임베디드시스템S/W설계

[Assignment 1 & Assignment 2]

담당교수 : 김태석 교수님

학 번 : 2021202058

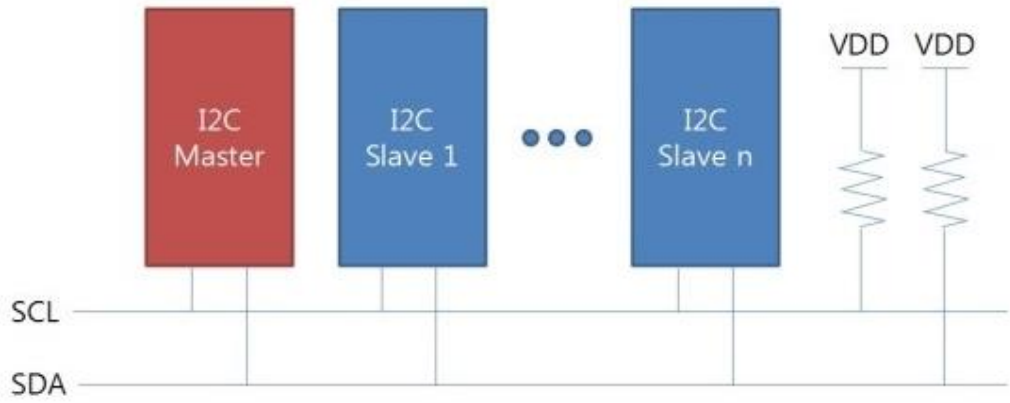
성 명 : 송채영

[Assignment 1]

- **I2C**

  I2C의 정식 명칭은 Inter-Integrated Circuit이고 마이크로프로세서와 저속 주변장치 사이의 통신을 용도로 Philips에서 개발한 프로토콜이다. 두 라인을 이용해 데이터를 교환해 Two-Wired Interface(TWI)라고도 불린다. I2C는 두 개의 신호 선으로 구성되며, 하나는 데이터 라인인 SDA(Serial Data)이고 다른 하나는 클럭 라인인 SCL(Serial Clock)이다. 또한 I2C는 Master-Slave 형태로 동작하고 속도면에서는 다른 방식들에 비해 느리지만, 하드웨어적으로 간단히 구성할 수 있고 대화형 protocol을 만들 수 있으며 하나의 버스에 많은 노드를 연결할 수 있다는 장점이 존재한다.

  SCL은 통신의 동기를 위한 클럭용 선이고 SDA는 데이터용 선이다. Master는 SCL로 동기를 위한 클럭을 출력하며 Slave는 SCL로 출력 되는 클럭에 맞추어 SDA를 통해 데이터를 출력하거나 입력 받는다. SDA 한 선으로 데이터를 주고받기 때문에 I2C는 half duplex 통신만 가능하다. 또한 각 장치는 고유한 7 비트 주소를 가지며 Master는 이를 통해 특정 Slave를 식별해 통신한다. 이때 주소의 길이가 7 비트이므로 최대 128개의 Slave를 지원할 수 있다.



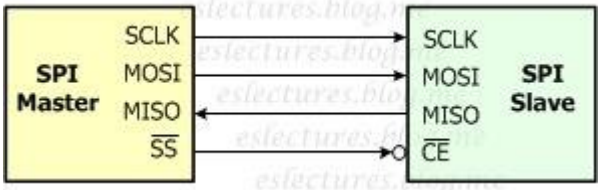
3) I2C 통신방식

I2C 프로토콜에는 시작 조건, 정지 조건, 데이터 안정 구간이 있다. 우선 시작 조건은 SCL이 1인 상태에서 SDA가 1에서 0으로 전환될 때 발생하며, 정지 조건은 SCL이 1인 상태에서 SDA가 0에서 1로 전환될 때 발생한다. 데이터 안정 구간에서는 SCL이 1인 상태에서 SDA가 그 상태를 유지해야 한다. 이를 통해 Master는 통신의 시작과 끝 그리고 소유권을 다른 장치들에게 알릴 수 있다.

- **SPI**

  SPI의 정식 명칭은 Serial Peripheral Interconnect이고 마이크로 컨트롤러 또는 주변 장치

와 통신하기 위한 하드웨어 통신 프로토콜 중 하나이다. Motorola에서 개발했으며 full duplex 통신이 가능하다. SPI는 Master-Slave 방식으로 동작하며, Master는 동기를 위한 클럭을 출력하고 각 Slave 장치는 chip enable (/CE) 입력을 가지고 있으며 이 입력이 활성화되었을 때만 동작한다. 이에 따라 Master는 여러 개의 slave select(ss) 선을 Slave들의 /CE에 연결하고 한 순간의 하나의 slave만 선택하는 방법을 사용해 두 개 이상의 slave 장치를 구동할 수 있다. SPI는 빠른 통신 속도와 다양한 데이터 전송 형식을 지원한다는 점이 장점이다.



SPI는 네 개의 주요 신호선을 사용한다. 먼저 SCLK(Serial Clock)은 master가 클럭 신호를 생성하여 통신 속도를 동기화하는데 사용된다. 다음으로 MOSI(Master Out Slave In)은 master에서 slave로 데이터를 전송하는데 사용된다. MISO(Master In Slave Out)은 slave에서 master로 데이터를 전송하는데 사용된다. 마지막으로 SS(Slave Select)는 master가 특정 slave를 선택해 통신할 때 사용된다. 이 신호는 다중 slave 구성에서 각 slave에 연결된 개별 ss 신호선이 있을 수 있다. 위에서 언급한 chip enable (/CE)는 slave 장치가 master로부터 데이터를 수신하거나 전송하기 위해 활성화되는데 사용된다. SPI의 통신 과정은 다음과 같다. Master는 선택된 slave의 /CE 신호를 활성화하여 해당 slave와의 통신을 준비한다. Master는 클럭 신호를 생성하여 SCLK 신호 선을 통해 클럭을 slave에 동기화 한다. 이후 master는 MOSI 신호를 통해 데이터를 slave에 전송한 동시에 MISO를 통해 데이터를 master에게 전송한다. Master와 slave는 full-duplex 통신을 통해 데이터가 주고받아지며, 클럭에 의해 동기화된다. 통신이 완료되면 master는 /CE 신호를 비활성화하여 해당 slave와의 통신을 마무리하고 다른 slave와의 통신을 준비한다.

- **SDIO**

SDIO의 정식 명칭은 Secure Digital Input/Output이고 주로 휴대폰, 디지털 카메라, MP3 플레이어 같은 소형 디지털 장치에서 사용되는 인터페이스이다. SDA에서 개발했으며 주로 외부 장치와의 통신을 위한 기능을 지원하고 이를 통해 디지털 장치의 확장성과 기능성을 높인다.

SDIO의 동작 방식은 인터페이스 설정, 명령 전송, 데이터 전송, 응답처리, 인터페이스 해제, 크게 5단계로 나눌 수 있다. 먼저 인터페이스 설정 부분은 호스트 장치는 SDIO 장치와의 통신을 시작하기 위해 인터페이스를 설정한다. 이 단계에서는 SDIO 장치의 전원을 활성화하고 클럭 신호를 설정하는 것이 포함된다. 이어서 명령 전송 부분이다. 호스트

장치는 SDIO 장치에 명령을 전송한다. SDIO 장치의 기능을 제어하거나 데이터 전송을 시작하는데 사용되며 CMD 신호를 통해 전송되고 SDIO 장치는 해당 명령에 대한 응답을 반환한다. 데이터 전송 부분은 명령이 성공적으로 전송되면 이루어진다. 데이터 블록이 SDIO 장치와 호스트 장치 간에 전송되며 데이터는 데이터 신호를 통해 전송되고 클럭 신호를 통해 동기화된다. 데이터 처리가 완료되면 SDIO 장치는 호스트 장치에 대한 응답을 반환하는 응답 처리 부분이다. 응답은 데이터 전송의 성공 또는 실패 여부를 나타내고 호스트 장치는 이에 따라 적절히 처리한다. 마지막으로 통신이 완료되면 호스트 장치는 SDIO 장치와의 인터페이스를 해제하고 SDIO 장치의 전원을 비활성화한다.

- **PCI**

  PCI의 정식 명칭은 Peripheral Component Interconnect로 컴퓨터의 주요 부품인 CPU, 메모리, 그래픽 카드, 네트워크 카드 등의 다양한 주변 장치들을 연결하기 위한 표준 인터페이스이다. 이를 통해 다양한 하드웨어를 컴퓨터 시스템에 쉽게 추가하고 확장할 수 있다. 높은 대역폭을 가지고 있어 데이터 전송 속도가 빠르며 대부분의 주변 장치들이 PCI 슬롯에 호환된다는 장점이 있다.

  PCI의 동작 방식은 다음과 같다. 시스템이 시작될 때 PCI 버스 컨트롤러는 시스템에 연결된 모든 PCI 장치를 검색하고 각 장치에 고유한 주소를 할당한다. PCI 장치는 시스템에 식별될 수 있도록 PIC Configuration Space에 자신의 정보를 기록한다. 이어서 주소 매핑 부분이다. PCI는 버스-마스터링 구조를 사용해 다중 장치 간 통신을 가능하게 한다. 각 PCI 장치는 메모리 공간 또는 입출력(I/O) 포트 주소를 할당 받아 CPU가 해당 장치와 통신할 수 있도록 한다. 데이터 전송 부분이다. CPU가 PCI 장치와 통신하려면 해당 장치의 주소를 목적지로 하는 명령 및 데이터를 버스에 전송해야 한다. PCI는 데이터 전송에 DMA(Direct Memory Access)를 사용하여 CPU의 개입 없이 데이터를 메모리에 직접 전송할 수 있고 주변 장치는 CPU로부터 받은 명령을 처리하고 결과를 반환한다. 다음으로 Interrupt 처리 부분이다. PCI장치가 CPU에게 어떤 이벤트를 알리고자 할 때 인터럽트를 발생시키며 CPU는 해당 인터럽트를 받아들여 처리할 수 있고 인터럽트 서비스 루틴을 실행해 해당 이벤트를 처리한다. 마지막으로 PCI는 Plug and Play 기능을 지원해 새로운 장치를 시스템에 추가할 때 자동으로 장치를 인식하고 구성한다. 새로운 장치가 연결되면 PCI버스 컨트롤러가 장치를 탐지하고 해당 장치의 리소스를 할당한다. 위의 동작 방식을 통해 PCI는 컴퓨터 시스템에서 다양한 주변 장치들과 효율적으로 통신할 수 있다고 한다.

