임베디드 리눅스 커널의 실행속도 향상을 위한 메모리 맵 분석

이두완*, 장경식*

*한국기술교육대학교

The Analysis of Memory Map for Improving the Execution Speed of Embedded Linux Kernel

Doo-Wan Lee*, Kyung-Sik Jang*

*Korea University of Technology and Education

e-mail : neomenie@kut.ac.kr, ksjang@kut.ac.kr

요 약

본 논문에서는 임베디드 리눅스 시스템 성능 향상을 위한 방안으로 리눅스 커널 메모리 맵을 분석하였다. 안정성과 다양한 H/W 플랫폼을 지원하고 범용 시스템에 최적화 되어 있는 리눅스 커널 메모리 맵은 부팅시간과 효율적인 시스템 자원 활용에 중요한 역할을 담당하므로 자원 제한적인 임베디드 리눅스 시스템의 성능 향상을 위해 커널 메모리 맵의 분석이 요구된다. 분석결과, 리눅스 커널 메모리의 할당 위치에 따라 임베디드 리눅스 시스템의 부팅속도 및 메모리 효율성의 향상을 확인하였다. 그러므로 본 논문에서 제안한 부트로더 및 커널 메모리 할당 방안이 임베디드 리눅스 시스템의 메모리 활용성 향상에 적합할 것으로 사료된다.

ABSTRACT

In this paper, the Linux kernel memory map was analyzed as the approach to Improving performance for Embedded Linux system. Since the Linux kernel memory map supporting a stability and various H/W platforms and in which it becomes to the general purpose system with optimization manages the role of being important in the booting time and the efficient system utilization of resources, the analysis of the kernel memory map is required for the performance improvement of the Embedded Linux system in which it is restrictive the resources. According to the analysis result, and of the Linux kernel memory, the booting speed of and improvement of the memory efficiency were confirmed. It is therefore considered that the proposed in this paper and kernel memory allocation method are suitable to the memory availability improvement of the Embedded Linux system.

키워드

Embedded Linux, Embedded System, Linux Kernel, Memory map

Ⅰ. 서 론

임베디드 시스템은 마이크로 프로세서/마이크로 컨트롤러를 내장하여 원래 제작자가 의도한 미리 정해진 특정한 기능을 수행하도록 프로그램이 내장되어 있는 시스템을 말한다.[1] 일반적으로 보다 큰 시스템의 일부이거나 독립된 시스템으로 특별한 업무를 수행하거나 사용자가 임의로 정한 업무를 수행하는 곳에 사용되며, 개인 휴대 정보 단말, 지리 정보 시스템, 의료 정보 단말, 정보가전, 게임기기, 자동차, 항공기 및 우주선, 의료 및 산업 원격 조종장비 등의 시스템을 예로 들 수 있다.[2]

임베디드 시스템의 운영체제로 리눅스를 채택한 이유는 리눅스의 모든 소스가 공개되어 있기 때문에 안정적이고 다양한 서비스를 제공함으로써 가격대비 품질 경쟁력에서 다른 운영체제를 사용한 임베디드 시스템보다 선택적으로 우위에 위치하고, 다양한 응용 S/W개발이 용이하기 때문에 확장 가능한 임베디드 시스템에도 적용이 가능한 장점이 있다. 최근에는 임베디드 리눅스의 이러한 장점 때문에 많은 임베디드 시스템의 운

영체제로 리눅스가 채택되어 지고 있는 실정이다.

본 논문에서는 임베디드 리눅스 환경에서 시스템의 속도를 향상시킬 수 있는 방법을 찾아내기 위해 시스템의 부팅과정과 시스템 동작시 적용되는 메모리의 레이아웃을 분석하여 부팅단계에서 소요되는 시간을 단축하고, 시스템 실행 시 소요되는 실행 시간을 단축 시킬 수 있는 방안을 찾도록 한다.

본 논문의 구성은 2장에서 부팅과정과 부팅과정 중 각 단계 메모리 맵의 구조를 분석하고, 3장에서 결론 및 향후 과제를 기술한다.

