2001 년 한국정보처리학회 추계 학술발표논문집 제 8 권 제 2 호

# 리눅스 기반 고정 스케줄링 시간을 갖는 스케줄러의 설계 및 구현

정영준*, 이형석*, 김흥남*
*한국전자통신연구원

e-mail : jjing@etri.re.kr

# Design and Implementation of Fixed Scheduling Time Scheduler based on Linux

Young-Jun Jung*, Hyung-Suk Lee*, Jae-Ho Lee*
*Electronics and Telecommunications Research Institute

요 약

인터넷 정보가전분야에 적용되는 내장형 리눅스 시스템은 제어장치, 휴대형 단말기나 홈서버등 용도에 따라 시간 제한과 관련한 서비스를 지원해야 하므로 실시간성을 가져야 한다. 그러나, 현재 적용되고 있는 대부분의 내장형 리눅스(Embedded Linux) 시스템은 표준 리눅스 시스템을 참조하여 구축되어, 리눅스의 스케줄링 구조에 따라 실시간 태스크라 할지라도 작업 수행시간에 대한 예측성(Predictability)이 떨어져 실시간성을 보장할 수 없다. 본 연구는 리눅스의 스케줄링 기법을 비트맵(Bitmap)을 이용한 기법으로 수정하여 실시간 태스크들에 대한 고정 스케줄링 시간을 갖는 스케줄러(scheduler)를 구현했으며, 시뮬레이션을 통한 표준 리눅스 스케줄러와 구현된 스케줄러의 비교 자료를 제시했다.

## 1. 서론

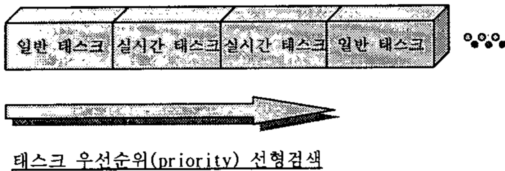
최근 연구가 활발해지고 있는 인터넷 정보가전 분야의 인터넷 정보가전(Internet Appliance)이라 함은 유. 무선 정보통신망에 연결돼 데이터 송수신이 가능한 디지털 TV 와 인터넷 냉장고, 디지털 다기능 디스크(DVD), 디지털 비디오 등과 같은 차세대 가전기기를 말한다. 또한, 최근 인터넷 정보가전분야는 앞서 열거한 인터넷 정보가전기기 들을 제어하기 위한 수단으로 휴대용 단말기, 셋톱박스, 웹패드, 홈서버, 홈시큐리티 등 광범위하게 연구되고 있다. 이러한 제어수단으로 사용되는 각종 제어장치에는 데스크톱 컴퓨터와는 달리 운영체제와 제어소프트웨어가 내장되어 들어가야 한다. 인터넷 정보가전기기에 사용될 운영체제는 제한된 자원 내에서 사용되어야 하며, 주로 사용되는 분야가 감지기, 스위치, 버튼등과 같은 제어장치로 사용되므로 수행되는 작업들이 어떤 시간적인 제한을 가지고 수행되고, 그 각각의 시간적인 제한과 작업 수행 결과의 정확도에 대한 보장을 해줄 수 있는 실시간성을 가져야 한다. 그러나, 현재 업계에서 실용화되고 있는 내장형 리눅스는 그의 참조 모델이 표준 리눅스이므로 실시간성이 떨어져 인터넷 정보가전 분야의 운영체제로는 부족하다. 그 이유는 표준 리눅스 커널은 전통적인 유닉스의 monolithic 커널 형태를 이루고 있어 인터럽트에 대한 응답 지연 시간이 길고, 태스크에 대한 스케줄링의 핵심인 스케줄러가 실시간성의 필수 요소인 작업 수행시간의 예측성(Predictability)이 떨어지는 구조로 되어 있으므로, 실시간 태스크라 할지라도 항상 제한된 시간 내에서 스케줄링 된다고 보장할 수 없다. 따라서, 본 논문[1] 은 시스템의 실시간성과 관련한 리눅스 커널의 문제점 중 스케줄러 자체의 작업 수행시간의 예측성을 높이기 위한 측면에 집중하여 이에 대한 개선을 통한 내장형 리눅스에 적

[1] 본 논문은 정보통신부 정보가전용 RTOS 커널 개발 중 프로세스 기반 RTOS 개발의 일부 결과물 임.

용하는 것이 목표이다.

