한국컴퓨터정보학회 동계학술대회 논문집 제31권 제1호 (2023. 1)

다중코어 운영체제를 위한 부트로더 설계 및 구현

김동휘, 박연택, 정해람, 방태훈, 주용완, 이준동
강릉원주대학교
e-mail: awesome_devnet@outlook.com, parkyt123@naver.com, jhr83199@gmail.com,
kiyawo0210@naver.com, ywju@gwnu.ac.kr, jlee@gwnu.ac.kr

Design and Implementation of Boostrap Loader on Multi Core Operating System

DongHwi Kim, YeonTaek Park, HaeRam Jung, TaeHoon Bang, YongWan Ju, JunDong Lee
GangNeungWonju National University

● 요 약 ●

운영체제(Operating System)는 사용자의 하드웨어, 시스템 자원(System Resources)을 제어하고 프로그램에 대한 일반적 서비스를 지원하는 시스템 소프트웨어(System Software)이다. 시스템 하드웨어를 관리할 뿐 아니라 응용 소프트웨어를 실행하기 위하여 하드웨어 추상화 플랫폼과 공통 시스템 서비스를 제공한다. 최근에는 가상화 기술의 발전에 힘입어 실제 하드웨어가 아닌 가상 머신(HyperVisor) 위에서 실행되기도 한다.

본 연구에서는 다중 코어 프로세서를 타겟으로 한 소규모 운영체제 개발 프로젝트의 일환으로 부트로더를 설계하고 구현하였다. 부팅은 최초 컴퓨터에 전원이 들어온 후 운영체제가 실행할 수 있는 환경을 구축하는 데 가장 중요한 역할을 하는 프로그램이며, 이를 잘 활용하면, 임베디드 시스템, IOT 등 다양한 분야에 이용할 수 있다.

키워드: 운영체제(Operating System), 부팅(Booting),
부트 로더(Boot Loader), 시스템 소프트웨어(System Software)

I. Introduction

운영체제(Operating System)는 사용자의 하드웨어, 시스템 자원(System Resources)을 제어하고 프로그램에 대한 일반적 서비스를 지원하는 시스템 소프트웨어(System Software)이다. 시스템 하드웨어를 관리할 뿐 아니라 응용 소프트웨어를 실행하기 위하여 하드웨어 추상화 플랫폼과 공통 시스템 서비스를 제공한다. 최근에는 가상화 기술의 발전에 힘입어 실제 하드웨어가 아닌 가상 머신(HyperVisor) 위에서 실행되기도 한다.

부팅(Booting)이란 전원이 들어오고, 사용자가 시작할 수 있는 상태가 될 때까지의 과정을 말하는데, 운영체제가 잘 동작하기 위해서는 필수적으로 이 과정이 잘 동작해야 한다.

본 논문에서는 다중코어 운영체제를 위한 부트로더 설계 및 구현에 관하여 설명한다.

II. Preliminaries

2.1 운영체제 개발 환경

운영체제를 개발하기 위해 사용하는 언어 및 라이브러리는 GCC(GNU C Compiler), NASM, Binutils, Make tools 등을 사용한다. 다른 C++ 등의 언어들을 활용하지 않고 C를 사용하는 이유는 우선 특별한 기반 소스코드를 수정하는 것이 아닌 기반부터 작성하는 프로젝트의 특성 상, 기본적으로 컴파일러에 제공되는 라이브러리를 사용할 수 없기 때문이다.

추가적으로 레지스터 제어 등 C언어에서 제어하기 어려운 기능의 구현은 NASM의 어셈블리어를 사용한다. 우선 NASM의 어셈블리어를 사용하는 이유는 INTEL 어셈블리어 문법의 형태가 반영되어 있기에 본 프로젝트에 적합하다.

관련된 언어 및 라이브러리를 사용하기 위해서는 Cygwin을 사용하여 해당 기능들을 설치한다. Cygwin은 Windows에서 Linux의 터미널을 사용할 수 있도록 해 주는 GNU의 프로그램이다. Cygwin은 시그너스 솔루션스가 개발한 자유 소프트웨어 모음집으로 현재는 GNU GPL로 Release되어 자유롭게 사용이 가능하다. 프로젝트에

사용할 언어 및 라이브러리는 Cygwin을 통해 설치해야 한다.