II. 본 론

1. 임베디드 시스템의 부팅과정

부팅은 컴퓨터의 시동을 뜻하며, 보조기억장치(bootable device:주로 플로피디스크 또는 하드디스크)를 사용하여 컴퓨터가 동작할 수 있도록 시스템에 운영체제를 불러 들여 작동을 준비하는 작업이다. 즉, 컴퓨터 시스템을 시동하거나 초기 설정하는 것을 뜻한다.

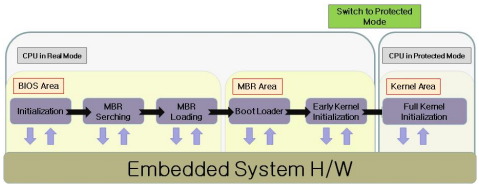

그림 1. 리눅스 부팅과정

메인보드에 전원이 공급된다면 CPU를 구동하기 전에 BIOS(Basic Input/Output System)를 초기화 하고 CPU 구동을 시작하게 된다. CPU의 구동이 제대로 시작 되었다면 CPU는 Bootstrap Process (BSP)를 실행하여 부팅작업을 시작하게 된다.

부트 시퀀스(Boot Sequence)는 시스템에 전원이 들어오거나 시스템을 재시작하는 경우 커널을 로드하기 까지의 일련의 과정이라고 할 수 있다. 이 과정 중에는 보조기억장치에서 커널을 메모리에 적재하는 등의 여러가지 작업들이 있지만, 그 중에서 가장 선행되는 작업은 BIOS에 의해 행해지는 검증 작업이다. BIOS는 IBM 호환 기종의 PC에서 전원이 들어오는 즉시 구동되는 플래시메모리(Flash Memory)내의 펌웨어(Firmware)이며, 흔히 ROM BIOS라고도 부른다.[9]

CPU에 의해 첫 번째로 실행되는 명령어는 Reset Vector라 불리는 16 byte 코드에서부터 시작된다. 이 Reset Vector는 BIOS entry point에 매핑된 메모리위치(JUMP to 0xF0000)로 점프하라는 명령어가 포함되어 있다. 메모리 하위부분에 사용되는 0x00000000에서 0x000A0000(640KB)까지는 OS가 사용하는 Data영역이 된다. 보통 0x000A0000 상위영역을 시스템영역이라 하고 하위를 OS Data 영역이라고 말하기도 한다.[13]

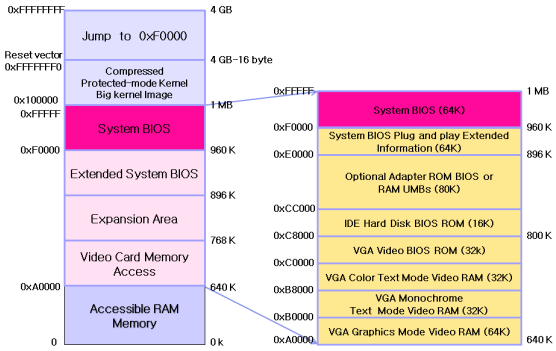

그림 2. BIOS POST 과정 메모리 맵

2. POST 과정수행

POST(Power-On Self-Test)는 시스템 내의 장치들을 검사하여 오류가 있는지를 찾아내며, 문제가 발견되면 시스템을 정지시킨다. POST 수행 결과 H/W가 정상이라면 BIOS는 INT19(Interrupt 19)를 발생시켜 다음 과정을 진행 하도록 한다. 시스템 내의 출력장치(VGA)를 검색하고, 출력장치안의 다른 BIOS가 있는지 확인하여, 해당 BIOS를 호출하게 된다.[3] 출력장치 검사가 완료되면 기타 장치들의 추가 테스트를 진행한다. POST 과정을 완료하게 되면 BIOS는 부팅 가능한 장치에서 MBR을 검색하게 된다.

3. MBR(Master Boot Record)

MBR(Master Boot Record) 또는 파티션 섹터는 파티션된 보조기억장치(Ex. 하드디스크)의 첫 섹터 (섹터 0)인 512byte 시동 섹터이다.