- **USB**

  USB의 정식 명칭은 Universal Serial Bus이며 컴퓨터와 외부 장치 간에 데이터 전송, 전원공급 등을 위한 표준 인터페이스로 키보드, 마우스, 프린트, 외장 저장 장치, 카메라 등 다양한 장치를 컴퓨터에 연결하여 사용할 수 있다.

USB는 기존의 PC와 주변 기기간의 인터페이스를 통합해 나가는 것을 목적으로 Compaq, Intel, Microsoft, NEC의 엔지니어들이 모여 차세대의 주변 인터페이스에 대한 공동연구를 시작하며 생겼다. 1996년 USB 1.0 버전이 공개되어 컨트롤러 LSI 제품화로 PC에 USB 장착이 가능 해졌고, 12Mbps의 전송 속도로 다양한 기기를 연결할 수 있었으나 화상 데이터의 실시간 전송에 한계가 있었다. 이에 USB 2.0이 1999년에 등장하면서 480Mbps의 초고속 전송이 가능해져 화상기기와 대용량 스토리지 등의 멀티미디어 기기에 대한 문제가 해결되었다.

USB는 다양한 통신 방식과 속도 선택을 통해 디바이스의 특성에 맞춰 유연한 사용이 가능하며 plug and play 및 hot plug in 기능을 지원하여 편리하게 디바이스를 연결 및 해제할 수 있다. 또한 통일된 규격의 커넥터로 인해 소형 커넥터를 사용하고 제작할 수 있어 디자인적 자유도도 높다. USB 프로토콜은 데이터의 안전성을 보장하며, 통신 버스에서 전원을 제공하여 디바이스에 별도의 전원 장치를 필요로 하지 않는다.

다만 전용 드라이버가 필요하며 통신 길이가 보통 5m로 제한되어 있어 장거리 통신이 어렵다는 점과 한 개의 호스트와 다수의 디바이스가 보통 1:1 연결을 폴링 방식으로 돌아가며 하기 때문에 동시에 다중 디바이스를 제어하는 것이 불가능하다는 점이 한계로 적용한다.

- **Bluetooth**

Bluetooth는 전자기기들 간에 데이터를 무선으로 주고받기 위한 표준 무선 통신 기술을 말한다. 주로 휴대전화, 헤드셋, 스마트 워치, 노트북, 스피커, 자동차 시스템 등 다양한 전자기기 간에 데이터를 전송하기 위한 무선 기술로 사용된다. 케이블 연결이 필요 없으며 일정 거리 내에서 편리하게 데이터를 전송할 수 있다.

Bluetooth는 크게 4가지의 특징이 있다. 먼저 저전력 무선 통신 기술로 전력 소비가 적어 배터리 소모가 적다. 두 번째로 Bluetooth 칩의 생산 비용이 저렴하고 많은 전자기기에 내장되어 있다는 점이다. 세 번째로 보통의 Bluetooth 기기들은 10m 이내에서 통신이 가능하고, 최신 버전의 경우 더 긴 거리까지 지원이 가능하다는 점이다. 마지막으로 페어링 과정을 거친 후에는 Bluetooth 장치들끼리 쉽게 연결이 가능하다.

다음으로는 Bluetooth의 동작 방식과 관련된 내용이다. Bluetooth는 주파수 분할 다중 접속, FHSS(Frequency Hopping Spread Spectrum) 방식을 사용한다. 데이터를 전송하는 동안 주파수를 계속 변경하여 무선 간섭을 최소화하며 Bluetooth 장치들은 서로 통신을 시작하기 전에 페어링 과정을 거친다. 페어링은 장치 간의 신뢰성 있는 연결을 설정하는 과정으로 보안을 강화하기 위해 PIN 번호 또는 암호를 설정하여 사용이 가능하다.

위에서도 언급했듯이, 무선통신이 가능하다는 점과 저전력 그리고 다양한 전자기기에서 사용기 가능하며 많은 기업들이 Bluetooth를 지원하고 있지만, 짧은 거리에서만 통신이 가능하다는 점과 전송 속도가 Wi-Fi나 유선 연결보다 느릴 수 있고 페어링 또는 데이

터를 전송할 때 보안 문제가 발생할 수 있다는 점이 한계로 적용한다.

- **CAN**

  CAN의 정식명칭은 Controller Area Network, 즉 제어 영역 네트워크로 주로 실시간 제어 및 통신 시스템에서 사용되며, 자동차, 산업 자동화, 로봇 공학 등 다양한 분야에서 사용되고 있다. 또한 이 네트워크는 통신을 위한 데이터를 교환하고 버스 형태의 구조를 가지고 있다.

  CAN 인터페이스의 특징에 대해 살펴보면 다음과 같다. 멀티 master 및 멀티 slave로 여러 장치가 동시에 통신할 수 있다. 또한 오류 검출 및 복구 메커니즘이 내장되어 있어 신뢰성이 높다. 실시간 응용 프로그램에 적합하며 빠른 데이터 전송이 가능하다는 점에서 고성능이다. 마지막으로 전기적인 잡음에 대한 강한 내성을 보여 안정적인 통신이 가능하다.

  다음으로 동작방식과 관련된 부분이다. CAN 네트워크는 메시 구조 또는 버스 구조를 가지고 있어 여러 장치가 하나의 통신 매체인 버스에 연결되어 있다. 모든 장치는 이 버스를 공유한다. 또한 각 장치는 식별 가능한 고유한 주소를 가지고 있다. 이 주소를 사용해 데이터를 송수신하고 특정한 장치에게 메시지를 보낼 수 있다. 또한 CAN은 프레임 구조를 가지고 있다. 주로 두 가지 유형의 프레임이 있는데 첫 번째는 데이터 프레임이고 두 번째는 원격 프레임이다. 데이터 프레임은 실제 데이터를 전송하는 데 사용되고, 원격 프레임은 특정한 데이터를 요청하는데 사용된다. 또한 비트 스트리밍 방식으로 이루어져 있는데, 데이터가 비트 단위로 연속적으로 전송되는 것을 의미한다. 각 메시지는 시작 비트, 식별자, 데이터, CRC 및 종료 비트로 구성된다. 또한 충돌을 방지하기 위한 메커니즘을 내장하고 있는데 각 장치는 특정 메시지를 전송하기 전 버스를 감시하여 충돌이 발생하지 않는지 확인한다. 마지막으로 오류를 검출하고 복구할 수 있는 기능을 내장하고 있어 CRC를 사용해 데이터의 무결성을 검사하고, 오류가 발생한 경우 메시지를 재전송하여 데이터 손실을 방지한다고 한다.

  위에서 언급했듯이, CAN 인터페이스는 오류 감지 및 복구 기능이 있어 안정적인 통신을 제공하며 실시간 응용 프로그램에 적합해 빠른 데이터 전송이 가능하지만, 장비 및 소프트웨어의 초기 투자 비용이 높을 수 있다는 점과 네트워크 구성 및 관리가 복잡하고 다른 네트워크 프로토콜에 비해 전송 속도가 상대적 느릴 수 있다는 한계가 존재한다.

- **UART**

  UART의 정식 명칭은 Universal Asynchronous Receiver & Transmitter로 비동기 통신을 위한 전용 하드웨어를 의미한다. 데이터를 송신하려면 프로세서가 타이머를 사용해 미리 정한 직렬 통신 속도에 맞추어 병렬 데이터를 한 비트 씩 출력 핀으로 내보내야 한다. 수신의 경우 수신 핀의 상황을 계속 검사하거나 인터럽트를 사용하여 시작 비트가 수신이 되는지에 여부를 판단해야 한다. 시작 비트의 수신이 확신 되면 정해진 통신 속도에

맞추어 한 비트 씩 값을 읽어 저장해야 한다.

다음으로 UART는 비동기 통신에 필수적인 하드웨어 장치로, 직렬-병렬 데이터 변환 작업을 자동으로 처리한다. 이러한 기능으로 인해 비동기 통신이 필요한 경우에는 UART가 반드시 필요하다. 대부분의 PC는 16550와 같은 독립적인 UART IC를 사용하며 대다수의 마이크로컨트롤러는 내부에 UART를 포함하고 있다. 그러나 필요한 만큼의 UART가 내장되어 있지 않은 경우, 외부에 UARTIC을 추가로 연결해 사용할 수 있고 이를 통해 통신 채널을 확장할 수 있다. 이와 같은 역할로 UART는 비동기 통신에서 핵심적인 역할을 수행하며 효율적인 데이터 송수신을 가능하게 한다.

UART의 레지스터는 UART가 데이터를 송수신하고 제어하기 위해 사용되는 레지스터들로 구성된다. 먼저 송신 데이터 레지스터인 TxD 레지스터는 UART로 송신할 데이터를 저장하는 레지스터로 CPU가 이 레지스터에 값을 기록하면 데이터가 직렬로 출력된다. 다음으로 수신 데이터 레지스터인 RxD 레지스터는 UART로 수신된 데이터를 저장하는 레지스터로 CPU는 이 레지스터의 값을 읽어와 수신된 데이터를 처리한다. 제어 레지스터는 UART의 동작에 필요한 여러 설정을 조절하는 레지스터이고 상태 레지스터는 UART의 현재 상태를 나타내는 레지스터이다. 마지막으로 Baud rate 설정 레지스터는 직렬 통신의 속도를 설정하는 레지스터이다.

- **GPIO**

GPIO의 정식 명칭은 General Purpose Input/Output으로 일반적인 목적의 입출력 기능을 담당하는 인터페이스이다. 대개 CPU나 다른 칩에 연결되어 다른 칩으로부터 데이터를 읽어오거나 제어하는데 사용된다. 혼자 동작할 수도 있고 그룹으로 묶여 함께 동작하기도 한다. 주로 디지털 신호를 처리하며 주변장치의 상태를 감지하거나 디지털 신호를 생성해 외부 장치와 상호작용할 수도 있다. 주로 마이크로컨트롤러나 마이크로프로세서에게 제공된다.

GPIO의 특징은 다음과 같다. 다양한 주변 장치와 상호작용할 수 있는 다목적 인터페이스이며, 위에서도 언급했듯이 디지털 신호를 처리하기 위해 사용되고 디지털 핀을 통해 주로 제공된다. 소프트웨어를 통해 GPIO 핀을 제어할 수 있어 다양한 응용 프로그램에 유연하게 적용할 수 있다.