## 2. 연구 배경

실시간성 지원이 요구되는 인터넷 정보가전의 경우에 표준 리눅스를 참조하는 내장형 리눅스를 사용하는 것으로는 실시간 태스크에 대한 스케줄링 시간을 일정하게 보장할 수 없다. 그 이유는 표준 리눅스 커널 스케줄러의 구조가 [그림 1] 과 같이 실시간 태스크와 일반 태스크를 분리하여 다중 대기큐를 구성하지 않고 하나의 대기큐(runqueue)에 적재하여 사용하기 때문이다.



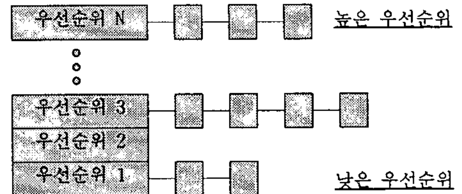

[그림 1] 리눅스 대기큐의 구조

이 경우에는 스케줄러가 호출되어 시스템내의 태스크들에 대한 재스케줄링이 이루어질 때 루프를 돌면서 대기큐에 적재된 모든 태스크들을 검사해야 하므로, 대기큐에 적재된 태스크 수에 따라 스케줄링 시간의 변화가 생길 수 밖에 없다. 따라서, 이런 구조로는 태스크 수에 상관없이 실시간 태스크들에 대한 고정적인 스케줄링 시간을 갖는 스케줄러를 기대할 수가 없게 된다. 그러므로, 본 연구는 이런 문제점을 해결하기 위해 실시간 태스크의 우선순위에 기반한 다중수준(Multi-Leveled)의 대기큐를 구현하고, 이를 선택하기 위한 기법으로 비트맵(bitmap) 알고리즘을 이용하여 실시간 태스크의 스케줄링 시간의 예측성을 높이도록 했다.

## 3. 스케줄러의 설계 요구사항

스케줄러는 다음과 같은 요구 사항을 고려하여 설계 및 구현됐다.

1. 리눅스의 기본 자료구조를 유지하여야 한다.
   - 리눅스의 기본적인 자료구조를 변형하지 않고 필요한 자료구조를 추가하여 구현하여야 한다.
2. 리눅스의 기본 스케줄링 정책(policy)가 유지되어야 한다.
3. 구현된 후에도 POSIX 표준 API(Application Programming Interface)를 따를 수 있어야 한다.
   - 구현된 후 사용자가 사용함에 있어 표준 리눅스 커널의 스케줄러와 비교하여 투명성(transparency)을 제공하며, 표준 리눅스 커널이 지원하는 스케줄링 관련 각종 POSIX API 를 그대로 이용할 수 있도록 하여야 한다.
4. 실시간 태스크를 위한 우선순위 기반의 다중수준의 대기큐를 생성하여 이용하여야 한다.
   - 표준 리눅스 스케줄러의 문제점인 하나의 대기큐로 구성된 구조보다 [그림 2] 와 같은 우선순위 순서대로 태스크의 적재가 가능한 대기큐의 구조를 가져야 한다.




[그림 2] 우선순위 기반 다중수준 대기큐

5. 비트맵(bitmap) 구조를 이용하여 최악의 상태를 가정한 스케줄링 시간의 예측이 가능하여야 한다.
   - 태스크 수에 상관없이 일정한 스케줄링 시간을 가질 수 있어야 한다.
6. 표준 리눅스 커널의 실시간 태스크 우선순위의 범위를 확장한다.
   - 표준 리눅스 커널의 실시간 태스크의 우선순위 범위는 1~99 이며, 우선순위 0 은 일반 태스크를 의미한다. 이에 실시간 태스크의 우선순위를 1~127 로 확장한다.
7. 구현된 스케줄러가 커널의 환경설정을 이용하여 선택 및 선택가능하지 않도록 할 수 있어야 한다.
   - 사용자의 편의에 따라 선택 가능하도록 이 기능을 커널의 빌드 환경설정시 선택사항으로 제공하여야 한다.

## 4. 스케줄러의 설계 및 구현

설계 및 구현은 위의 요구사항이 기준이며, 이에 부합하도록 했다.

