Project #1: 부트로더 개발

신정은, 조재호, 조한주
Soongsil University

1. 구현 목표

이번 과제에서 구현해야 할 구현 목표는 다음과 같다.

1) 부트 로더 작성
2) 가상 OS 이미지 작성
3) 부트 로더 및 OS 이미지를 빌드하는 Makefile 작성
4) 부트 로더에 부팅 시에 현재 시간을 출력하는 코드 추가

다양한 기능을 담고 있는 OS라 할지라도 처음 시작은 고작 512바이트의 부트로더부터 시작한다. 부트로더는 OS의 나머지 코드를 메모리에 복사해 실행시키는 역할을 한다. PC가 켜지면 BIOS라는 메인보드에 포함된 펌웨어가 하드웨어를 초기화하고 부트로더 이미지를 메모리로 복사한다. 부트로더는 OS 를 담고 있는 저장 매체의 가장 앞 부분에 위치한다. 부트로더가 존재한다면 OS 이미지 코드를 0x7C00 주소로 복사하고 프로세서가 0x7C00 주소부터 코드를 수행해서 OS가 작동한다. 만약 부트로더가 없다면 BIOS가 'Operating System Not Found'와 같은 오류 메시지를 출력하고 작업을 중단한다. 부트로더가 디스크에서 메모리로 복사되어 실행되었다는 것은 BIOS에 의해 PC가 정상적으로 구동되었다는 것을 의미하므로 부트로더를 만드는 것은 OS를 만드는 데에 있어서 필수적이다. 따라서 부트로더를 만드는 것이 처음으로 이루어진다.

부트로더를 구현하면 OS 이미지를 로딩할 수 있다. 하지만 아직 로딩 될 실제 OS 코드를 작성하지 않았다. 따라서 부트로더가 정상적으로 작동하는지 확인하기 위해 가상 OS 이미지를 작성한다.

make는 자동으로 소스 코드를 이용하여 실행파일과 라이브러리를 만들어주는 프로그램이다. 또한 소스 코드와 목적 파일을 비교하여 마지막으로 수정된 파일만 선택하여 빌드하여 빌드 시간을 크게 줄여준다. 현재는 부트로더와 가상 OS 이미지만 작성되어서 빌드하는게 크게 복잡하지 않다. 하지만 앞으로 OS가 완성되어감에 따라 소스 코드도 많아지고 빌드 방법도 복잡해진다. 이 때 make 프로그램을 이용하지 않고 손수 빌드를 하는 것은 매우 비효율적이며 실수를 유발하게 된다. 그리고 개인적으로 개발자는 DRW(Don't Reinvet the Wheel) 원칙을 항상 지키도록 노력해야 한다고 생각한다. 자주 사용될 것으로 예상되는 작업은 항상 자동화하도록 노력하고 점진적으로 모든 기능을 모듈화해야 한다. 이러한 원칙하에 소스 코드 빌드를 위해 Makefile을 작성하는 것은 매우 바람직하다.

부팅 시에 현재 시간을 출력하도록 부트로더에 코드를 추가한다. 출력화면에 나오는 시간은 host linux의 시간과 같다. 원래는 컴퓨터 메인보드 안에 수은 전지와 시계가 내장되어 있어서 BIOS로부터 시간을 받아온다. 하지만 Virtual Machine의 linux 상에서 QEMU 시뮬레이터로 OS를 구현하므로 시뮬레이터의 옵션을 이용하여 linux의 local time을 가져와 구현한다.

2. 구현 내용

2.1 부트 로더 작성

가장 먼저 실행되는 프로그램인 부트 로더를 작성한다. 부트 로더를 512바이트로 설정하고 마지막 2바이트를 0x55, 0xAA로 저장한다. 이는 이렇게 부트로더가 작성되기로 정해져있기 때문이다.

```
291 times 510 - ($ - $$)    db    0x00
292
293 db 0x55
294 db 0xAA
```