GPIO는 주로 두 가지 동작 모드를 가진다. 먼저 입력 모드는 외부 장치에서 신호를 읽어드려 주로 디지털 입력 핀으로 설정된다. 출력 모드는 마이크로컨트롤러가 외부 장치에서 신호를 출력하고 주로 출력 핀으로 설정된다. 주변장치의 상태가 변화할 때 마다 GPIO는 이벤트 인터럽트를 발생시키거나 주기적으로 상태를 폴링해 변화를 감지한다. 이러한 GPIO는 다양한 주변 장치와 상호작용할 수 있고 추가 하드웨어가 필요하지 않고 소프트웨어적으로 제어가 가능하며 소프트웨어를 통해 제어되기 때문에 사용자가 원하는 방식으로 구성할 수 있다는 장점이 있지만, 일부 애플리케이션에서는 속도가 느릴 수 있

고 제어가 복잡하다는 한계가 존재한다.

[Assignment 2]

- **ar**

  ar은 archiver라는 의미로 사용되며 주로 소스 코드 및 오브젝트 파일을 하나의 아카이브 파일로 결합하거나, 아카이브 파일에서 파일을 추출하고 수정하는데 사용된다.

  r 옵션을 사용해서 아카이브에 기존 파일을 교체하거나 새 파일을 삽입할 수 있다. r 옵션을 사용해서 현재 디렉토리에 모든 txt 파일을 포함하는 아카이브 test.a를 생성하였다. t 옵션을 사용해서 아카이브에 포함된 모든 파일 목록을 표시했다. p 옵션을 사용해서 아카이브에 포함된 파일의 내용을 직접 표시하였다. r 옵션을 사용해서 새 텍스트 파일인 d.txt를 기존 아카이브에 추가하였다. 마지막으로 d 옵션을 사용해서 방금 추가한 d.txt를 아카이브에서 삭제하였다.

- **as**

  as는 어셈블리 코드를 컴파일하여 오브젝트 파일을 생성하는 데 사용된다.

```
sslab@ubuntu: ~/test
#include <stdio.h>

int main()
{
    printf("Hello, ARM Linux!\n");
    return 0;
}
~
~
```

hello.c 코드

```
sslab@ubuntu:~/test$ ls
a.txt  b.txt  c.txt  d.txt  hello.c  test.a
sslab@ubuntu:~/test$ vi hello.c
sslab@ubuntu:~/test$ gcc -S hello.c
sslab@ubuntu:~/test$ ls
a.txt  b.txt  c.txt  d.txt  hello.c  hello.s  test.a
sslab@ubuntu:~/test$ as -o hello.o hello.s
sslab@ubuntu:~/test$ ls
a.txt  b.txt  c.txt  d.txt  hello.c  hello.o  hello.s  test.a
sslab@ubuntu:~/test$ gcc -o hello hello.o
sslab@ubuntu:~/test$ ./hello
Hello, ARM Linux!
sslab@ubuntu:~/test$
```

우선 hello.c를 gcc -S로 컴파일해서 어셈블리언어로 변환하였다. 이후 as 명령어를 사용해서 어셈블리언어를 컴파일 하여 오브젝트 파일을 생성하였다. 이후 실행파일을 생성하였고 실행파일을 실행하여 Hello Arm Linux가 출력되는 것을 확인해보았다.

man as 명령어를 통해 as의 옵션들에 대해 살펴보았다. -o 옵션은 출력 파일의 이름을 지정할 수 있고 -g 옵션은 디버깅 정보를 포함해 오브젝트 파일을 생성할 수 있다. 또 -f 옵션은 출력 파일의 형식을 저장하는 역할을 하며 이외에도 많은 옵션들이 있는 것을 확인했다.

- **g++**

g++은 GNU C++ 컴파일러로, C++ 코드를 컴파일하고 링크하여 실행 파일을 생성한다.

```
#include <iostream>

int main()
{
        std::cout << "Hello, World!" << std::endl;
        return 0;
}
```

hello.cpp 코드

```
sslab@ubuntu:~/test/c++$ vi hello.cpp
sslab@ubuntu:~/test/c++$ g++ -o hello hello.cpp
sslab@ubuntu:~/test/c++$ ./hello
Hello, World!
sslab@ubuntu:~/test/c++$
```

hello.cpp 파일을 생성한 후 g++의 -o 옵션을 사용해 c++ 소스 코드를 컴파일하였다. -o 옵션은 .cpp 파일을 컴파일 해 실행파일을 생성한 후 실행파일의 이름을 hello로 지정하였다. 이때 -o 옵션을 사용하지 않으면 기본적으로 a.out으로 지정된다. 생성된 실행파일을 실행 시 Hello World가 출력되는 것을 확인할 수 있다.

```
sslab@ubuntu:~/test/c++$ ls
hello.cpp
sslab@ubuntu:~/test/c++$ g++ -S hello.cpp
sslab@ubuntu:~/test/c++$ ls
hello.cpp  hello.s
sslab@ubuntu:~/test/c++$ g++ -c hello.cpp
sslab@ubuntu:~/test/c++$ ls
hello.cpp  hello.o  hello.s
sslab@ubuntu:~/test/c++$
```

g++의 다른 옵션들도 사용해보았다. 먼저 -S 옵션은 컴파일만 하는 경우에 사용한다. 위 예시처럼 hello.cpp 파일을 컴파일해서 어셈블리 코드를 생성하지만 링크 단계를 거치지 않는다. 위 사진처럼 .s 확장자를 가진 어셈블리 코드가 생성된다. 다음으로 -c옵션을 사용하면 컴파일을 해서 오브젝트 파일을 생성한다. 링크 단계는 수행하지 않고 .o 옵션을 가진 오브젝트 파일이 생성된 것을 확인할 수 있다.

- **gcc**

gcc는 GNU C 컴파일러로, C 코드를 컴파일하고 링크하여 실행 파일을 생성한다.

```c
#include <stdio.h>

int main()
{
    printf("Hello, World!\n");
    return 0;
}
~
```

hello.c 코드

```
sslab@ubuntu:~/test/gcc$ vi hello.c
sslab@ubuntu:~/test/gcc$ gcc -o hello hello.c
sslab@ubuntu:~/test/gcc$ ls
hello  hello.c
sslab@ubuntu:~/test/gcc$ ./hello
Hello, World!
sslab@ubuntu:~/test/gcc$
```

hello.c 파일을 생성한 후 gcc의 -o 옵션을 사용해 c++ 소스 코드를 컴파일하였다. -o 옵션은 .c 파일을 컴파일 해 실행파일을 생성한 후 실행파일의 이름을 hello로 지정하였다. 이때 -o 옵션을 사용하지 않으면 기본적으로 a.out으로 지정된다. 생성된 실행파일을 실행 시 Hello World가 출력되는 것을 확인할 수 있다.

```
sslab@ubuntu:~/test/gcc$ gcc -E hello.c
# 1 "hello.c"
# 1 "<built-in>"
# 1 "<command-line>"
# 31 "<command-line>"
# 1 "/usr/include/stdc-predef.h" 1 3 4
# 32 "<command-line>" 2
# 1 "hello.c"
# 1 "/usr/include/stdio.h" 1 3 4
# 27 "/usr/include/stdio.h" 3 4
# 1 "/usr/include/x86_64-linux-gnu/bits/libc-header-start.h" 1 3 4
# 33 "/usr/include/x86_64-linux-gnu/bits/libc-header-start.h" 3 4
# 1 "/usr/include/features.h" 1 3 4
# 461 "/usr/include/features.h" 3 4
# 1 "/usr/include/x86_64-linux-gnu/sys/cdefs.h" 1 3 4
# 452 "/usr/include/x86_64-linux-gnu/sys/cdefs.h" 3 4
# 1 "/usr/include/x86_64-linux-gnu/bits/wordsize.h" 1 3 4
# 453 "/usr/include/x86_64-linux-gnu/sys/cdefs.h" 2 3 4
# 1 "/usr/include/x86_64-linux-gnu/bits/long-double.h" 1 3 4
# 454 "/usr/include/x86_64-linux-gnu/sys/cdefs.h" 2 3 4
# 462 "/usr/include/features.h" 2 3 4
# 485 "/usr/include/features.h" 3 4
# 1 "/usr/include/x86_64-linux-gnu/gnu/stubs.h" 1 3 4
# 10 "/usr/include/x86_64-linux-gnu/gnu/stubs.h" 3 4
# 1 "/usr/include/x86_64-linux-gnu/gnu/stubs-64.h" 1 3 4
# 11 "/usr/include/x86_64-linux-gnu/gnu/stubs.h" 2 3 4
# 486 "/usr/include/features.h" 2 3 4
# 34 "/usr/include/x86_64-linux-gnu/bits/libc-header-start.h" 2 3 4
# 28 "/usr/include/stdio.h" 2 3 4
# 1 "/usr/lib/gcc/x86_64-linux-gnu/9/include/stddef.h" 1 3 4
# 209 "/usr/lib/gcc/x86_64-linux-gnu/9/include/stddef.h" 3 4
```

gcc에서 자주 사용되는 옵션 몇가지를 실습해보았다. 먼저 -E 옵션은 컴파일의 첫 단계인 전처리까지 실행한 결과를 화면에 출력한다.

```
extern void funlockfile (FILE *__stream) __attribute__ ((__nothrow__ , __leaf__));
# 858 "/usr/include/stdio.h" 3 4
extern int __uflow (FILE *);
extern int __overflow (FILE *, int);
# 873 "/usr/include/stdio.h" 3 4

# 2 "hello.c" 2


# 3 "hello.c"
int main()
{
    printf("Hello, World!\n");
    return 0;
}
sslab@ubuntu:~/test/gcc$ ls
hello  hello.c
sslab@ubuntu:~/test/gcc$ gcc -c hello.c
sslab@ubuntu:~/test/gcc$ ls
hello  hello.c  hello.o
```

위 사진에서 볼 수 있듯이 매우 많은 내용이 소스 파일 위에 붙는 것을 확인할 수 있다. 이처럼 -E 옵션만 주면 전처리된 결과가 화면에만 출력되고 파일로 저장되지 않는 것을 확인할 수 있었다. 또 다른 옵션인 -c 옵션은 전처리, 컴파일, 어셈블까지 실행하여 오브젝트 파일을 생성한 것을 확인할 수 있다.