그림 3은 MBR의 구조를 나타낸 것이다. 부트로더는 운영체제의 커널 이미지를 메모리로 읽어들이기 위해 BIOS에서 호출하는 프로그램이다. 하드디스크로 부팅하는 경우 첫 번째 섹터인 MBR에는 파티션 테이블과 시작할 운영체제가 들어있는 파티션의 위치를 읽어 들이는 작은 프로그램이 포함되어 있다. 이것이 부트코드이다. 이 코드의 역할은 파티션 테이블 중에서 활성(Active)플래그를 통해 부팅할 운영체제가 들어있는 파티션을 알아내게 된다.[10]

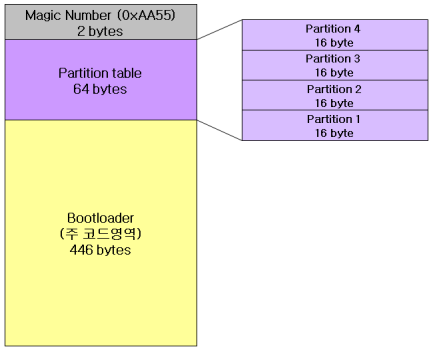

그림 3. MBR 구조

하나의 보조기억장치에서 주 파티션으로 4개까지 나눌 수 있기 때문에 각각의 파티션의 정보가 16 byte씩 4개로 저장되어 있다. 저장되어진 규칙은 그림 4와 같은 구조로 저장되어 있다.[14]

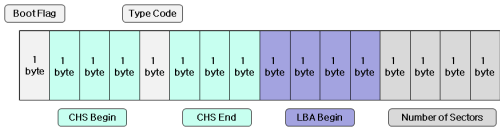

그림 4. Partition Table 구조(16byte)

Relative Offsets (within entry)	Length (bytes)	Contents
0	1	Boot Indicator (80h = active)
1 - 3	3	Starting CHS values
4	1	Partition-type Descriptor
5 - 7	3	Ending CHS values
8 - 11	4	Starting Sector (LBA)
12 - 15	4	Partition Size (in sectors)

표 1. Partition Table 크기[14]

① Boot Indicator
Booting이 가능한 파티션인지를 표시하며, 하나의 디스크에 꼭 1개만 존재한다. 부팅이 가능하면 0x80, 아니면 0x00으로 표시하면 된다.

② Starting CHS values / Ending CHS values
처음 및 마지막의 CHS(Cluster, Header, Sector)값을 표시한다. CHS의 범위를 넘어가면, FE FF FF로 기록된다.

③ Partition-type Descriptor
파티션의 유형을 정하는 것으로, 보통 File System의 종류를 표시한다.

코드	Partition 유형
0x07	NTFS, OS/2
0x05	Extended Partition
0x0B	FAT 32
0x0E	FAT 16
0x0F	Extended Partition, exp INT 13h
0x1B	Hidden FAT32
0x42	Secure File System
0x81	DR-Dos
0x82	Linux Swap partition, Prime or Solaris (Unix)
0x83	Linux native file Systems (EXT2/3, JFS, Reiser, xiafs, others)
0x1B	Hidden

표 2. Partition-type Descriptor

④ Starting Sector(LBA)
하드의 용량이 커지는 관계로 CHS가 모든 사이즈를 커버하기 힘들어지고, 내부구조를 알아야하는 단점 때문에 최근에는 LBA방식을 이용하고 있다. LBA는 CHS와는 달리 섹터만을 기록하고 있지만 BIOS에서 LBA를 지원하지 않을 경우 읽지 못하는 경우도 있다.

⑤ Partition Size(in sectors)
Starting Sector에서 마지막 섹터를 기록하지 않고 파티션 사이즈를 이용 한다. 그래서 마지막 섹터는 Starting Sector + Partition Size가 된다.

