리눅스 기반의 멀티레벨 런큐 스케쥴링

박동국° 윤상용 이용우
서울시립대학교 전자전기컴퓨터공학부
dongugui@kolidong.net°, racc@metalab.uos.ac.kr, ywlee@uos.ac.kr

A multi-level Run-Queue Scheduling System of Linux

Dong-Kook Park° Sang-Yong Yoon Yong-Woo Lee

Department of Electrical & Computer Engineering, The University of Seoul
90 Cheonnong-Dong, Dongdaemun-gu Seoul S.Korea
Phone: (02) 2210-2827

요 약

최근 멀티미디어 데이터 서버로 리눅스 시스템을 쓰는 경우가 많아 졌다. 이 경우, 멀티미디어 데이터에 관한 서비스를 효율적으로 제공할 수 있어야 한다. 이를 위하여, 본 논문에서는 기존의 리눅스 scheduling 방식이 갖는 단일레벨 run-queue 구조를 변형한 다중레벨 run-queue를 제안하였다.

기존의 단일레벨 run-queue에서는 queue 내에 프로세스의 수가 많아질수록 검색시간이 길어지는 단점이 있다. 본 논문에서는 기존의 run-queue를 여러 단계로 나누고, scheduling 과정에서 상위 queue부터 프로세스가 존재하는지를 조사하도록 스케줄러를 변형하였다. 따라서, 상위 queue에 프로세스가 있는 경우에는 하위의 queue는 더 이상 조사할 필요가 없게 되므로, 결과적으로 검색시간을 줄일 수 있게 된다. 한편, 다중 레벨의 run-queue를 사용할 경우, 이를 관리하기 위한 오버헤드가 별도로 발생한다. 본 논문에서는, 제안한 다중레벨의 run-queue 시스템의 성능을 최적화하기 위하여, queue의 적절한 개수 선정 및 각 프로세스를 어떤 queue에 넣을 것인지를 결정하는 것이 성능에 미치는 영향에 대하여 실험적으로 연구하여 새로운 스케쥴러의 성능을 기존 스케쥴러와 비교 분석하였다.

본 논문에서 제안한 멀티레벨 run-queue를 사용함으로써, 각 queue의 스케쥴링 정책(policy)과 관련 파라메터 값을 독자적으로 변화시킬 수 있다. 따라서, 여러 가지 상황에 적합한 스케쥴링을 각각의 경우에 맞게 최적화하는 것이 손쉬워 지므로 여러 분야에서 매우 유용하게 쓰일 것이다.

Keywords : Multi-level scheduler, LINUX

1. 서론

기존의 리눅스 scheduling 방식은 단일레벨의 run-queue를 가지고 있고, scheduling 과정에서 run-queue 내의 모든 프로세스에 대해서 priority값을 조사하고, 이 중 가장 높은 priority를 갖는 프로세스를 선정하는 방식으로 작동한다.[1][2]

본 연구에서는 기존의 run-queue를 여러 단계로 나누고, scheduling 과정에서 상위 queue부터 프로세스가 존재하는지를 조사하도록 변형해 보았다. 그렇게 되면, 상위 queue에 프로세스가 있는 경우에는 하위의 queue는 더 이상 조사할 필요가 없게 되므로, 검색시간을 줄일 수 있게 된다. 대기상태에 있는 프로세스의 수가 많아질수록 검색 시간이 개선되는 효과는 더욱 커질 것이다.

그런데, 다중 레벨의 run-queue를 사용할 경 우 이를 관리하기 위한 부가적인 오버헤드가 발생됨을 또한 고려해야 한다.

이 둘을 적절히 고려한 다중레벨 run-queue를 만드는 것은 쉽지가 않다. 그러나, 다중레벨 run-queue를 가짐으로써, 리눅스 시스템은 보다 진보되고, 복잡한 대형 시스템의 스케쥴링 방식에 필적하는 우수한 스케쥴링 관리가 가능해진다.

2. multi-level queue의 구현

2.1 개요

scheduling 과정에서, 프로세스의 우선권에 상관없이 일단 run-queue를 모두 한번 검색하는 과정이 필요하게 된다.

run-queue의 모든 프로세스를 조사하는 overhead를 줄이기 위해서 실제 run-queue를 여러 개의 level-queue로 나누고, 검색 시 상위 queue부터 조사하기 시작하면, 해당 queue에 프로세스가 존재하는 한, 하위 queue들을 검색할 필요가 없게 된다.

level-queue를 사용하면 run-queue를 부분적으로 우선 순위 정렬하는 효과를 얻을 수 있고, 적당한 파라메터 값

(level의 수, 구분기준)의 조정을 통해서 전체적인 scheduler의 성능 향상을 꾀할 수 있다.

2.2 multi-level queue의 설계

전체 queue는 3개로 나누고, 각 queue에는 다음과 같은 프로세스들이 들어간다.