- **ld**

ld는 링커로, 여러 오브젝트 파일과 라이브러리 파일을 결합하여 실행 파일을 생성한다.

sslab@ubuntu: ~/test/ld

```
#include <stdio.h>

void hello()
{
    printf("Hello, ");
}
```

hello.c 코드

sslab@ubuntu: ~/test/ld

```
#include <stdio.h>

void world()
{
    printf("world!\n");
}
```

world.c 코드

sslab@ubuntu: ~/test/ld

```
#include <stdio.h>

extern void hello();
extern void world();

int main()
{
    hello();
    world();
    return 0;
}
```

main.c 코드

```
sslab@ubuntu:~/test/ld$ ls
hello.c  main.c  world.c
sslab@ubuntu:~/test/ld$ gcc -c hello.c -o hello.o
sslab@ubuntu:~/test/ld$ gcc -c world.c -o world.o
sslab@ubuntu:~/test/ld$ gcc -c main.c -o main.o
sslab@ubuntu:~/test/ld$ ls
hello.c  hello.o  main.c  main.o  world.c  world.o
sslab@ubuntu:~/test/ld$  ld -dynamic-linker /lib64/ld-linux-x86-64.so.2 /usr/lib/x86_64-linux-gn
u/crt1.o /usr/lib/x86_64-linux-gnu/crti.o hello.o world.o main.o -lc /usr/lib/x86_64-linux-gnu/c
rtn.o -o hello_world
sslab@ubuntu:~/test/ld$ ls
hello.c  hello.o  hello_world  main.c  main.o  world.c  world.o
sslab@ubuntu:~/test/ld$ ./hello_world
Hello, world!
sslab@ubuntu:~/test/ld$
```

우선 위 사진에서 볼 수 있듯이 hello.c와 world.c 그리고 main.c를 간단하게 구현하였다. 이후 .c 파일을 컴파일해서 각각의 .o 파일을 생성하였다. 이후 ld 명령어 중 -dynamic-linker 옵션을 사용하여 동적 링크를 하였다. 처음에는 옵션 없이 진행하였는데, 아래와 같은 문제가 발생하였다.

```
sslab@ubuntu:~/test/ld$ vi hello.c
sslab@ubuntu:~/test/ld$ vi world.c
sslab@ubuntu:~/test/ld$ vi main.c
sslab@ubuntu:~/test/ld$ ls
hello.c  hello.o  main.c  main.o  world.c  world.o
sslab@ubuntu:~/test/ld$ ld -o hello_world main.o hello.o world.o -lc
ld: warning: cannot find entry symbol _start; defaulting to 0000000000401050
sslab@ubuntu:~/test/ld$ ld -o hello_world main.o hello.o world.o -lc -e main
sslab@ubuntu:~/test/ld$ ls
hello.c  hello.o  hello_world  main.c  main.o  world.c  world.o
sslab@ubuntu:~/test/ld$ ./hello_world
bash: ./hello_world: No such file or directory
sslab@ubuntu:~/test/ld$
```

우선 ld 후 뜨는 warning은 ld가 entry point를 찾지 못해 발생하는 경고로 -e 옵션을 사용하여 진입점을 main 함수로 설정해주었다. 하지만 실행파일을 실행했을 때 찾지 못하는 것을 확인할 수 있다. 여러 옵션을 주고 바꿔보며 실행해보았지만 해결하지 못하여 dynamic-linker 옵션을 사용하여 이 문제를 해결하였다. 자세히 설명해보면, 우선 -dynamic-linker /lib64/ld-linux-x86-64.so.2 이 부분은 실행 파일이 동적 링크를 사용하도록 동적 링크 로더의 경로를 지정해주었다. 이어서 /usr/lib/x86_64-linux-gnu/crt1.o, /usr/lib/x86_64-linux-gnu/crti.o, /usr/lib/x86_64-linux-gnu/crtn.o 이부분은 모두 아래의 사진과 같이 find 명령어를 사용해 찾았다.

```
sslab@ubuntu:~/test/ld$ find /usr/lib -name 'crt*.o' -o -name 'libc.so*'
/usr/lib/x86_64-linux-gnu/crt1.o
/usr/lib/x86_64-linux-gnu/libc.so.6
/usr/lib/x86_64-linux-gnu/crtn.o
/usr/lib/x86_64-linux-gnu/libc.so
/usr/lib/x86_64-linux-gnu/crti.o
/usr/lib/gcc/x86_64-linux-gnu/9/crtprec64.o
/usr/lib/gcc/x86_64-linux-gnu/9/crtoffloadend.o
/usr/lib/gcc/x86_64-linux-gnu/9/crtbeginT.o
/usr/lib/gcc/x86_64-linux-gnu/9/crtbeginS.o
/usr/lib/gcc/x86_64-linux-gnu/9/crtoffloadbegin.o
/usr/lib/gcc/x86_64-linux-gnu/9/crtfastmath.o
/usr/lib/gcc/x86_64-linux-gnu/9/crtprec32.o
/usr/lib/gcc/x86_64-linux-gnu/9/crtend.o
/usr/lib/gcc/x86_64-linux-gnu/9/crtprec80.o
/usr/lib/gcc/x86_64-linux-gnu/9/crtbegin.o
/usr/lib/gcc/x86_64-linux-gnu/9/crtoffloadtable.o
/usr/lib/gcc/x86_64-linux-gnu/9/crtendS.o
sslab@ubuntu:~/test/ld$
```

위 파일들은 C 프로그램을 실행하고 초기화 하고 종료하는데 필요한 파일들로, crt1.o는

프로그램의 진입점을 crti.o는 초기화, crtn.o는 종료 지점을 지정해주는 역할을 한다. 이어서 lc 옵션은 C 프로그램을 컴파일 때 필요한 부분이다. 이렇게 실행파일을 생성해준 후 실행파일을 실행하면 hello world가 잘 출력이 되는 것을 볼 수 있고 이를 통해 하나의 실행파일로 잘 링크가 된 것을 알 수 있다.

- **nm**

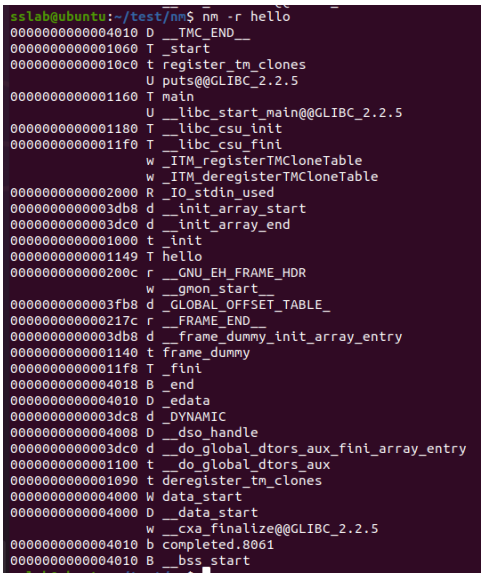
nm은 오브젝트 파일이나 실행 파일 내의 심볼 테이블을 보여주는 유틸리이다.

```c
#include <stdio.h>

void hello()
{
        printf("Hello\n");
}

int main()
{
        hello();
        return 0;
}
~
~
~
~
```

hello.c 코드

```
sslab@ubuntu:~/test/nm$ vi hello.c
sslab@ubuntu:~/test/nm$ gcc -o hello hello.c
sslab@ubuntu:~/test/nm$ nm hello
0000000000004010 B __bss_start
0000000000004010 b completed.8061
                 w __cxa_finalize@@GLIBC_2.2.5
0000000000004000 D __data_start
0000000000004000 W data_start
0000000000001090 t deregister_tm_clones
0000000000001100 t __do_global_dtors_aux
0000000000003dc0 d __do_global_dtors_aux_fini_array_entry
0000000000004008 D __dso_handle
0000000000003dc8 d _DYNAMIC
0000000000004010 D _edata
0000000000004018 B _end
00000000000011f8 T _fini
0000000000001140 t frame_dummy
0000000000003db8 d __frame_dummy_init_array_entry
000000000000217c r __FRAME_END__
0000000000003fb8 d _GLOBAL_OFFSET_TABLE_
                 w __gmon_start__
000000000000200c r __GNU_EH_FRAME_HDR
0000000000001149 T hello
0000000000001000 t _init
0000000000003dc0 d __init_array_end
0000000000003db8 d __init_array_start
0000000000002000 R _IO_stdin_used
                 w _ITM_deregisterTMCloneTable
                 w _ITM_registerTMCloneTable
00000000000011f0 T __libc_csu_fini
0000000000001180 T __libc_csu_init
                 U __libc_start_main@@GLIBC_2.2.5
0000000000001160 T main
                 U puts@@GLIBC_2.2.5
00000000000010c0 t register_tm_clones
0000000000001060 T _start
0000000000004010 D __TMC_END__
```

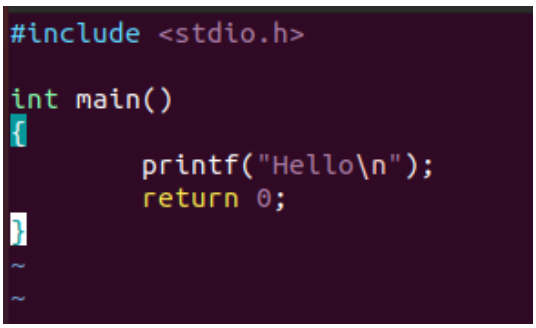
위 사진은 nm 명령어를 통해 생성된 hello 실행 파일에 포함된 심볼들의 목록을 보여주는 결과이다. 앞에서부터 주소 값, 타입, 심볼 이름의 형식을 가지고 있다. 위 사진처럼 옵션 없이 사용했을 때 실행파일에 포함된 모든 심볼들을 주소 값에 따라 정렬하여 출력한다. 대표적인 타입들에 대해서만 살펴보면 T는 Text 섹션에 위치한 코드를, D는 데이터 섹션에 위치한 초기화된 데이터를, B는 BSS(Block Started by Symbol) 섹션에 위치한 초기화되지 않은 데이터를, U는 정의되지 않은 심볼을 나타낸다. 예를 들어 hello.c 파일에서 정의된 main 함수는 T 타입의 심볼로 표시되어 있고, hello 함수도 T 타입의 심볼인 것을 알 수 있다.



왼쪽의 사진은 -p 옵션을 사용해 기호 테이블 순서대로 출력하는 것이며, 오른쪽 사진은 -r 옵션을 사용해 역순으로 분류한 결과이다.