1) GCC

GNU Project의 오픈 소스 컴파일러 모음이다. UNIX/LINUX 계열 플랫폼의 표준 컴파일러이다. 초기에는 C언어만을 다루는 컴파일러였으나, 오늘날은 C++(g++), Objective-C(gobjc) 등의 다양한 언어들의 컴파일을 지원한다.

2) Make

Make는 파일을 관리하기 위해 제공되는 유틸리티의 한 종류이며, 파일 간의 종속 관계를 파악하여 makefile 문서에 작성된 표준에 따라 컴파일러에 명령하여 Shell 명령이 순차적으로 실행함으로써 여러 개의 개발 언어로 작성된 여러 개의 소스 파일을 일정 규칙에 따라 자동화된 빌드가 가능하도록 하기 위해 사용하였다. Make의 장점은 다음과 같다.

각 파일에 대한 반복적 명령의 자동화로 인한 시간 절약이 가능하다.

프로그램의 종속된 구조를 빠르게 파악할 수 있으며 관리 또한 용이하다.

3) Binary Utils(binutils)

리눅스 환경의 터미널을 활용하는 만큼 Linux 환경에서 Source를 Compile 혹은 Debug할 때 해당 작업에 관련한 ld나 as 등의 다양한 도구를 제공한다.

2.2 Boot

<그림 1>은 Boot의 과정을 나타낸 것이다.

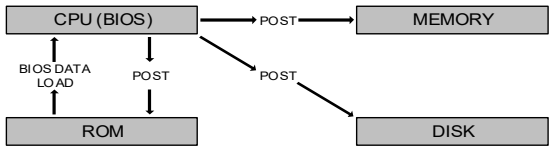

Fig. 1. 하드웨어 구조

1. BIOS(Basic Input Output System)

PC에 전원 인가 시 CPU는 ROM에 있는 BIOS의 Data를 가져온다.

2. POST(Power on Self Test)

BIOS 환경에서 주변 장치들에 대하여 하드웨어들이 정상적으로 작동하는지 검사한다.

3. POST에 이상이 없으면 BIOS는 BootStrap을 실행하여 Disk의 MBR 영역, 즉 첫 번째 Sector에 있는 부트 데이터를 담고 있는 부트로더를 메모리로 로드한다.

4. MBR에 정의된 부분에 의해 Disk 상의 운영체제 이미지를 메모리로 로드한다.

5. 로드 완료 후 해당 운영체제의 이미지를 복사한 메모리의 주소로 jmp한다.

III. The Proposed Scheme

3.1 프로젝트 설계

프로젝트 디렉토리는 <그림 2>와 같다.

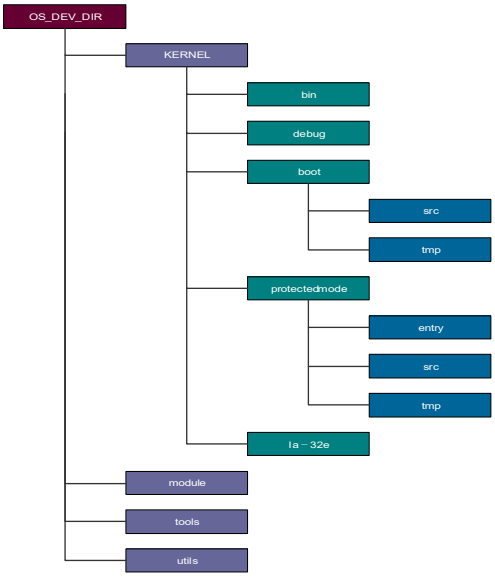

Fig. 2. 프로젝트 디렉토리

1) Kernel/Modules/tools/Utils를 최상위 폴더로 분류하여 각각의 파트에 대한 소스 및 바이너리로 구분하여 생성

2) Kernel\bin

bin 디렉토리는 컴파일된 바이너리 및 오브젝트 데이터만을 분리하여 소스 코드에 섞이지 않도록 구성

3) Kernel\Boot

16Bit Real Mode, 즉 BootStrap Loader에 대한 소스 코드가 해당 디렉토리 내부 \src에 작성된다.

4) Kernel\ProtectedMode

32Bit Protected Mode, 보호 모드 커널에 대한 소스 코드가 해당 디렉토리 내부 \src\에 작성된다.

