QoS를 필요로 하는 프로세스의 주기를 정확히 판독하기 어렵다.

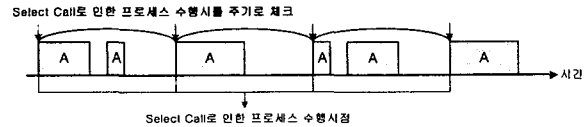

그림 3 : 제안하는 주기성 모니터링 기법

그림 3은 제안하는 주기성 모니터링 기법을 도식화한 것이다. 본 연구는 기존 연구의 문제점을 해결하기 위해 QoS를 필요로 하는 프로세스는 이를 지원하기 위해 커널에 다양한 방법으로 요청을 할 것이라는 가정한다. 그리고 이를 처리하기 위한 커널흐름을 모니터링함으로써 주기성을 확인한다. 제시하는 주기성 모니터링 기법은 다양한 프로세스가 있을 시에도 다른 프로세스나 인터럽트에 영향을 받지 않고 해당 프로세스의 실제 주기를 확인할 수 있다.

주기성을 처리하는 커널흐름은 다음과 같이 세가지 경우가 있다. 첫째, QoS를 요구하는 프로세스 중에는 동영상 재생과 같이 주기적으로 I/O 장치를 통해 데이터를 처리하는 경우가 있다. 이러한 프로세스들은 주기적 수행을 위해 select나 poll시스템 콜을 사용한다. 따라서 select 및 poll의 커널내 처리함수 내에서 특정 파일 디스크립터를 읽고 쓰는 간격을 확인하여 주기성을 확인할 수 있다. 둘째, X window 환경 하에서 패널의 시계와 같은 경우 갱신을 위해 setitimer나 alarm등을 이용한다. 이 경우 프로세스를 일정기간마다 블록상태에서 실행상태로 변경하기 위해 커널이 해당 프로세스에 시그널을 보내므로 이 시간간격을 측정하여 주기성을 확인한다. 셋째, 주기성을 가진 프로세스는 시스템콜 이외에 다양한 방법으로 주기적 수행을 요청할 경우 커널상에서는 결국 타이머를 통해 주기적 수행을 처리한다. 따라서 커널의 타이머 처리 루틴을 모니터링하여 주기성을 확인할 수 있다. 예를 들어, 그림 4는 리눅스에서 구현한 커널의 타이머 처리 루틴 모니터링 구조를 도식화한 것이다. 프로세스가 일정주기 후에 수행되도록 커널에 요청할 경우 프로세스는 schedule_timeout() 함수를 호출하여 처리한다. schedule_timeout()은 process_timeout() 함수를 호출하여 동적타이머에 일정 주기를 등록한 후, 해당 주기가 끝났을 때 프로세스를 수행시킨다. 따라서 process_timeout() 함수를 통해 실행되는 프로세스의 주기를 확인하여 프로세스의 주기성을 확인할 수 있다.

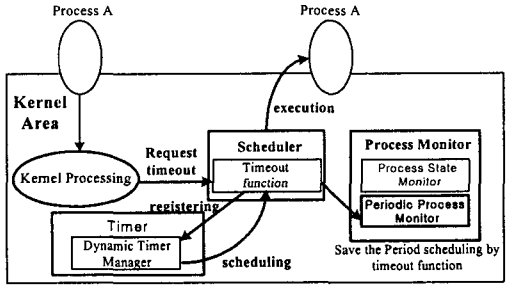

그림 4 : schedule_timeout()을 통한 주기성 모니터링

이상의 세가지 커널루틴을 모니터링하여 측정된 각 프로세스의 시간간격은 주기성을 확인하는 계산을 통해 주기성여부를 확인하게 된다. 주기성 계산식은 다음과 같다.

$$P = \{p_{n-k-1}, \cdots, p_{n-2}, p_{n-1}, p_n\}$$

$$\frac{AVG(P)}{w} < MAX(P) - MIN(P) \quad (1)$$

P는 프로세스의 해당 수행주기의 수행시간이며 p_n은 가장 최근의 수행주기이다. 주기성 계산을 위해 프로세스 수행주기와 수행시간을 최근 k개 저장한다. 저장한 주기중의 최대값에 최소값을 뺀 후

오차가 저장한 주기의 평균값의 w분의 1안인지를 확인하여 주기성을 확인한다. 리눅스상에서의 실험결과로는 k는 5, w는 10의 값이 적정함을 확인하였다.

제시한 주기성 모니터링 기법은 주기성을 띄는 프로그램의 실제 수행방식에 따른 커널흐름을 모니터링함으로써 다른 프로세스의 선점, 인터럽트등의 외부요인에 영향을 받지 않고 정확히 프로세스의 주기정보를 판독할 수 있다.

3. 실험 및 분석

프로세스 모니터링 구조는 리눅스 커널 2.4.21에 실제 구현하였다. 실험은 센트리노 프로세서가 탑재된 노트북을 사용하였으며 일반적으로 사용되는 응용프로그램을 선정하여 수행시킨 후 모니터링으로 얻은 데이터를 저장, 시스템콜을 사용하여 추출하였다.

실험에 사용된 응용프로그램은 MP3 플레이어인 xmms, 동영상 플레이어인 gmplayer, 웹브라우져인 mozilla, 시스템 시간보정 데몬인 ntpd이다. 실험은 제시한 프로세스 모니터링 구조가 프로세스의 상태를 정확히 확인할 수 있는가를 알아보기 위해 각 프로그램을 동시에 수행한 상태에서 프로세스들의 수행시작시간과 수행시간을 확인하였으며 수행결과는 그림 5와 같다.