- **objcopy**

objcopy는 오브젝트 파일에서 데이터를 복사하고 변환하는 유틸리티이다.



hello.c 코드

```
sslab@ubuntu:~/test/objcopy$ gcc -o hello hello.c
sslab@ubuntu:~/test/objcopy$ ls
hello  hello.c
sslab@ubuntu:~/test/objcopy$ objcopy hello hello_copy
sslab@ubuntu:~/test/objcopy$ ./hello_copy
Hello
sslab@ubuntu:~/test/objcopy$ ./hello
Hello
sslab@ubuntu:~/test/objcopy$
sslab@ubuntu:~/test/objcopy$
```

우선 hello 실행파일을 만들어준 후 hello 파일을 hello_copy로 복사해주었다. 기존의 hello와 동일한 결과를 나오는 것을 확인할 수 있다.

```
sslab@ubuntu:~/test/objcopy$ ls
hello  hello.c  hello_copy
sslab@ubuntu:~/test/objcopy$ objcopy -S hello new_hello
sslab@ubuntu:~/test/objcopy$ ls
hello  hello.c  hello_copy  new_hello
sslab@ubuntu:~/test/objcopy$ ./new_hello
Hello
sslab@ubuntu:~/test/objcopy$ ls -l hello new_hello
-rwxrwxr-x 1 sslab sslab 16696 Apr  7 03:26 hello
-rwxrwxr-x 1 sslab sslab 14472 Apr  7 03:34 new_hello
sslab@ubuntu:~/test/objcopy$
```

다음으로 -s 옵션을 사용한 결과이다. 이 옵션을 사용할 시 실행파일에서 재배치 정보와 심볼을 제거할 수 있다. 이로 인해 실행파일의 동작은 변하지 않으나 크기가 줄어든다. ls -l 옵션을 사용해 두 크기를 비교해보았다.

```
sslab@ubuntu:~/test/objcopy$ objcopy -O ihex hello hello.hex
sslab@ubuntu:~/test/objcopy$ cat hello.hex
:100318002F6C696236342F6C642D6C696E75782D7C
:0C0328007838362D36342E736F2E3200DC
:1003380004000001000000005000000474E5500B2
:10034800020000C004000000300000000000000DC
:1003580004000001400000003000000474E550090
:10036800E93F7D684C8370DB061BA5706A012BB5DD
:04037800E57AFF3AE9
:10037C000400000010000000010000000474E550072
:10038C0000000000030000000200000000000005C
:1003A000020000006000000010000000600000003E
:1003B000000810000000000600000000000000B6
:0403C000D165CE6DC8
:1003C8000000000000000000000000000000000025
:1003D8000000000000000003D00000020000000B8
:1003E8000000000000000000000000000000000005
:1003F8000B000000120000000000000000000000D8
:10040800000000000000000001F00000012000000B3
:1004180000000000000000000000000000000000D4
:1004280059000000200000000000000000000004B
:1004380000000000000000006800000020000002C
:1004480000000000000000000000000000000000A4
:1004580010000000220000000000000000000062
:08046800000000000000008C
:10047000006C6962632E736F2E36007075747300A2
:100480005F5F6378615F66696E616C697A65005F62
:100490005F6C6962635F73746172745F6D61696ED2
:1004A00000474C4942435F322E322E35005F49549B
:1004B0004D5F6465726567697374657544D436C12
:1004C0006F6E655461626C65005F5F676D6F6E5F34
:1004D00073746172745F5F005F49544D5F7265674A
:1004E000697374657544D436C6F6E655461626CD0
:0204F0006500A5
```

다음으로 -O 옵션으로 새로 생성할 파일의 포맷을 지정할 수 있는 것을 확인해보았다. Intel HEX 형식으로 변환한 결과이다. cat 명령어를 통해 변환이 되었다는 것을 확인했다.

- **objdump**

objdump는 라이브러리, 컴파일된 오브젝트 모듈 등의 바이너리 파일들의 정보를 보여주며 elf 파일을 어셈블리어로 보여주는 디스어셈블러로 사용될 수 있다.

```c
#include <stdio.h>

int main()
{
        printf("hello\n");
        return 0;
}
~
```

hello.c 코드

```
sslab@ubuntu:~/test/objdump$ objdump hello
Usage: objdump <option(s)> <file(s)>
 Display information from object <file(s)>.
 At least one of the following switches must be given:
  -a, --archive-headers    Display archive header information
  -f, --file-headers       Display the contents of the overall file header
  -p, --private-headers    Display object format specific file header contents
  -P, --private=OPT,OPT... Display object format specific contents
  -h, --[section-]headers  Display the contents of the section headers
  -x, --all-headers        Display the contents of all headers
  -d, --disassemble        Display assembler contents of executable sections
  -D, --disassemble-all    Display assembler contents of all sections
      --disassemble=<sym>  Display assembler contents from <sym>
  -S, --source             Intermix source code with disassembly
      --source-comment[=<txt>] Prefix lines of source code with <txt>
  -s, --full-contents      Display the full contents of all sections requested
  -g, --debugging          Display debug information in object file
  -e, --debugging-tags     Display debug information using ctags style
  -G, --stabs              Display (in raw form) any STABS info in the file
  -W[lLiaprmfFsoRtUuTgAckK] or
  --dwarf[=rawline,=decodedline,=info,=abbrev,=pubnames,=aranges,=macro,=frames,
          =frames-interp,=str,=loc,=Ranges,=pubtypes,
          =gdb_index,=trace_info,=trace_abbrev,=trace_aranges,
          =addr,=cu_index,=links,=follow-links]
                           Display DWARF info in the file
  --ctf=SECTION            Display CTF info from SECTION
  -t, --syms               Display the contents of the symbol table(s)
  -T, --dynamic-syms       Display the contents of the dynamic symbol table
  -r, --reloc              Display the relocation entries in the file
  -R, --dynamic-reloc      Display the dynamic relocation entries in the file
  @<file>                  Read options from <file>
  -v, --version            Display this program's version number
  -i, --info               List object formats and architectures supported
  -H, --help               Display this information
sslab@ubuntu:~/test/objdump$
```

우선 objdump를 옵션 없이 사용하면 다음과 같이 사용법 및 옵션의 목록이 표시된다.

```
sslab@ubuntu:~/test/objdump$ objdump -f hello

hello:     file format elf64-x86-64
architecture: i386:x86-64, flags 0x00000150:
HAS_SYMS, DYNAMIC, D_PAGED
start address 0x0000000000001060
```

이어서 f 옵션을 사용하면 파일의 헤더 정보를 알려준다. 파일 포맷, 아키텍처, 플래그, 시작 주소를 알려준다.

```
sslab@ubuntu:~/test/objdump$ objdump -D hello

hello:     file format elf64-x86-64


Disassembly of section .interp:

0000000000000318 <.interp>:
 318:   2f                      (bad)
 319:   6c                      insb   (%dx),%es:(%rdi)
 31a:   69 62 36 34 2f 6c 64    imul   $0x646c2f34,0x36(%rdx),%esp
 321:   2d 6c 69 6e 75          sub    $0x756e696c,%eax
 326:   78 2d                   js     355 <_init-0xcab>
 328:   78 38                   js     362 <_init-0xc9e>
 32a:   36 2d 36 34 2e 73       ss sub $0x732e3436,%eax
 330:   6f                      outsl  %ds:(%rsi),(%dx)
 331:   2e 32 00                xor    %cs:(%rax),%al

Disassembly of section .note.gnu.property:

0000000000000338 <.note.gnu.property>:
 338:   04 00                   add    $0x0,%al
 33a:   00 00                   add    %al,(%rax)
 33c:   10 00                   adc    %al,(%rax)
 33e:   00 00                   add    %al,(%rax)
 340:   05 00 00 00 47          add    $0x47000000,%eax
 345:   4e 55                   rex.WRX push %rbp
 347:   00 02                   add    %al,(%rdx)
 349:   00 00                   add    %al,(%rax)
 34b:   c0 04 00 00             rolb   $0x0,(%rax,%rax,1)
 34f:   00 03                   add    %al,(%rbx)
 351:   00 00                   add    %al,(%rax)
 353:   00 00                   add    %al,(%rax)
 355:   00 00                   add    %al,(%rax)
        ...

Disassembly of section .note.gnu.build-id:
```

다음으로 -D 옵션을 사용하면 파일의 모든 영역을 디어셈블 해준다. 만약 원하는 영역만 보고 싶을 땐 grep 명령을 사용해서 볼 수 있다.

```
sslab@ubuntu:~/test/objdump$ objdump -t hello

hello:     file format elf64-x86-64

SYMBOL TABLE:
0000000000000318 l    d  .interp        0000000000000000              .interp
0000000000000338 l    d  .note.gnu.property     0000000000000000              .note.gnu.property
0000000000000358 l    d  .note.gnu.build-id     0000000000000000              .note.gnu.build-id
000000000000037c l    d  .note.ABI-tag  0000000000000000              .note.ABI-tag
00000000000003a0 l    d  .gnu.hash      0000000000000000              .gnu.hash
00000000000003c8 l    d  .dynsym        0000000000000000              .dynsym
0000000000000470 l    d  .dynstr        0000000000000000              .dynstr
00000000000004f2 l    d  .gnu.version   0000000000000000              .gnu.version
0000000000000500 l    d  .gnu.version_r 0000000000000000              .gnu.version_r
0000000000000520 l    d  .rela.dyn      0000000000000000              .rela.dyn
00000000000005e0 l    d  .rela.plt      0000000000000000              .rela.plt
0000000000001000 l    d  .init  0000000000000000              .init
0000000000001020 l    d  .plt   0000000000000000              .plt
0000000000001040 l    d  .plt.got       0000000000000000              .plt.got
0000000000001050 l    d  .plt.sec       0000000000000000              .plt.sec
0000000000001060 l    d  .text  0000000000000000              .text
00000000000011e8 l    d  .fini  0000000000000000              .fini
0000000000002000 l    d  .rodata        0000000000000000              .rodata
000000000000200c l    d  .eh_frame_hdr  0000000000000000              .eh_frame_hdr
0000000000002050 l    d  .eh_frame      0000000000000000              .eh_frame
0000000000003db8 l    d  .init_array    0000000000000000              .init_array
0000000000003dc0 l    d  .fini_array    0000000000000000              .fini_array
0000000000003dc8 l    d  .dynamic       0000000000000000              .dynamic
0000000000003fb8 l    d  .got   0000000000000000              .got
0000000000004000 l    d  .data  0000000000000000              .data
0000000000004010 l    d  .bss   0000000000000000              .bss
0000000000000000 l    d  .comment       0000000000000000              .comment
0000000000000000 l    df *ABS*  0000000000000000              crtstuff.c
0000000000001090 l     F .text  0000000000000000              deregister_tm_clones
00000000000010c0 l     F .text  0000000000000000              register_tm_clones
0000000000001100 l     F .text  0000000000000000              __do_global_dtors_aux
0000000000004010 l     O .bss   0000000000000001              completed.8061
0000000000003dc0 l     O .fini_array     0000000000000000              __do_global_dtors_aux_fini
_array_entry
0000000000001140 l     F .text  0000000000000000              frame_dummy
0000000000003db8 l     O .init_array     0000000000000000              __frame_dummy_init_array_e
ntry
0000000000000000 l    df *ABS*  0000000000000000              hello.c
0000000000000000 l    df *ABS*  0000000000000000              crtstuff.c
0000000000002154 l     O .eh_frame       0000000000000000              __FRAME_END__
0000000000000000 l    df *ABS*  0000000000000000
0000000000003dc0 l       .init_array     0000000000000000              __init_array_end
0000000000003dc8 l     O .dynamic        0000000000000000              _DYNAMIC
0000000000003db8 l       .init_array     0000000000000000              __init_array_start
000000000000200c l       .eh_frame_hdr   0000000000000000              __GNU_EH_FRAME_HDR
0000000000003fb8 l     O .got    0000000000000000              _GLOBAL_OFFSET_TABLE_
0000000000001000 l     F .init   0000000000000000              _init
00000000000011e0 g     F .text  0000000000000005              __libc_csu_fini
```