### 4.1. 자료 구조 설계

내부 자료 구조 중 핵심인 다중수준 대기큐는 128 개의 우선순위를 가지도록 설계했다. 이는 표준 리눅스 커널의 99 까지의 실시간 태스크 우선순위가 127 로 확장됨을 의미한다. 본 연구에서 사용한 비트맵은 4 개의 워드(word)를 이용하여 127 까지의 우선순위를 가지는 다중수준 대기큐에 맵핑했다. 대기큐는 이중연결 리스트로 구현하여 태스크의 삽입과 삭제시 부담(overhead)가 생기지 않도록 설계했다. 새로운 자료구조를 이용하여 다중수준 대기큐를 설계했으므로, 표준 리눅스의 기본 틀과 POSIX 의 스케줄링 정책과 관련한 표준 API 를 그대로 유지하기 위해서는 스케줄링 정책을 반영할 수 있는 함수가 필요하다. 이는 다중수준 대기큐에 있는 태스크들에 대한 스케줄링 정책을 확인 및 설정할 수 있고, 우선순위도 설정할 수 있도록 한다. [표 1]은 기본적인 자료 구조 및 구현 함

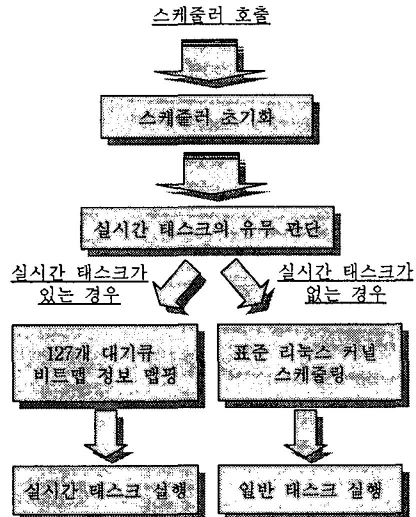
수이다.

```
#define MAX_PRI 127 /* 가장 높은 우선순위 정의 */
/* 대기큐의 속성 정의 */
static struct rq {
        struct task_struct *next_run ;
        struct task_struct *prev_run ;
} rq[MAX_PRI + 1] ;
/* 대기큐의 정의 */
struct list_head {
     struct list_head *next, *prev;
};
static struct list_head rq[MAX_PRI+1]
/* bitmap 자료 구조 정의 */
static struct {
        int guard ;
        /* 128 level priority support */
        int rq_bit_ary[(MAX_PRI/32) + 1] ;
} rq_bits = {-1, {0, 0, 0, 0}} ;
#define rq_bit_map rq_bits.rq_bit_ary
/* 대기큐의 가장 높은 우선순위를 가지는
태스크의 우선순위 표시 */
static int high_prio ;
/* 각 우선순위별 대기큐 헤더의 위치정보 반환 정의 */
#define Rdy_Q_Hed(pri) &rq[pri]
/* 우선순위를 변경할 수 있는 함수 */
static void newprio_executing(struct task_struct
        *tptr, int newprio) ;
```

[표 1] 자료 구조 및 구현 함수

## 4.2. 스케줄링 기법

기본적으로 리눅스의 스케줄링 정책인 SCHED_OTHER, SCHED_FIFO, SCHED_RR 을 그대로 적용 가능하도록 한다. 위의 스케줄링 정책 중 SCHED_FIFO 와 SCHED_RR 은 리눅스 시스템에서 실시간 태스크에 대한 스케줄링 정책이며, SCHED_OTHER 보다 우선하여 처리된다. SCHED_OTHER 는 리눅스의 일반적인 태스크에 설정되며, 대부분의 태스크가 여기에 속한다. SCHED_FIFO 는 대기큐에 먼저 적재된 태스크를 우선하여 처리하며, 더 높은 우선순위를 가지는 태스크나 처리중인 태스크가 블록이 되는 경우에만 다른 태스크에게 스케줄링 된다. SCHED_RR 의 기본적인 정책은 SCHED_FIFO 와 같으나, 같은 우선순위를 가지는 태스크들에 한하여 각 태스크에게 제한된 시간(time slice)가 할당되어 그 시간 동안만 CPU 의 서비스를 받도록 하는 것이다. 전체적인 스케줄링의 제어 흐름은 [그림 3]과 같다. 스케줄러가 호출되면 표준 리눅스 스케줄러의 초기화와 설계된 스케줄러의 추가된 자료 구조에 대한 초기화를 수행한다. 그런 후 현재 대기큐에 실시간 태스크의 존재 유무 판단을 해야 하는데 이 때, 위의 자료 구조 중 high_prio 의 값을 확인하여 이 값이 0 이 아닌 양수의 값을 가지면 실시간 태스크가 존재하는 것이며, 0 이면 SCHED_OTHER 의 스케줄링 정책을 갖는 일반 태스크만 존재하는 것으로 판단한다. high_prio 의 값은 현재 가장 높은 우선순위를 가지는 태스크가 존재하는 대기큐를 의미하며, 태스크가 대기큐에 적재 및 삭제되거나 우선순위가 변경될 때 수정될 수 있다. 이렇듯 실시간 태스크의 유무가 가려져 실시간 태스크가 존재하는 경우에는 실시간 태스크의 전체 우선순위와 동일한 128 개의 비트맵 정보와 맵핑시키는 스케줄링 루틴으로 넘어가고, 그렇지않은 경우에는 표준 리눅스 커널의 작업과 동일하며, 이 경우에 표준 리눅스 커널의 스케줄러는 SCHED_OTHER 로 표시된 일반 태스크만 수행 시킨다.