4. 부트로더 (Boot Loader : Grub)

부트로더(Boot Loader)란 간단히 말해서 컴퓨터를 켰을 때 가장 먼저 실행되는 프로그램이다. 부트로더는 OS의 커널을 로드하고 몇몇 커널 파라메터를 커널에 넘겨주는 일을 한다. GNU GRUB은 원래 처음에는 GNU Hurd를 위해 개발되었지만 그 막강한 기능 때문에 리눅스에서도 점차 LILO 대신 GRUB을 사용하는 사용자들이 늘고 있다. GRUB은 현재 리눅스를 비롯하여 많은 공개 운영체제와 Chain-Loading을 사용하는 상용 운영체제를 로드할 수 있다. GRUB의 가장 큰 특징은 파일시스템과 커널 포맷을 이해한다는 점이다. 따라서 디스크상에서 커널의 물리적인 위치를 알 필요 없이 단지 파일명과 커널이 위치하고 있는 파티션만 알고 있으면 커널을 로드할 수 있다. LILO의 경우 부트로더가 커널의 하드디스크상의 물리적인 위치를 알고 있어야 한다. 따라서 커널을 다시 설치할 때마다 'lilo'를 실행해 부트로더를 다시 설치해 주어야 하지만, GRUB의 경우 파일명만 알면 되기 때문에 파일명이 바뀌지 않는 한 다시 실행해 줄 필요가 없다.

현재 GRUB은 다음과 같은 기능들을 제공한다.

① a.out 포맷과 ELF포맷의 커널 사용가능
② Linux, FreeBSD, NetBSD, OpenBSD등 비-멀티부트 커널을 지원
③ 멀티플 모듈 로드가능
④ 텍스트 형식의 설정 파일을 제공
⑤ 메뉴 인터페이스를 제공
⑥ 유연한 커맨드라인 인터페이스를 제공
⑦ 다양한 파일시스템을 지원

GRUB의 부팅 단계는 2~3단계를 거치게 된다. 각 단계를 Stage라 부르며, 내용은 <표 3>와 같다.[12]

단계	내용
Stage 1	MBR에 의해 Loading되어 GRUB이 제어권을 갖게 된다.
Stage 1.5 (Optional)	Stage 2를 불러오기 위해 특별하게 사용되는 File System을 선택할 수 있다.
Stage 2	운영체제를 선택할 수 있는 메뉴 화면으로 진입하는 단계이다.

표 3. GRUB 단계

Stage 2 단계에서 커널을 선택하게 되면 부트로더는 메모리에 선택된 커널 압축이미지를 올리게 되고 지정된 위치에 압축을 풀 수 있도록 제어권을 커널로 넘겨 주게 된다.

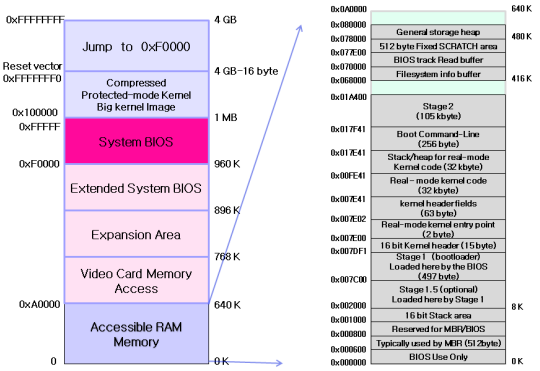

그림 5. GRUB 실행 후 메모리 맵

5. 커널

압축된 커널이미지에 따라 메모리의 위치가 달라지게 된다. 일반적으로 큰 크기로 압축된 커널이미지는 메모리의 상위영역(0x100000 : 1MB)에 위치하게 되고, 압축된 이미지를 풀기위해 준비해 주는 Setup code는 메모리의 하위영역에 위치하여 제어권을 압축 해제 코드인 head_32.S로 넘겨주게 된다. head_32.S는 압축을 해제하기 위해 misc.c 내부에 있는 decompress_kernel() 함수를 호출하여 압축을 해제하게 된다. 압축이 해제된 후 최종적으로 main.c에 포함되어 있는 start_kernel()을 실행하게 되는 것이다.[4]