다음으로 t 옵션을 사용할 경우 ELF 파일의 심볼 테이블을 확인할 수 있다.

```
sslab@ubuntu:~/test/objdump$ objdump -d hello

hello:     file format elf64-x86-64


Disassembly of section .init:

0000000000001000 <_init>:
    1000:       f3 0f 1e fa             endbr64
    1004:       48 83 ec 08             sub    $0x8,%rsp
    1008:       48 8b 05 d9 2f 00 00    mov    0x2fd9(%rip),%rax        # 3fe8 <__gmon_start__>
    100f:       48 85 c0                test   %rax,%rax
    1012:       74 02                   je     1016 <_init+0x16>
    1014:       ff d0                   callq  *%rax
    1016:       48 83 c4 08             add    $0x8,%rsp
    101a:       c3                      retq

Disassembly of section .plt:

0000000000001020 <.plt>:
    1020:       ff 35 9a 2f 00 00       pushq  0x2f9a(%rip)        # 3fc0 <_GLOBAL_OFFSET_TABLE_
+0x8>
    1026:       f2 ff 25 9b 2f 00 00    bnd jmpq *0x2f9b(%rip)        # 3fc8 <_GLOBAL_OFFSET_TAB
LE_+0x10>
    102d:       0f 1f 00                nopl   (%rax)
    1030:       f3 0f 1e fa             endbr64
    1034:       68 00 00 00 00          pushq  $0x0
    1039:       f2 e9 e1 ff ff ff       bnd jmpq 1020 <.plt>
    103f:       90                      nop

Disassembly of section .plt.got:

0000000000001040 <__cxa_finalize@plt>:
    1040:       f3 0f 1e fa             endbr64
    1044:       f2 ff 25 ad 2f 00 00    bnd jmpq *0x2fad(%rip)        # 3ff8 <__cxa_finalize@GLI
BC_2.2.5>
    104b:       0f 1f 44 00 00          nopl   0x0(%rax,%rax,1)

Disassembly of section .plt.sec:

0000000000001050 <puts@plt>:
    1050:       f3 0f 1e fa             endbr64
    1054:       f2 ff 25 75 2f 00 00    bnd jmpq *0x2f75(%rip)        # 3fd0 <puts@GLIBC_2.2.5>
    105b:       0f 1f 44 00 00          nopl   0x0(%rax,%rax,1)

Disassembly of section .text:
```

d 옵션을 사용할 경우 ELF 파일의 어셈블리어 코드를 확인할 수 있다.

```
sslab@ubuntu:~/test/objdump$ objdump -x hello

hello:     file format elf64-x86-64
hello
architecture: i386:x86-64, flags 0x00000150:
HAS_SYMS, DYNAMIC, D_PAGED
start address 0x0000000000001060

Program Header:
    PHDR off    0x0000000000000040 vaddr 0x0000000000000040 paddr 0x0000000000000040 align 2**3
         filesz 0x00000000000002d8 memsz 0x00000000000002d8 flags r--
  INTERP off    0x0000000000000318 vaddr 0x0000000000000318 paddr 0x0000000000000318 align 2**0
         filesz 0x000000000000001c memsz 0x000000000000001c flags r--
    LOAD off    0x0000000000000000 vaddr 0x0000000000000000 paddr 0x0000000000000000 align 2**12
         filesz 0x00000000000005f8 memsz 0x00000000000005f8 flags r--
    LOAD off    0x0000000000001000 vaddr 0x0000000000001000 paddr 0x0000000000001000 align 2**12
         filesz 0x00000000000001f5 memsz 0x00000000000001f5 flags r-x
    LOAD off    0x0000000000002000 vaddr 0x0000000000002000 paddr 0x0000000000002000 align 2**12
         filesz 0x0000000000000158 memsz 0x0000000000000158 flags r--
    LOAD off    0x0000000000002db8 vaddr 0x0000000000003db8 paddr 0x0000000000003db8 align 2**12
         filesz 0x0000000000000258 memsz 0x0000000000000260 flags rw-
 DYNAMIC off    0x0000000000002dc8 vaddr 0x0000000000003dc8 paddr 0x0000000000003dc8 align 2**3
         filesz 0x00000000000001f0 memsz 0x00000000000001f0 flags rw-
    NOTE off    0x0000000000000338 vaddr 0x0000000000000338 paddr 0x0000000000000338 align 2**3
         filesz 0x0000000000000020 memsz 0x0000000000000020 flags r--
    NOTE off    0x0000000000000358 vaddr 0x0000000000000358 paddr 0x0000000000000358 align 2**2
         filesz 0x0000000000000044 memsz 0x0000000000000044 flags r--
0x6474e553 off    0x0000000000000338 vaddr 0x0000000000000338 paddr 0x0000000000000338 align 2**
3
         filesz 0x0000000000000020 memsz 0x0000000000000020 flags r--
EH_FRAME off    0x000000000000200c vaddr 0x000000000000200c paddr 0x000000000000200c align 2**2
         filesz 0x0000000000000044 memsz 0x0000000000000044 flags r--
   STACK off    0x0000000000000000 vaddr 0x0000000000000000 paddr 0x0000000000000000 align 2**4
         filesz 0x0000000000000000 memsz 0x0000000000000000 flags rw-
   RELRO off    0x0000000000002db8 vaddr 0x0000000000003db8 paddr 0x0000000000003db8 align 2**0
         filesz 0x0000000000000248 memsz 0x0000000000000248 flags r--

Dynamic Section:
  NEEDED               libc.so.6
  INIT                 0x0000000000001000
  FINI                 0x00000000000011e8
  INIT_ARRAY           0x0000000000003db8
  INIT_ARRAYSZ         0x0000000000000008
  FINI_ARRAY           0x0000000000003dc0
  FINI_ARRAYSZ         0x0000000000000008
  GNU_HASH             0x00000000000003a0
```

x 옵션을 사용할 경우 바이너리 파일의 기본 정보를 확인할 수 있으며 바이너리 파일의 섹션, 헤더 등을 출력한다.

- **readelf**

readelf는 오브젝트 파일이나 실행 파일의 ELF(Executable and Linkable Format) 포맷 정보를 보여주는 유틸리티이다.

```c
#include <stdio.h>

int main()
{
        printf("hello\n");
        return 0;
}
~
```

hello.c 코드

```
sslab@ubuntu:~/test/readelf$ vi hello.c
sslab@ubuntu:~/test/readelf$ gcc -o hello hello.c
sslab@ubuntu:~/test/readelf$ ls
hello  hello.c
sslab@ubuntu:~/test/readelf$ readelf hello
Usage: readelf <option(s)> elf-file(s)
 Display information about the contents of ELF format files
 Options are:
  -a --all               Equivalent to: -h -l -S -s -r -d -V -A -I
  -h --file-header       Display the ELF file header
  -l --program-headers   Display the program headers
     --segments          An alias for --program-headers
  -S --section-headers   Display the sections' header
     --sections          An alias for --section-headers
  -g --section-groups    Display the section groups
  -t --section-details   Display the section details
  -e --headers           Equivalent to: -h -l -S
  -s --syms              Display the symbol table
     --symbols           An alias for --syms
  --dyn-syms             Display the dynamic symbol table
  -n --notes             Display the core notes (if present)
  -r --relocs            Display the relocations (if present)
  -u --unwind            Display the unwind info (if present)
  -d --dynamic           Display the dynamic section (if present)
  -V --version-info      Display the version sections (if present)
  -A --arch-specific     Display architecture specific information (if any)
  -c --archive-index     Display the symbol/file index in an archive
  -D --use-dynamic       Use the dynamic section info when displaying symbols
  -x --hex-dump=<number|name>
                         Dump the contents of section <number|name> as bytes
  -p --string-dump=<number|name>
                         Dump the contents of section <number|name> as strings
  -R --relocated-dump=<number|name>
                         Dump the contents of section <number|name> as relocated bytes
  -z --decompress        Decompress section before dumping it
  -w[lLiaprmfFsoRtUuTgAckK] or
  --debug-dump[=rawline,=decodedline,=info,=abbrev,=pubnames,=aranges,=macro,=frames,
               =frames-interp,=str,=loc,=Ranges,=pubtypes,
               =gdb_index,=trace_info,=trace_abbrev,=trace_aranges,
               =addr,=cu_index,=links,=follow-links]
                         Display the contents of DWARF debug sections
  --dwarf-depth=N        Do not display DIEs at depth N or greater
  --dwarf-start=N        Display DIEs starting with N, at the same depth
                         or deeper
  --ctf=<number|name>    Display CTF info from section <number|name>
  --ctf-parent=<number|name>
                         Use section <number|name> as the CTF parent

  --ctf-symbols=<number|name>
                         Use section <number|name> as the CTF external symtab

  --ctf-strings=<number|name>
```