-level 1
: SCHED_FIFO, SCHED_RR을 policy로 갖는 프로세스들
-level 2
: SCHED_OTHER를 policy로 갖는 프로세스들 중 counter값이 MEASURE 값보다 큰 프로세스들
-level 3
: SCHED_OTHER를 policy로 갖는 프로세스들 중 counter값이 MEASURE 값보다 작은 프로세스들

위에서 사용한 MEASURE값은 define해준 값이고, 이 값을 여러 가지로 조절하면서 성능테스트를 한 후, 적당한 값을 찾아보았다.

프로세스가 run-queue에 들어갈 필요가 있을 때, 우선 policy값을 조사해서 SCHED_OTHER가 아니면 level1-queue에 넣고, SCHED_OTHER라면 counter값을 MEASURE와 비교해서 크면 level2-queue에, 작으면 level3-queue에 넣는다.

프로세스가 run-queue에서 제거될 때는 프로세스가 각 level-queue 중 어디에 있는지 먼저 조사하고, 발견되면 해당 queue에서 제거한다. 이때, level1-queue에 들어있는 프로세스의 수는 통상적으로 많지 않으므로, level2, level3, level1의 순서대로 조사를 하도록 구현하였다.

[그림 1]에 전체적인 순서도를 보였다.

3. 실험결과 및 분석

3.1 실험에 사용된 work-load

일반적으로 사용자들이 많이 사용하는 프로그램들을 선택하여서 work-load를 구성하였다. work-load를 구성하는 프로그램들은 다음과 같다.

가) gcc : 간단한 게임 프로그램의 컴파일
나) cp : 파일의 복사 (크기별로 3가지)
다) cat : smb mount된 파일의 내용 출력
라) ls : 1000개 파일의 정보/목록 출력
마) ping : localhost로의 ping
바) tar : tar로 묶인 파일의 해제

3.2 측정에 사용된 커널의 종류

가) original : 기존의 리눅스 커널 (2.4.3)
나) mod-3 : MEASURE값을 3으로 컴파일
다) mod-4 : MEASURE값을 4로 컴파일
라) mod-8 : MEASURE값을 8로 컴파일
마) mod-12 : MEASURE값을 12로 컴파일

3.3 성능 측정 방식

우선 각 프로그램들로 구성된 work-load를 수행시키면서 각 프로그램별로 총 수행에 걸리는 시간을 측정하고, 그 시간을 모두 더해서 스크립트의 총 수행시간을 구한다.

각 커널별로 위의 과정을 수회 반복해서 평균적인 분포를 구하였다.

테스트는 각 커널별로 재부팅 후 바로 수행하였고, 테스트 수행 중에는 기본적인 데몬을 제외한 다른 프로그램은 수행하지 않았으며, 각 반복 간에 시간 간격은 최소화 하였다.

3.3 테스트 결과/분석

[그림 2]에 최초 10회 반복 수행결과를 보였다.

테스트를 처음 시작할 때는 캐쉬에 관련 정보가 없기 때문에 시간이 좀 오래 걸리지만, 2회부터 반복할 때에는 캐쉬 정보를 이용하기 때문에 속도가 빨라진다.

따라서, 본 논문에서는 2~10회에 대한 내용만 검토했다.

측정 결과, 멀티레벨로 구성한 커널들이 기존의 스케줄러보다는 대략 1-2초 정도 수행시간이 줄어서, 약간 향상된 성능을 보여주었다.

4. 결론

본 논문에서는 기존의 리눅스가 사용하는 단일레벨 run-queue 구조를 변형한 멀티레벨 run-queue를 제안하였다.

단일레벨 run-queue 시스템이 갖는 많은 스케줄링상의 제약과 어려움이 멀티레벨 run-queue 시스템에서는 획기적으로 개선이 된다. 즉, 각각의 queue마다 별도의 스케줄링의 적용이 가능하게 되므로, 그 효용성은 이루 말할 수 없다.

멀티레벨 run-queue를 가짐으로써, 멀티 queue를 관리하는 오버헤드가 추가로 발생하지만, 프로세스의 수가 많아질수록 검색시간이 단일 레벨 run-queue의 경우보다는 줄어든다. 따라서, 다중레벨 run-queue 시스템은 어떻게 구현하고 관리하는가에 따라 그 성능이 많이 차이가 날 수 있다.

본 연구에서 위의 사항들을 고려하여 성능 측정을 해 본 결과, 기존의 리눅스 스케줄러의 성능과 비교했을 때, 떨어지지 않는 성능을 보이는 것을 알 수 있었다. 본 논문의 내용을 기반으로, 추후 개선을 통하여 더욱 효율적인 스케줄러를 개발할 수 있으리라 생각한다.