5) Kernel\Debug

파트 당 프로토타입에 대한 임시 빌드 데이터를 구분한다.

3.2 보호모드

보호 모드는 현재 구현 중으로, 부트 로더에서 현재 구현 중인 32비트 보호 모드에 대한 이미지로 jmp하는 코드까지 구현되어 있다.

[어셈블리 코드]

```
[ORG 0]
[BITS 16]

SECTION .text
jmp 0x07C0:_SET
```

```
_SET:
    mov ax, cs
    mov ds, ax

(중략)
_TRANSPOSE_KERNEL:
    jmp 0x1000:0000
    hlt

_ISERROR:
    jmp $

times 510 - ($ - $$) db 0
db 0x55, 0xAA
```

512바이트 구성을 위하여 마지막 2줄을 통해 510바이트를 0으로 채우고 나머지 두 바이트를 각각 0x55와 0xAA를 넣어 512바이트로 만들어 주는 형태의 구성이다. 코드에서는 보호 모드에 대한 이미지를 불러올 메모리 상의 주소를 0x10000번지로 지정하였으며, 로드 완료 후 해당 번지로 jmp하여 모드를 전환한다. 실제로 윈도우 프로그램과는 달리 커널 작성의 파트이므로 육안 상 무언가 표시하기는 다소 어렵다. 하여 정상 작동의 시연을 위해 다음과 같이 비디오 메모리(0xB8000)를 활용하여 문자를 출력한다.

[어셈블리 코드]

```
_SET:
    mov ax, cs
    mov ds, ax

    mov ax, 0xB800
    mov es, ax

(중략)
mov si, 0
_INIT:
    mov byte[es:si], 0
    mov byte[es:si+1], 0x0A
    add si, 2
    cmp si, 80 * 25 * 2
    jl _INIT
mov si, 0
mov di, 0
PRINTTEXT:
    mov cl, byte[si+SZTEXT]
    cmp cl, 0
    je _TRANSPOST_KERNEL
    mov byte[es:di], cl
    inc si
    add di, 2
    jmp PRINTTEXT

_TRANSPOSE_KERNEL:
    jmp 0x1000:0000
    hlt

_ISERROR:
    jmp $

times 510 - ($ - $$) db 0
db 0x55, 0xAA

SZTEXT: db "Load Complete..", 0
```

비트 보호 모드에 대한 이미지로 jmp하는 코드까지 구현되어 있다. 해당 코드에 의해 정상 작동을 확인하는 시연은 <그림 3>과 같다.

Fig. 3. 코드의 시연

작성된 부트 로더의 이미지를 가상 환경인 QEMU를 통해 마운트한 사진이다.

IV. Conclusions

운영체제(Operating System)는 사용자의 하드웨어, 시스템 자원(System Resources)을 제어하고 프로그램에 대한 일반적 서비스를 지원하는 시스템 소프트웨어(System Software)이다.

부팅(Booting)이란 전원이 들어오고, 사용자가 시작할 수 있는 상태가 될 때까지의 과정을 말하는데, 운영체제가 잘 동작하기 위해서는 필수적으로 이 과정이 잘 동작해야 한다.

본 연구에서는 다중 코어 프로세서를 타겟으로 한 소규모 운영체제 개발 프로젝트의 일환으로 부트로더를 설계하고 구현하였다. 부팅은 최초 컴퓨터에 전원이 들어온 후 운영체제가 실행할 수 있는 환경을 구축하는데 가장 중요한 역할을 하는 프로그램이며, 이를 잘 활용하면, 임베디드 시스템, IOT 등 다양한 분야에 이용할 수 있다.

ACKNOWLEDGEMENT

이 논문은 2022년도 정부(산업통상자원부)의 재원으로 한국 산업기술진흥원의 지원을 받아 수행된 연구임(과제번호: P0011930, 2022년 산학융합지구조성사업).

REFERENCES

[1] SeungHoon Han, “IT EXPERT, 64Bit MultiCore OS Principal and Structure”, HanbitMedia, 2014.

[2] Kawai Hidemi, “OS structure and principle”, HanbitMedia, 2007
[3] Harvey M.Deitel, Paul J. Deitel, David R.Choffnes, “Operating Systems“, HanbitMedia, 2009
[4] Hyun-Hwoi Ku, “Operating System”, HanbitMedia, 2016