<그림1> 부트로더의 빈부분을 0으로 채우고 마지막 2바이트를 0x55, 0xAA로 저장

보통의 OS는 부팅 시에 환영 메시지와 부팅 과정을 출력하는데 우리도 이를 구현한다. 비디오 메모리의 어드레스는 0xB800에서 시작한다. 텍스트 모드에서의 화면의 크기는 가로 80, 세로 25문자이고 한 글자는 2바이트(문자와 속성 1바이트씩)를 차지한다. 총 메모리 크기는 4000바이트이다. 0xB800번지의 주소부터 1바이트(들어갈 char) + 1바이트(문자의 속성)을 넣어주면 문자열이 화면에 출력된다. 이 때 세그먼트 레지스터에 BIOS에서 사용하던 값들이 들어있어 엉뚱한 주소에 접근할 수 있으므로 세그먼트 레지스터를 초기화해야한다. CS와 DS레지스터는 BIOS

가 부트 로더를 디스크에서 읽어 복사하는 위치인 0x7C00으로 설정하고 ES 세그먼트 레지스터는 비디오 출력에 사용하기 위해 0xB800으로 설정한다. 화면을 지우고 출력하는 소스는 반복되므로 함수로 작성하고 스택 자료 구조를 사용해서 호출 가능한 구조로 만든다. 이제 OS 이미지 로딩을 구현한다. OS 이미지는 플로피 디스크에 저장되며 섹터x헤드x트랙 단위로 저장된다. 이를 읽어서 메모리로 올리는 코드를 작성한다.

2.2 가상 OS 이미지 작성

부트 로더가 정상 작동하는지 확인하기 위해 가상 OS 이미지를 만든다. 가상 OS는 특별한 일을 하지는 않지만 실제 OS 이미지가 최소 수백KB이기 때문에 512KB(1024섹터)로 설정하여 가상 OS 이미지를 만든다. 우리가 사용하는 NASM 어셈블러가 전처리기를 제공하므로 전처리기의 매크로, 조건, 반복을 사용하여 쉽게 512KB를 채우는 OS 이미지를 만들 수 있다. 가상 OS 이미지는 1024섹터 중 1023섹터의 코드는 화면에 자신을 출력하고 섹터의 빈 공간은 0으로 채우게 해서 정상적으로 로딩되는지 확인한다. 마지막 섹터는 같은 과정을 하고 무한 루프에 빠지도록 한다. 이 과정은 NASM 어셈블러의 %rep와 %if %else를 이용해 쉽게 구현 가능하다.

2.3 부트 로더 및 OS 이미지를 빌드하는 Makefile 작성

부트 로더 및 OS 이미지가 준비되었으므로 이 파일들을 하나로 합쳐 부팅 이미지로 만든다. 이 때 make 명령을 사용하기 때문에 Makefile을 작성한다. Makefile은 all 레이블에 BootLoader Kernel32 Disk.img를 입력해서 부트 로더 -> 가상 OS 이미지 -> 부팅 이미지 순으로 빌드한다. 부팅 이미지는 각각 어셈블러가 번역한 부트 로더 목적 파일과 가상 OS 이미지 목적 파일은 cat 명령을 이용하여 출력한다.

2.4 부트 로더에 부팅 시에 현재 시간을 출력하는 코드 추가

데이터 영역에 TIMEBUF를 잡아놓는다. 그리고 그 위치를 bx레지스터에 넣어놓는다. 그러면 bx레지스터로 TIMEBUF의 시간 값을 쓸 위치에 접근할 수 있다.

```
TIMEBUF: db '00:00:00', 0

mov bx, TIMEBUF
```

BIOS는 특별한 기능을 외부로 제공한다. 이는 특수기능을 가진 함수의 주소를 Interrupt Vector Table에 넣어 두고 Software Interrupt를 호출하는 방법으로 구현한다. 이때 작업에 파라미터가 필요한데 이는 레지스터를 이용한다. 레지스터에 파라미터를 저장하고, 또 결과로 레지스터로 받아온다. 우리가 필요한 기능은 인터럽트 벡터 테이블의 1ah 인덱스에 있다. 이는 실시간 시계 서비스를 제공하는 SWI다. 그리고 ah 레지스터에 0x02로 설정하면 RTC 시간을 읽어온다. 그리고 결과값은 ch, cl, dh 레지스터에 각각 시간, 분, 초가 1바이트씩 저장된다. 이 코드는 다음과 같다.

```
mov ah, 0x02
int 0x1a
```