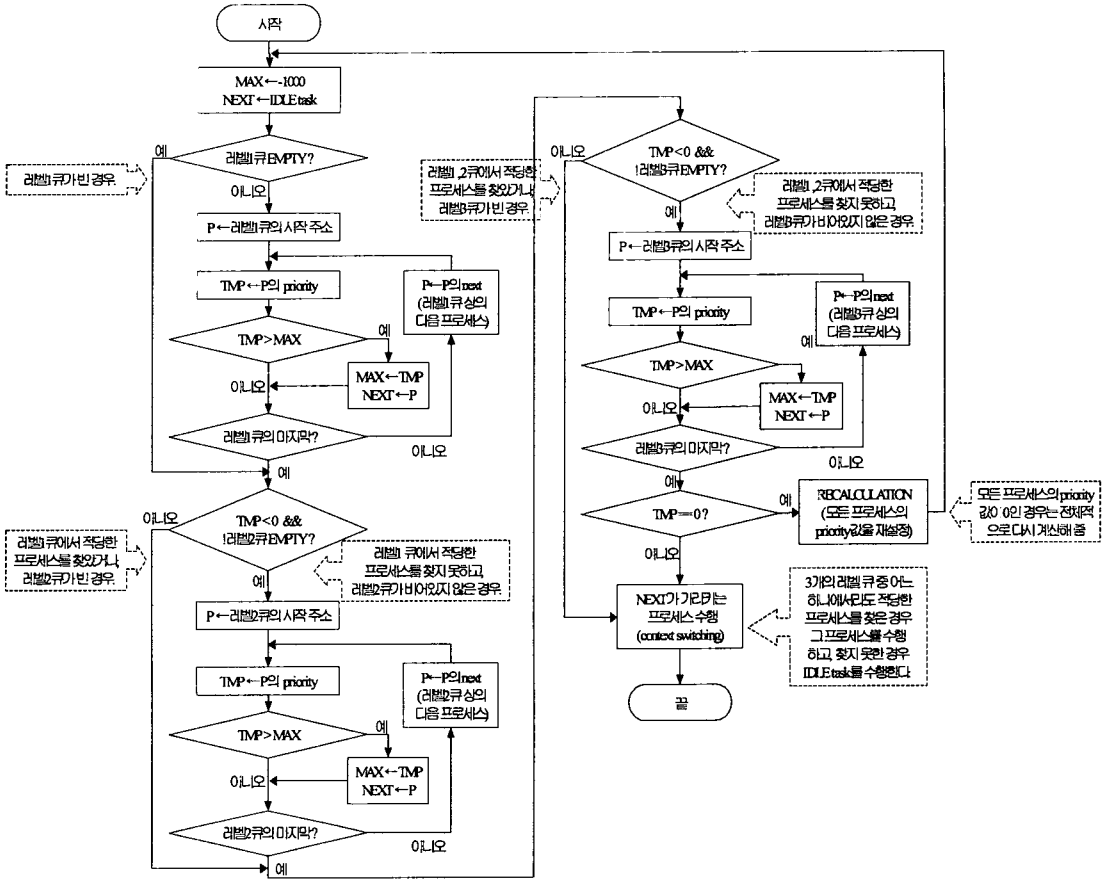
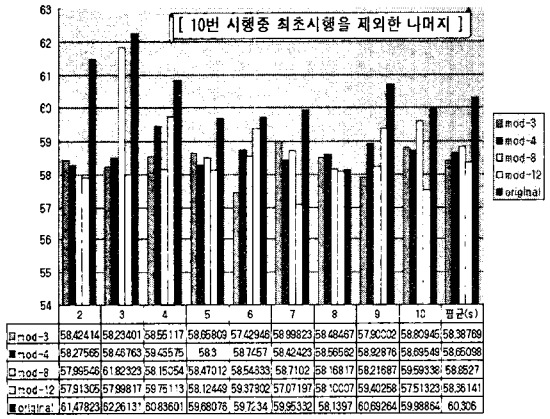


그림 1. 멀티레벨 run-queue 방식의 scheduling 과정



	2	3	4	5	6	7	8	9	10	평균(s)
mod-3	58.42414	58.23401	58.55117	58.65809	57.42946	58.99823	58.48467	57.90032	58.80945	58.38789
mod-4	58.27585	58.48763	59.43575	583	587457	58.42423	58.56582	58.92876	58.69548	58.65098
mod-8	57.99546	61.82323	58.15054	58.47012	58.54333	587102	58.16817	58.21687	59.59338	58.8527
mod-12	57.91305	57.99817	59.75113	58.12449	59.37902	57.07197	58.10037	59.40258	57.51323	58.35141
original	61.47823	62.26131	60.83601	59.68076	59.7234	59.95332	58.1397	60.69264	59.98864	60.306

그림 2. 최초 10회 수행 결과

참고문헌

[1]. Daniel P. Bovet & Marco Cesati, "Understanding the Linux Kernel", O'REILLY, 2000.

[2]. Scott Maxwell, "Linux Core Kernel Commentary 2nd Edition", 2001, Coriolis Open Press.

[3]. M Beck 외 5명, "Linux Kernel Internals", 1999, Addison-Wesley.

[4]. David A Rusling, "The Linux Kernel", 1999.(PDF)

[5]. Randall L. Hyde, "The Art of Assembly Language" (PDF)