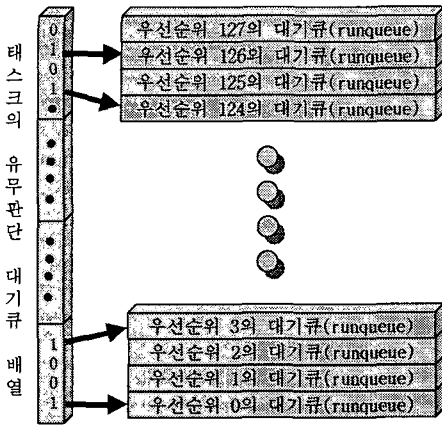



[그림 3] 설계된 스케줄러의 제어 흐름

본 스케줄러의 FIFO, Round-Robin, Other 스케줄링의 동작에서 가장 중요한 요소중의 하나는 CPU 의 서비스를 받기 위해 다중수준 대기큐에 적재되어 대기상태에 있는 태스크 중에서 CPU 의 서비스를 받게 할 가장 높은 우선순위의 태스크를 골라내는 일이다. 이를 위하여는 우선순위에 의거하여 대기큐를 정렬(sorting)한 후 head 나 tail 에 있는 태스크를 골라낸다. 그런데, 이런 방법을 이용하면 몇 가지 문제가 따른다. 대기 큐에 태스크를 넣고 빼는 동작이 일어남에 따라 매번 다시 정렬해야 하는 문제가 발생하므로 중요한 자원의 낭비를 유발한다. 더구나 이러한 방법에 의하면 태스크의 우선순위와 대기 큐에 있는 태스크의 수에 따라서 매번 처리시간이 달라진다. 이는 시스템의 작업 수행 시간의 예측성(predictability)를 저해하여 deterministic 한 실시간 시스템 구성에 악영향을 끼칠 수 있으므로 바람직하지 못하다. 따라서 본 스케줄러는 대기 큐에서 가장 높은 우선순위의 태스크를 골라내는 방법으로 매 스케줄링 처리시간이 일정할 수 있도록 비트맵(bitmap) 검색을 이용한다.

비트맵을 이용한 검색은 실시간 태스크를 관리하기 위한 두 가지의 주요한 자료구조를 이용하며, [그림 4]에 잘 나타나있다. 그 하나는 128 개의 대기 큐의 header 로 구성된 배열(array) 로서, 그 배열의 원소(element)는 실시간 우선순위를 의미한다. 두 번째는

스케줄러에게 실시간 대기 큐에 대기하고 있는 태스크가 있는지의 검사를 빨리 할 수 있도록 하는 4 개의 워드(word) 비트맵(128 bits) 자료 구조이다.