우선 아무 옵션 없이 사용할 경우 다음과 같이 사용법 및 옵션의 목록이 표시된다

```
sslab@ubuntu:~/test/readelf$ readelf -h hello
ELF Header:
  Magic:   7f 45 4c 46 02 01 01 00 00 00 00 00 00 00 00 00
  Class:                             ELF64
  Data:                              2's complement, little endian
  Version:                           1 (current)
  OS/ABI:                            UNIX - System V
  ABI Version:                       0
  Type:                              DYN (Shared object file)
  Machine:                           Advanced Micro Devices X86-64
  Version:                           0x1
  Entry point address:               0x1060
  Start of program headers:          64 (bytes into file)
  Start of section headers:          14712 (bytes into file)
  Flags:                             0x0
  Size of this header:               64 (bytes)
  Size of program headers:           56 (bytes)
  Number of program headers:         13
  Size of section headers:           64 (bytes)
  Number of section headers:         31
  Section header string table index: 30
```

h 옵션을 사용할 경우 바이너리 파일의 기본 정보를 표시한다. 즉 바이너리 파일의 ELF

헤더를 표시해 파일의 기본 정보를 제시하고 있다.

```
sslab@ubuntu:~/test/readelf$ readelf -S hello
There are 31 section headers, starting at offset 0x3978:

Section Headers:
  [Nr] Name              Type             Address           Offset
       Size              EntSize          Flags  Link  Info  Align
  [ 0]                   NULL             0000000000000000  00000000
       0000000000000000  0000000000000000           0     0     0
  [ 1] .interp           PROGBITS         0000000000000318  00000318
       000000000000001c  0000000000000000   A       0     0     1
  [ 2] .note.gnu.propert NOTE             0000000000000338  00000338
       0000000000000020  0000000000000000   A       0     0     8
  [ 3] .note.gnu.build-i NOTE             0000000000000358  00000358
       0000000000000024  0000000000000000   A       0     0     4
  [ 4] .note.ABI-tag     NOTE             000000000000037c  0000037c
       0000000000000020  0000000000000000   A       0     0     4
  [ 5] .gnu.hash         GNU_HASH         00000000000003a0  000003a0
       0000000000000024  0000000000000000   A       6     0     8
  [ 6] .dynsym           DYNSYM           00000000000003c8  000003c8
       00000000000000a8  0000000000000018   A       7     1     8
  [ 7] .dynstr           STRTAB           0000000000000470  00000470
       0000000000000082  0000000000000000   A       0     0     1
  [ 8] .gnu.version      VERSYM           00000000000004f2  000004f2
       000000000000000e  0000000000000002   A       6     0     2
  [ 9] .gnu.version_r    VERNEED          0000000000000500  00000500
       0000000000000020  0000000000000000   A       7     1     8
  [10] .rela.dyn         RELA             0000000000000520  00000520
       00000000000000c0  0000000000000018   A       6     0     8
  [11] .rela.plt         RELA             00000000000005e0  000005e0
       0000000000000018  0000000000000018  AI       6    24     8
  [12] .init             PROGBITS         0000000000001000  00001000
       000000000000001b  0000000000000000  AX       0     0     4
  [13] .plt              PROGBITS         0000000000001020  00001020
       0000000000000020  0000000000000010  AX       0     0     16
  [14] .plt.got          PROGBITS         0000000000001040  00001040
       0000000000000010  0000000000000010  AX       0     0     16
  [15] .plt.sec          PROGBITS         0000000000001050  00001050
       0000000000000010  0000000000000010  AX       0     0     16
  [16] .text             PROGBITS         0000000000001060  00001060
       0000000000000185  0000000000000000  AX       0     0     16
  [17] .fini             PROGBITS         00000000000011e8  000011e8
       000000000000000d  0000000000000000  AX       0     0     4
  [18] .rodata           PROGBITS         0000000000002000  00002000
       000000000000000a  0000000000000000   A       0     0     4
  [19] .eh_frame_hdr     PROGBITS         000000000000200c  0000200c
       0000000000000044  0000000000000000   A       0     0     4
```

다음으로 S옵션을 사용할 경우 바이너리 파일의 섹션 헤더 정보를 표시해 파일 내의 섹션들과 각 섹션의 속성 및 크기를 보여준다.

```
sslab@ubuntu:~/test/readelf$ readelf -s hello

Symbol table '.dynsym' contains 7 entries:
   Num:    Value          Size Type    Bind   Vis      Ndx Name
     0: 0000000000000000     0 NOTYPE  LOCAL  DEFAULT  UND
     1: 0000000000000000     0 NOTYPE  WEAK   DEFAULT  UND _ITM_deregisterTMCloneTab
     2: 0000000000000000     0 FUNC    GLOBAL DEFAULT  UND puts@GLIBC_2.2.5 (2)
     3: 0000000000000000     0 FUNC    GLOBAL DEFAULT  UND __libc_start_main@GLIBC_2.2.5 (2)
     4: 0000000000000000     0 NOTYPE  WEAK   DEFAULT  UND __gmon_start__
     5: 0000000000000000     0 NOTYPE  WEAK   DEFAULT  UND _ITM_registerTMCloneTable
     6: 0000000000000000     0 FUNC    WEAK   DEFAULT  UND __cxa_finalize@GLIBC_2.2.5 (2)

Symbol table '.symtab' contains 65 entries:
   Num:    Value          Size Type    Bind   Vis      Ndx Name
     0: 0000000000000000     0 NOTYPE  LOCAL  DEFAULT  UND
     1: 0000000000000318     0 SECTION LOCAL  DEFAULT    1
     2: 0000000000000338     0 SECTION LOCAL  DEFAULT    2
     3: 0000000000000358     0 SECTION LOCAL  DEFAULT    3
     4: 000000000000037c     0 SECTION LOCAL  DEFAULT    4
     5: 00000000000003a0     0 SECTION LOCAL  DEFAULT    5
     6: 00000000000003c8     0 SECTION LOCAL  DEFAULT    6
     7: 0000000000000470     0 SECTION LOCAL  DEFAULT    7
     8: 00000000000004f2     0 SECTION LOCAL  DEFAULT    8
     9: 0000000000000500     0 SECTION LOCAL  DEFAULT    9
    10: 0000000000000520     0 SECTION LOCAL  DEFAULT   10
    11: 00000000000005e0     0 SECTION LOCAL  DEFAULT   11
    12: 0000000000001000     0 SECTION LOCAL  DEFAULT   12
    13: 0000000000001020     0 SECTION LOCAL  DEFAULT   13
    14: 0000000000001040     0 SECTION LOCAL  DEFAULT   14
    15: 0000000000001050     0 SECTION LOCAL  DEFAULT   15
    16: 0000000000001060     0 SECTION LOCAL  DEFAULT   16
    17: 00000000000011e8     0 SECTION LOCAL  DEFAULT   17
    18: 0000000000002000     0 SECTION LOCAL  DEFAULT   18
    19: 000000000000200c     0 SECTION LOCAL  DEFAULT   19
    20: 0000000000002050     0 SECTION LOCAL  DEFAULT   20
    21: 0000000000003db8     0 SECTION LOCAL  DEFAULT   21
    22: 0000000000003dc0     0 SECTION LOCAL  DEFAULT   22
    23: 0000000000003dc8     0 SECTION LOCAL  DEFAULT   23
    24: 0000000000003fb8     0 SECTION LOCAL  DEFAULT   24
    25: 0000000000004000     0 SECTION LOCAL  DEFAULT   25
    26: 0000000000004010     0 SECTION LOCAL  DEFAULT   26
    27: 0000000000000000     0 SECTION LOCAL  DEFAULT   27
    28: 0000000000000000     0 FILE    LOCAL  DEFAULT  ABS crtstuff.c
    29: 0000000000001090     0 FUNC    LOCAL  DEFAULT   16 deregister_tm_clones
    30: 00000000000010c0     0 FUNC    LOCAL  DEFAULT   16 register_tm_clones
    31: 0000000000001100     0 FUNC    LOCAL  DEFAULT   16 __do_global_dtors_aux
    32: 0000000000004010     1 OBJECT  LOCAL  DEFAULT   26 completed.8061
    33: 0000000000003dc0     0 OBJECT  LOCAL  DEFAULT   22 __do_global_dtors_aux_fin
    34: 0000000000001140     0 FUNC    LOCAL  DEFAULT   16 frame_dummy
    35: 0000000000003db8     0 OBJECT  LOCAL  DEFAULT   21 __frame_dummy_init_array_
    36: 0000000000000000     0 FILE    LOCAL  DEFAULT  ABS hello.c
```

다음으로 s 옵션을 사용할 경우 바이너리 파일의 테이블 정보를 표시해 파일 내 정의된 함수, 변수 등의 심볼을 보여준다.

```
sslab@ubuntu:~/test/readelf$ readelf -l hello

Elf file type is DYN (Shared object file)
Entry point 0x1060
There are 13 program headers, starting at offset 64

Program Headers:
  Type           Offset             VirtAddr           PhysAddr
                 FileSiz            MemSiz              Flags  Align
  PHDR           0x0000000000000040 0x0000000000000040 0x0000000000000040
                 0x00000000000002d8 0x00000000000002d8  R      0x8
  INTERP         0x0000000000000318 0x0000000000000318 0x0000000000000318
                 0x000000000000001c 0x000000000000001c  R      0x1
      [Requesting program interpreter: /lib64/ld-linux-x86-64.so.2]
  LOAD           0x0000000000000000 0x0000000000000000 0x0000000000000000
                 0x00000000000005f8 0x00000000000005f8  R      0x1000
  LOAD           0x0000000000001000 0x0000000000001000 0x0000000000001000
                 0x00000000000001f5 0x00000000000001f5  R E    0x1000
  LOAD           0x0000000000002000 0x0000000000002000 0x0000000000002000
                 0x0000000000000158 0x0000000000000158  R      0x1000
  LOAD           0x0000000000002db8 0x0000000000003db8 0x0000000000003db8
                 0x0000000000000258 0x0000000000000260  RW     0x1000
  DYNAMIC        0x0000000000002dc8 0x0000000000003dc8 0x0000000000003dc8
                 0x00000000000001f0 0x00000000000001f0  RW     0x8
  NOTE           0x0000000000000338 0x0000000000000338 0x0000000000000338
                 0x0000000000000020 0x0000000000000020  R      0x8
  NOTE           0x0000000000000358 0x0000000000000358 0x0000000000000358
                 0x0000000000000044 0x0000000000000044  R      0x4
  GNU_PROPERTY   0x0000000000000338 0x0000000000000338 0x0000000000000338
                 0x0000000000000020 0x0000000000000020  R      0x8
  GNU_EH_FRAME   0x000000000000200c 0x000000000000200c 0x000000000000200c
                 0x0000000000000044 0x0000000000000044  R      0x4
  GNU_STACK      0x0000000000000000 0x0000000000000000 0x0000000000000000
                 0x0000000000000000 0x0000000000000000  RW     0x10
  GNU_RELRO      0x0000000000002db8 0x0000000000003db8 0x0000000000003db8
                 0x0000000000000248 0x0000000000000248  R      0x1

 Section to Segment mapping:
  Segment Sections...
   00
   01     .interp
   02     .interp .note.gnu.property .note.gnu.build-id .note.ABI-tag .gnu.hash .dynsym .dynstr
.gnu.version .gnu.version_r .rela.dyn .rela.plt
   03     .init .plt .plt.got .plt.sec .text .fini
   04     .rodata .eh_frame_hdr .eh_frame
   05     .init_array .fini_array .dynamic .got .data .bss
   06     .dynamic
```

l 옵션을 사용할 경우 바이너리 파일의 프로그램 헤더 정보를 표시해 실행 가능한 섹션과 해당 메모리에 로딩될 때의 정보를 보여준다.