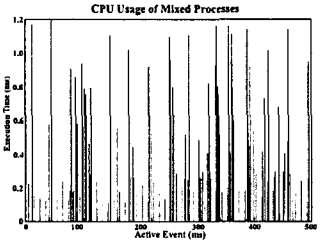

그림 5 : 프로세스의 프로세서 수행상태 모니터링 결과

그림 5에서 보듯이 멀티프로세스 환경에서 각 응용프로그램과 관련한 프로세스 상태를 확인하기 어렵다. 제시한 모니터링 기법은 이상의 여러 프로세스가 수행한 상태에서 각 응용프로그램과 관련한 프로세스의 상태는 추출할 수 있으며, 결과는 그림 6과 같다.

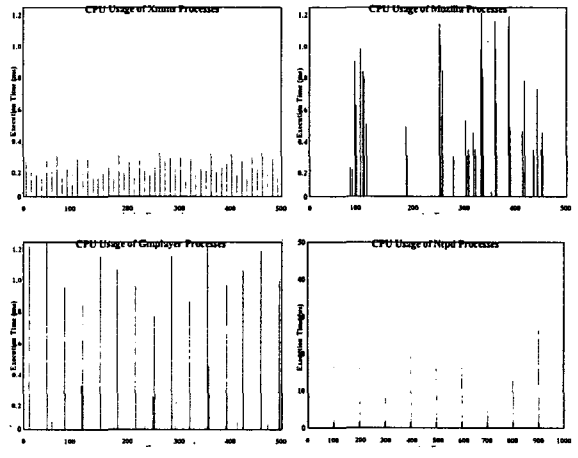

그림 6 : 각 응용프로그램의 프로세서 수행상태 모니터링 결과

그림 6은 그림 5에서 동시에 수행되는 프로세스 중에서 각 응용프로그램과 관련된 프로세스들의 프로세서 수행상태를 us단위로 정확히 모니터링할 수 있음을 보여준다. 그림 6에서 응용프로그램의 프로세스중에는 주기성을 가지고 수행하는 프로세스가 있다. 표 1은 위의 실험에서 수행한 각 프로세스에 대한 주기성 판독 결과를 보여준다.

프로세스이름	프로세스ID	모니터링결과	평균수행시간(μs)	주기(μs)
mozilla	3678	비주기	-	-
	3666	비주기	-	-
gmplayer	4074	select	1098	35058
	4194	process_timeout	35	59829
xmms	4060	비주기	-	-
	4062	process_timeout	4	99708
	4195	process_timeout	212	19933
	4196	process_timeout	22	9858
ntpd	3314	setitimer	16	997054

표 1 : 프로세스 주기성 모니터링 결과

표 1과 같이 모니터링 구조는 네 가지의 응용프로그램중 gmplayer, xmms, ntpd의 쓰레드 및 프로세스의 일부만이 주기성을 가지고 있음을 구별해 내었으며, 해당 응용프로그램의 주기와 해당 주기의 평균수행시간을 정확히 판독하였다. 이상의 실험을 통해 제시하는 모니터링 구조는 프로세스 단위로 프로세서 수행시간을 정확히 모니터링할 수 있으며, 해당 프로세스의 주기성을 판독하고 주기성이 있을 시에는 해당 주기와 평균수행시간을 판독해낼 수 있다. 이러한 모니터링 구조는 QoS를 고려하여 운영체제상에서 자동으로 프로세스 상태에 따른 DVS알고리즘에 중요한 자료로 사용된다.

4. 결론

본 논문에서는 프로세스 상태에 따라 커널상에서 자동으로 동적 전압변경이 가능토록 하기 위해 필요한 프로세스 모니터링 기법을 제안한다. 그리고 실제 실험을 통해 프로세스 상태 및 QoS를 필요로 하는 프로세스의 주기성 확인이 가능함을 밝혔다. 제시한 프로세스 모니터링 기법은 리눅스상에 실제 구현하였으며, 커널상에서 응용프로그램의 수정없이 자동으로 프로세스 상태 및 주기성을 모니터링하므로 적절한 DVS 알고리즘에 적용 가능하다. 향후에는 주기적 프로세스의 주기를 훼손하지 않으면서도 비주기적 프로세스의 급변하는 상태변화를 고려한 DVS 알고리즘을 개발할 예정이다.

참고문헌

[1] L. Benini, A. Bogliolo and G. De Micheli, "A Survey of Design Techniques for System-Level Dynamic Power Management", *IEEE Transactions on VLSI*, June. 2000, pp. 299~315.

[2] K. Flautner and T. Mudge, "Vertigo: automatic performance -setting for Linux", *Proceedings of 5th Symposium on OSDI*, Dec. 2002, pp. 105-116.

[3] L. Benini, A. Bogliolo, S. Cavallucci, and B. Ricco, "Monitoring System Activity for OS-Directed Dynamic Power Mangement", *Proceedings of the ISLPED 1998*, Nov. 1999.

[4] H. Zeng, X. Fan, C. Ellis, A. Lebeck, and A. Vahdat, "ECOSystem: Managing Energy as a First Class Operating System Resource", *Proceedings of ASPLOS 2002*, Oct. 2002, pp. 123-132.