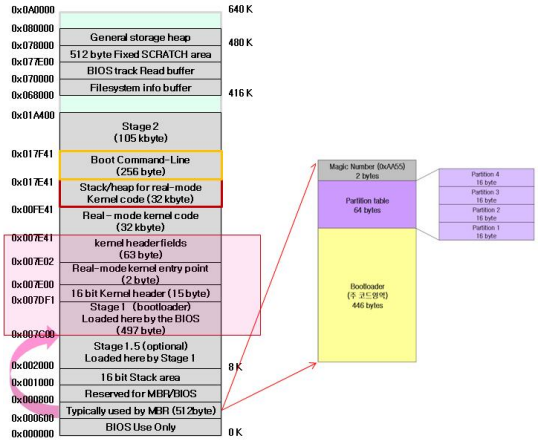

그림 6. 부팅 후 하위 OS 영역 메모리 맵

부팅이 완료된 후의 하위 OS영역 메모리 맵은 그림 6과 같다.

III. 결론 및 향후 연구과제

본 연구에서는 임베디드 시스템의 운영체제로 사용되고 있는 리눅스 커널의 실행속도를 향상시키기 위해 부팅과정과 부팅시 실행되는 메모리 맵을 분석하였다. 부팅시 실행 되는 메모리 맵의 분석결과 부팅속도를 향상시킬 수 있는 부분은 POST 과정의 최소화와 부트로더 모듈의 최소화, 마지막으로 커널 이미지의 최적화가 부팅 시간과 임베디드 시스템의 전체적인 실행속도를 향상시킬 수 있는 방법이다.

POST 과정시 메모리 맵에서 작동하는 H/W 테스트 과정을 최소화 시켜 메모리에 읽고 쓰는 시간을 줄일 수 있고, 부트로더를 포함하여 실행되는 모듈들을 최소화 하면 부팅 시간을 향상 시킬 수 있다. 또한 이러한 작업을 통해 부족한 임베디드 시스템의 자원을 효율적으로 활용할 수 있다는 결론이 나왔다.

운영체제의 부팅이 완료된 후에는 커널의 모듈 크기, 커널 환경설정 최적화를 통해 커널 압축이미지 크기를 줄이고, 파일 시스템의 최적화에 대한 연구가 과제로 남아 있다.

참고문헌

[1] 엄윤호, 박종득, “임베디드 시스템의 리눅스 커널 2.6 포팅”, 공주대학교, 2005

[2] 신광무, 박성호, 정기동, “임베디드 시스템에서 리눅스의 빠른 부팅”, 한국정보과학회 2005 가을 학술발표 문집, 2005

[3] 기용철, “임베디드 시스템을 위한 임베디드 리눅스 커널 포팅에 관한 연구”, 전남대학교대학원, 2003

[4] Tim R. Bird, "Methods to Improve Bootup Time in Linux", Linux Symposium, 2004

[5] Yanbing Li, Miodrag Potkonjak, Waye Wolf, "Real-Time Operating System for Embedded Computing", UCLA, 1997

[6] Bill Weinberg, Claes Lundholm, "Embedded Linux-Ready for Real-time", 2001

[7] David Selvakumar & Chester Reveiro, "RTLinux on Memory Constraint Systems", Real Time Linux Workshop, 2004

[8] 이형석, 정영준, “임베디드 운영체제 커널 기술 동향”, 전자통신동향분석 제21권, 2006

[9] 타카하시 히로카즈, “Linux Kernel 2.6 구조와 원리", 한빛미디어, 2007

[10] 보베이, 다니엘, “리눅스커널의 이해”, 한빛미디어, 2006

[11] 김사혁, “임베디드 리눅스 표준제정”, 정보사회연구실, 2003

[12] Erich Boleyn, "GRUB Technical Info", http://www.uruk.org/orig-grub/technical.html

[13] Gustavo Duarte Blog, "Gustavo Duarte software, Computers, and Business.", http://duartes.org/gustavo/blog/

[14] http://www.pixelbeat.org/