1a 인터럽트 호출 시, 에러가 발생할 경우를 대비하여 함수영역에 GETTIMEERROR 함수를 정의하고, 데이터 영역에 있는 TIMEERRORMESSAGE를 출력하게 하였다.

```
jc GETTIMEERROR

GETTIMEERROR:
        push TIMEERRORMESSAGE
        push 1
        push 20
        call PRINTMESSAGE

        jmp $

TIMEERRORMESSAGE: db 'TIME ERROR~', 0
```

인터럽트를 호출하였으므로 ch, cl, dh 레지스터에 각각 시, 분, 초가 들어가 있다. 우선 시(hour)를 예를 들어 생각해 본다. ch레지스터에 있는 값을 우선 al레지스터에 저장한다. 그런데 이때 BCD형태로 되어 있으므로 우선 ah레지스터에도 al과 같은 값을 넣어준다. 24시라면 0x2424로 저장되어 있다. ah레지스터에는 십의자리가 있어야하고 al레지스터에는 일의자리가 있어야한다. 필요한 부분만 남기기 위해 0xf00f와 ax를 and연산하면 ah레지스터엔 십의자리가, al레지스터엔 일의자리가 남아있다. 수행하고 나면 0x2004가 된다. 0x0204형태가 되어야 하므로 ah레지스터를 오른쪽으로 4비트를 shift한다. 그런데 이것을 그대로 출력하면 이상한 문자가 나온다. 아스키문자로 변환하기 위해 ax레지스터에 0x3030을 더한다. ah, al레지스터 각각에 더해야 하기 때문이다.

그리고 처리한 것을 TIMEBUF에 넣어줄 차례이다. 처음에 bx레지스터에 TIMEBUF주소를 넣어줬으므로 bx레지스터가 갖고있는 값의 바이트를 ah레지스터의 값으로 덮어쓴다. 다음으로 일의자리를 넣기 위해 bx값을 1을 증가시켜서 al레지스터의 값으로 덮어쓴다. 그 다음엔 ':' 문자가 있으므로 bx레지스터 값을 2증가시켜준다.

위의 과정은 분, 초에서도 동일하다. 따라서 맨 처음에 al레지스터에 값을 넣는 부분 이후의 과정을 GETTIME함수에 넣고 코드 영역에서 GETTIME함수를 call하는 것으로 바꿔준다.

```
;hour
mov al, ch
call GETTIME

;minute
mov al, cl
call GETTIME

;second
mov al, dh
call GETTIME
```

```
GETTIME:
        mov ah, al
        and ax, 0xf00f
        shr ah, 4
        add ax, 0x3030

        mov byte[ bx ], ah
        add bx, 1
        mov byte[ bx ], al
        add bx, 2
        ret
```

그리고 TIMEBUF 를 인자로하여 PRINTMESSAGE 함수를 호출한다.

```
push TIMEBUF
push 1
push 15
call PRINTMESSAGE
add sp, 6
```

2.5 개발 시 경험한 문제점 및 해결점

- 비디오 출력 과정을 정확히 이해하지 못한 문제
처음에는 단순하게 기존의 PRINTMESSAGE 함수를 사용하면 되지 않을까 생각했다. 그래서 'current time: ' 까지는 출력을 쉽게 했지만 'OS Image Loading…' 의 출력이 사라졌다. 다시 소스 코드를 살펴보니 0xB800 에서 시작하는 메모리 시작 주소부터 시작해서 한 줄에 160 바이트씩 출력된다는 것을 확인했다. 따라서 IMAGELOADINGMESAGE 및 LOADINGCOMPLETEMESSAGE 와 가상 OS 이미지 작동 문자열을 160 바이트씩 뒤에 출력되게 수정하니 정상으로 출력되었다.