- **size**

size는 오브젝트 파일이나 실행 파일의 섹션 별 크기를 보여주는 유틸리티이다.

```c
#include <stdio.h>

int main()
{
        printf("hello\n");
        return 0;
}
~
```

hello.c 코드

```
sslab@ubuntu:~/test/size$ vi hello.c
sslab@ubuntu:~/test/size$ gcc -o hello hello.c
sslab@ubuntu:~/test/size$ size hello
   text    data     bss     dec     hex filename
   1559     600       8    2167     877 hello
sslab@ubuntu:~/test/size$ size -A hello
hello  :
section              size    addr
.interp                28     792
.note.gnu.property     32     824
.note.gnu.build-id     36     856
.note.ABI-tag          32     892
.gnu.hash              36     928
.dynsym               168     968
.dynstr               130    1136
.gnu.version           14    1266
.gnu.version_r         32    1280
.rela.dyn             192    1312
.rela.plt              24    1504
.init                  27    4096
.plt                   32    4128
.plt.got               16    4160
.plt.sec               16    4176
.text                 389    4192
.fini                  13    4584
.rodata                10    8192
.eh_frame_hdr          68    8204
.eh_frame             264    8272
.init_array             8   15800
.fini_array             8   15808
.dynamic              496   15816
.got                   72   16312
.data                  16   16384
.bss                    8   16400
.comment               43       0
Total                2210
```

우선 옵션 없이 사용할 경우 실행파일의 섹션 정보를 확인할 수 있다. 해당 파일의 섹션 별 크기를 표시하는 코드, 데이터 및 BSS섹션의 크기를 확인할 수 있다. 이 부분을 통해 프로그램의 크기와 메모리 사용량을 확인할 수 있다. 이어서 A 옵션을 사용한 경우 모든 섹션의 크기를 표시할 수 있다.

```
sslab@ubuntu:~/test/size$ size -t hello
   text    data     bss     dec     hex filename
   1559     600       8    2167     877 hello
   1559     600       8    2167     877 (TOTALS)
sslab@ubuntu:~/test/size$
```

다음으로 t 옵션을 사용한 경우 섹션의 합계를 표시할 수 있다.

```
sslab@ubuntu:~/test/size$ size -V hello
GNU size (GNU Binutils for Ubuntu) 2.34
Copyright (C) 2020 Free Software Foundation, Inc.
This program is free software; you may redistribute it under the terms of
the GNU General Public License version 3 or (at your option) any later version.
This program has absolutely no warranty.
sslab@ubuntu:~/test/size$
```

V 옵션을 사용한 경우 size 명령어의 버전 정보를 표시할 수 있다.

- **strings**

strings는 바이너리 파일에서 문자열을 추출하여 출력한다.

```c
#include <stdio.h>

int main()
{
        printf("hello\n");
        return 0;
}
~
```

hello.c 코드

```
sslab@ubuntu:~/test/strings$ vi hello.c
sslab@ubuntu:~/test/strings$ gcc -o hello hello.c
sslab@ubuntu:~/test/strings$ strings hello
/lib64/ld-linux-x86-64.so.2
/aWSe
libc.so.6
puts
__cxa_finalize
__libc_start_main
GLIBC_2.2.5
_ITM_deregisterTMCloneTable
__gmon_start__
_ITM_registerTMCloneTable
u+UH
[]A\A]A^A_
hello
:*3$"
GCC: (Ubuntu 9.4.0-1ubuntu1~20.04.2) 9.4.0
crtstuff.c
deregister_tm_clones
__do_global_dtors_aux
completed.8061
__do_global_dtors_aux_fini_array_entry
frame_dummy
__frame_dummy_init_array_entry
hello.c
__FRAME_END__
__init_array_end
_DYNAMIC
__init_array_start
__GNU_EH_FRAME_HDR
_GLOBAL_OFFSET_TABLE_
__libc_csu_fini
_ITM_deregisterTMCloneTable
puts@@GLIBC_2.2.5
_edata
__libc_start_main@@GLIBC_2.2.5
__data_start
__gmon_start__
__dso_handle
_IO_stdin_used
__libc_csu_init
```

위 사진과 같이 strings 명령어를 옵션 없이 사용했을 경우 해당 파일에서 추출된 모든 문자열을 표시한다.

```
sslab@ubuntu:~/test/strings$ strings -n 16 hello
/lib64/ld-linux-x86-64.so.2
__libc_start_main
_ITM_deregisterTMCloneTable
_ITM_registerTMCloneTable
GCC: (Ubuntu 9.4.0-1ubuntu1~20.04.2) 9.4.0
deregister_tm_clones
__do_global_dtors_aux
__do_global_dtors_aux_fini_array_entry
__frame_dummy_init_array_entry
__init_array_end
__init_array_start
__GNU_EH_FRAME_HDR
_GLOBAL_OFFSET_TABLE_
_ITM_deregisterTMCloneTable
puts@@GLIBC_2.2.5
__libc_start_main@@GLIBC_2.2.5
_ITM_registerTMCloneTable
__cxa_finalize@@GLIBC_2.2.5
.note.gnu.property
.note.gnu.build-id
```

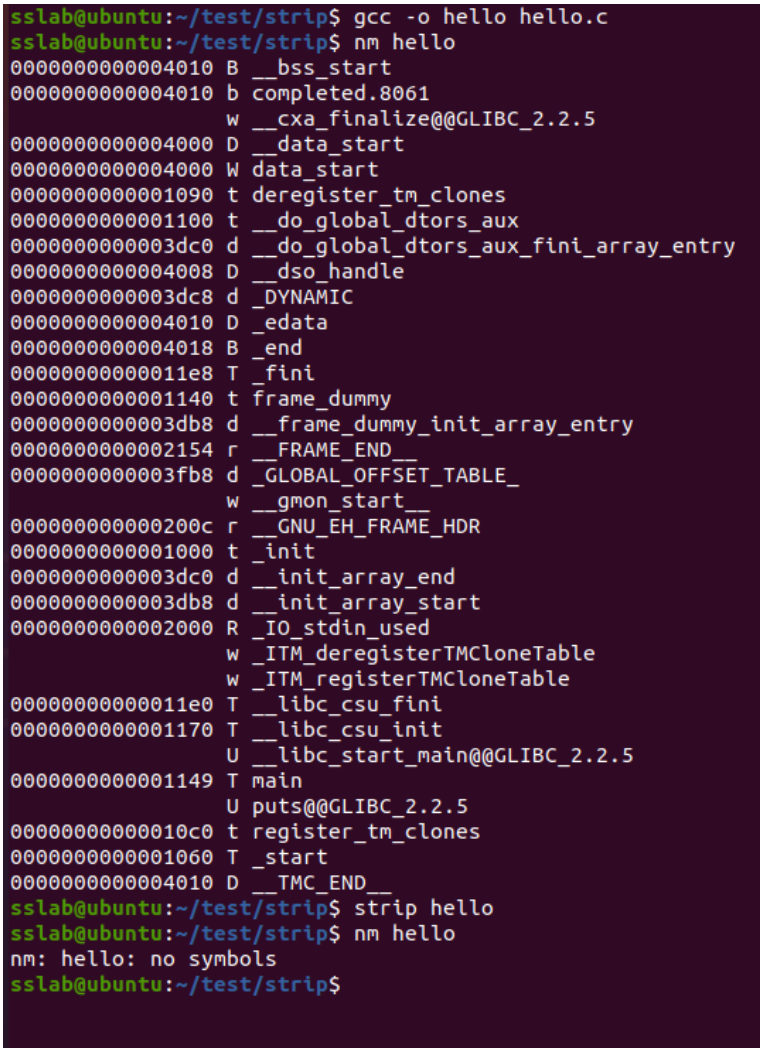
n 옵션 뒤에는 숫자가 오며 이 숫자는 출력되는 문자열의 최소 길이를 나타낸다. 위 처럼 16을 넣을 경우 출력되는 문자열이 최소한 16개의 문자로 이루어져야 함을 의미한다. 즉 16개 미만의 문자로 이루어진 문자열을 출력되지 않는다.

- **strip**

실행 파일에서 디버그 정보와 심볼을 제거하여 파일 크기를 줄이는 유틸리티이다.

```c
#include <stdio.h>

int main()
{
        printf("hello\n");
        return 0;
}
~
```

hello.c 코드



위 사진에서 볼 수 있듯이 nm 명령어로 심볼을 확인한 후 strip 명령어를 사용해서 모든 심볼을 제거한 후 다시 nm 명령어로 모든 심볼이 제거된 것을 확인하였다. strip 명령어의 경우 옵션을 사용하지 않거나 -s 옵션을 사용하면 모든 심볼을 제거할 수 있다.

[참고자료]

- **I2C**

https://mintnlatte.tistory.com/201

https://trts1004.tistory.com/12109609

https://blog.naver.com/tnsehf12345/220100382674

- **SPI**

https://advancedtestingservices.tistory.com/600

https://mintnlatte.tistory.com/199

https://blog.naver.com/techref/222303368153

- **SDIO**

https://trts1004.tistory.com/12109213

- **PCI**

https://blog.naver.com/ycpiglet/222139210239

- **USB**

https://mintnlatte.tistory.com/203

- **CAN**

https://www.ni.com/ko/shop/seamlessly-connect-to-third-party-devices-and-supervisory-system/controller-area-network--can--overview.html

- **UART**

https://mintnlatte.tistory.com/202

- **GPIO**

https://mintnlatte.tistory.com/200

- **ar**

https://ko.linux-console.net/?p=3665

- **as**

https://www.geeksforgeeks.org/as-command-in-linux-with-examples/

- **g++**

https://htsstory.tistory.com/entry/g-%EA%B8%B0%EB%B3%B8-%EC%98%B5%EC%85%98

- **gcc**

https://seamless.tistory.com/2

- **ld**

https://junsoolee.gitbook.io/linux-insides-ko/summary/misc/linux-misc-3

https://jayy-h.tistory.com/10

http://korea.gnu.org/manual/release/ld/ld-sjp/ld-ko_2.html

- **nm**

https://www.ibm.com/docs/ko/aix/7.2?topic=n-nm-command

- **objcopy**

https://itwiki.kr/w/%EB%A6%AC%EB%88%85%EC%8A%A4_objcopy

https://damduc.tistory.com/147

- **objdump**

https://smleenull.tistory.com/213

- **readelf**

https://itwiki.kr/w/%EB%A6%AC%EB%88%85%EC%8A%A4_readelf

- **size**

https://chanchan-father.tistory.com/341

- **strings**

https://www.ibm.com/docs/ko/aix/7.2?topic=s-strings-command

- **strip**

https://itwiki.kr/w/%EB%A6%AC%EB%88%85%EC%8A%A4_strip