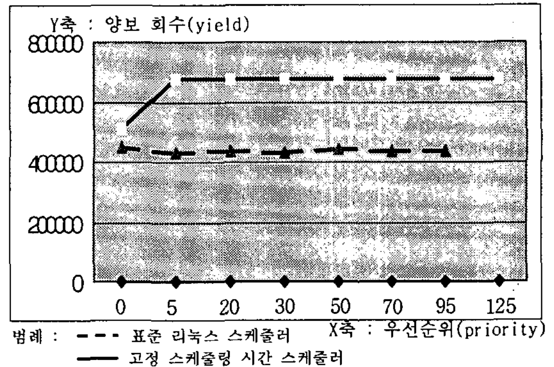



[그림 4] 비트맵 맵핑 구조

이 기법은 가장 높은 우선순위를 가지는 비어있지 않는 실시간 대기 큐를 찾아내기 위해서 처음부터 끝까지 선형적인 방법으로 찾을 필요가 없다. 단지 비트맵을 구성하는 4 개의 워드만 검사하여 항상 일정한 시간 내에 가장 높은 우선순위의 실시간 태스크를 찾아낼 수가 있도록 할 수 있는 것이다. 또한, 비트맵 검색시 검색 함수는 표준 리눅스에서 어셈블리로 구현된 함수를 사용하여 부담(overhead)을 최대한 줄였다. 실시간 태스크가 생성되어 대기큐에 삽입하는 과정은 할당된 태스크의 실시간 우선순위를 이용하여 비트맵 대기큐 배열 중 해당 우선순위의 대기큐에 태스크를 삽입한다. 그런 후 비트맵 대기큐 배열의 해당 비트의 값을 1 로 설정한다. 비트맵은 시작 비트를 우선순위 127 과 맵핑 시킨다. 위의 [그림 4]에서는 비트맵 배열의 두 번째 비트가 1 로 설정되어 우선순위 126 에 대기하고 있는 실시간 태스크가 있음을 알리고 있다. 스케줄러가 가장 높은 우선순위의 실시간 태스크를 찾아내어 실행시키게 되면, 같은 수준의 다른 실시간 태스크가 있는지를 검사한 뒤, 없는 경우에는 비트맵의 해당 비트를 0 으로 설정한다.

## 5. 스케줄러의 시뮬레이션

구현된 고정 스케줄링 시간 스케줄러와 표준 리눅스 커널 스케줄러의 비교를 위해 시뮬레이션 했다. 시뮬레이션 환경은 펜티엄 III 600 MHz 컴퓨터에서 리눅스 커널 버전 2.4.4 를 이용했으며, 시뮬레이션 당시 시스템 내부 태스크 수는 같도록 조정하였다. 스케줄링 방법은 10 개의 태스크를 생성 후 주어진 시간(1 초) 안에 스케줄링 정책을 바꾸어가며 CPU 를 양보(yield)하는 회수를 측정한다. 이는 곧 단위 시간 안에 얼마나 많이 문맥교환(context switch)가 일어날 수 있는지를 의미하는 것이며, 최종적으로는 스케줄링 시간이 어떤 것이 더 짧게 일어나는지를 알 수가 있는 것이다. 이때, 시뮬레이션은 실시간 우선순위와 스케줄링 정책을 변경해가며 시뮬레이션 했다. 시뮬레이션 결과는 [그림 5]에 나타나있다. 우선순위를 의미하는 x 축의 값이 0 인 것은 일반적인 태스크로 생성한 것이며, 0 이 아닌 것은 실시간 태스크로 생성한 것으로 0 이 아닌 값에서의 비교가 중요하다. 시뮬레이션은 각 우선순위 별로 10 회씩 실시한 평균값이다.




[그림 5] 스케줄러 성능 비교

시뮬레이션 결과는 일반 태스크로 설정한 경우에는 두 스케줄러가 비슷한 성능을 나타냈으나, 실시간 태스크로 설정하여 시뮬레이션 한 경우에는 고정 스케줄링 시간 스케줄러가 약 55% 가량 스케줄링 성능이 향상된 것으로 시뮬레이션됐다.

## 6. 결론 및 향후 연구방향

표준 리눅스 커널의 핵심인 스케줄러의 형태와 스케줄링 관련 POSIX API 를 유지하며 실시간 태스크를 위한 비트맵 스케줄러 코드를 구현함으로써 실시간 태스크의 스케줄링 시간을 어느 정도 보장하여 인터넷 정보가전에 적용시 시스템 내부 실시간 태스크의 수행 시간에 대한 예측성(Predictability)을 표준 리눅스 커널을 참조한 임베디드 리눅스보다 높일 수 있음을 확인했다. 그러나, CPU 의 처리속도, cache hit rate, 네트워크, 입출력 등 시스템의 모든 부분에 걸쳐 실시간 태스크의 처리 시간을 보장하는 것은 매우 어려운 문제이며, 기존의 연구들도 각 부분별로 이에 대한 연구가 진행되고 있는 추세이다. 본 연구는 리눅스 커널의 스케줄러에 한하여 실시간 태스크 스케줄링의 예측가능성을 높이는데 중점을 두었으며, 향후 연구과제로 많은 연구에도 불구하고 부족한 표준 리눅스를 이용한 실시간성 지원에 집중할 것이다.

## 참고문헌

[1] M BECK, H BOHME, M DZIADZKA, U KUNITZ, R MAGNUS, D VERWORNER "Linux Kernel Internals" 2nd Ed, Addison-Wesley

[2] Montavista Software co. "Real-Time Scheduler for Hard Hat Linux" white paper

[3] Bill O. Gallmeister "POSIX.4.: Programming for the Real World" O'Reilly & Associates, Inc.