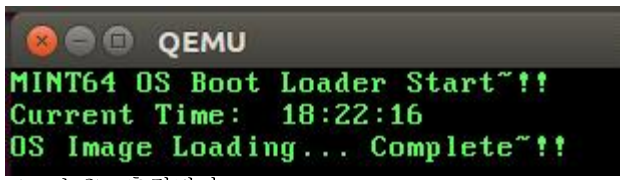
- 필요한 레지스터를 제대로 알지 못한 문제
시간을 알기 위해서는 인터럽트 벡터 테이블의 1a 인덱스에 있는 SWI 호출하여 정보를 ch, cl, dh 레지스터에 받아와야 한다. 그런데 자꾸 'invalid combination of opcode and operands' 오류가 발생했다. 알고보니 이는 ch, cl, dh 레지스터는 한 바이트 씩이고 산술 연산에 쓰이는 ax 레지스터는 16 비트이기 때문이었다. 이를 알고 al 을 사용하니 정상적으로 연산이 되었다. 한 바이트를 2 부분으로 나누어 문자로 만드는 부분은 저번 학기의 시스템 프로그래밍에서 다루었던 비트 연산을 알고 있어 쉽게 해결하였다.

- 스택의 작동 원리를 제대로 이해 못한 문제
숫자의 출력까지 다 되었는데 ':' 문자를 출력하려고하면 time 이 음수가 되었다는 오류가 생겼다. 처음에 PRINTCURRENTMESSAGE 함수를 구현할때 기존의 PRINTMESSAGE 함수를 바탕으로 구현 했는데 이때 push 한 레지스터와 pop 하는 레지스터를 제대로 맞춰주지 않아서 오류가 발생했다. 이를 알고 제대로 사용하는 레지스터들을 적어주니 해결되었다.

- 이상한 문자가 나온 문제
값을 제대로 처리해서 출력을 했는데 자꾸 이상한 문자가 나왔다. 그냥 16 진수 숫자로 출력했기 때문이다. 그래서 아스키문자로 변환하기 위해 ah, al 레지스터 각각에 0x30 씩 더해줘서 문자로 변환해줬다.

- 데이터 영역을 활용하지 못한 문제
시간을 출력할 때 처음에는 TIMEBUF 를 데이터영역에 잡지 않고 레지스터 값을 시, 분, 초 나온 대로 출력을 했지만 이렇게 하면 똑같은 코드를 계속 반복하게 된다. 그래서 c 언어처럼 데이터영역에 공간을 잡아놓고 거기에 덮어쓰면 되지 않을까 생각하게 되어서 TIMEBUF 를 '00:00:00'으로 데이터 영역에 선언해 놓고 bx 레지스터에 TIMEBUF 의 주소를 넣고 bx 레지스터의 값을 증가시키며 해당 주소에 값을 넣어주었다. 그리고 TIMEBUF 를 PRINTMESSAGE 함수의 인자로 넘겨주어 출력하기만 하였다.

- 스택을 완전히 활용하지 못한 문제
위에서 데이터 영역을 사용하여 GETTIME 함수를 통해 코드의 반복을 줄였지만, 여전히 al 레지스터에 값을 넣고 GETTIME 을 호출하는 부분은 반복된다. 그래서 스택에 ch, cl, dh 레지스터 값을 넣어놓고 차례로 출력하며 al 레지스터에 넣고 GETTIME 함수를 호출하는 함수를 만들려고 하였으나 스택이 2 바이트라서 al 레지스터의 값을 넣으면 크기가 같지 않다고 에러가 떴다. 그래서 ax 레지스터를 0 으로 초기화 한 후, ax 레지스터에 넣으려고 했지만 계속 invalid 에러가 났다. 계속 코딩을 하다보니 ax 를 0 으로 계속 초기화하는 것과 bp 값을 바꾸면서 sp 와 비교하는 것이 더 비효율적인 것 같아서 이 부분은 그냥 남겨두게 되었다. 하지만 해결하려는 과정에서 넘겼던 부분(스택의 한칸이 2 바이트로 되어있고 이 OS 에서는 1word 가 2byte 라는 것)을 보게 되었다.



<그림 2> 출력결과

3. 조원별 기여도

신정은 33%, 조재호 34%, 조한주 33%
팀원간의 협업을 위해 코드 리뷰 및 관리 용도로 깃허브를 이용한다.
팀 프로젝트 깃허브 주소 :
https://github.com/birdbirdgod/2018_OS