동국대학교 경주캠퍼스

컴퓨터공학과

이은진

dldmswls4482@naver.com

Linux 환경에서 Cgroups를 활용한  
I/O weight에 따른 성능 분석

FIO 벤치마크 사용

**목차**

[1. Cgroup 정의 3](#_Toc35958151)

[**1-1. Cgroup이란?** 3](#_Toc35958152)

[**1-2. Cgroup의 특징** 3](#_Toc35958153)

[**1-3. Cgroup의 서브시스템** 4](#_Toc35958154)

[**1-4. blkio 서브시스템** 5](#_Toc35958155)

[2. Ubuntu 부팅 USB 만들기 6](#_Toc35958156)

[**2-1. Ubuntu를 설치할 파티션 설정하기** 6](#_Toc35958157)

[**2-2. Ubuntu 부팅 USB 만들기** 10](#_Toc35958158)

[3. Ubuntu 사용법 및 실험 준비 14](#_Toc35958159)

[**3-1. Ubuntu 터미널 사용법** 14](#_Toc35958160)

[**3-2. 루트 계정 전환 및 Linux, Ubuntu 버전 확인** 15](#_Toc35958161)

[**3-3. Linux 주요 명령어 설명** 15](#_Toc35958162)

[4. FIO 툴 설치 방법 16](#_Toc35958163)

[5. Cgroup blkio weight 100, 200, 400, 800 실험 19](#_Toc35958164)

[**5-1. 디스크 추가** 19](#_Toc35958165)

[**1. 디스크 확인** 19](#_Toc35958166)

[**2. 파티션 생성** 20](#_Toc35958167)

[**3. 파티션 포맷** 22](#_Toc35958168)

[**4. 자동 마운트** 23](#_Toc35958169)

[**5-2. Cgroup Block I/O 컨트롤러 사용** 26](#_Toc35958170)

[**1. Block I/O 컨트롤러 실행 조건** 26](#_Toc35958171)

[**2. Cgroup 마운트 및 Block I/O 컨트롤러 사용** 27](#_Toc35958172)

[**3. Cgroup에 프로세스 할당** 28](#_Toc35958173)

[**4. weight 비율 할당** 31](#_Toc35958174)

[**5-3. FIO 툴을 사용해 IOPS 및 BW 측정** 32](#_Toc35958175)

[**1. FIO Job File 생성** 32](#_Toc35958176)

[**2. FIO Job File 명령어 설명** 36](#_Toc35958177)

[**3. FIO 실행 방법** 37](#_Toc35958178)

[**5-4. 실험 결과 및 고찰** 40](#_Toc35958179)

[**1. 실험 결과** 40](#_Toc35958180)

[**2. 고찰** 41](#_Toc35958181)

[6. Cassandra DB를 이용한 Cgroup 성능 측정 42](#_Toc35958182)

[**6-1. Cassandra DB 란?** 42](#_Toc35958183)

[**6-2. Cassandra DB 특징** 42](#_Toc35958184)

[**6-3. Cassandra DB 설치 방법** 43](#_Toc35958185)

[**1. JDK 설치** 43](#_Toc35958186)

[**2. Cassandra DB 설치** 48](#_Toc35958187)

[**3. Cassandra DB 설치 테스트** 51](#_Toc35958188)

[**6-4. Cassandra DB를 이용한 Cgroup 성능 측정** 53](#_Toc35958189)

[**6-5. 실험 결과 및 고찰** 54](#_Toc35958190)

[**1. 실험 결과** 54](#_Toc35958191)

[**2. 고찰** 54](#_Toc35958192)

# **1. Cgroup 정의**

## **1-1. Cgroup이란?**

* 구글이 개발한 시스템 소프트웨어로 **Control group의 약자**이다.
* C언어로 짜여 있는 파일 시스템이다.
* **CPU, Memory, Network, I/O 등 하드웨어 자원 할당을 프로세스 그룹 단위로 제어**하는 리눅스 커널 모듈이다.
* **리소스를 분배 및 제한하기 위해 사용**되는 기술이다.
* 리눅스에서는 **하드웨어와 소프트웨어 사이에 Cgroup 계층을 두어 자원을 관리**한다.
* **하나 이상의 스레드 모음으로 동작**한다.

## **1-2. Cgroup의 특징**

* **직관적 인터페이스**로, **계층적**이며 **상속**의 속성을 갖는다.
* 시스템에 **하나 또는 복수의 다른 Cgroup 계층이 동시에 존재할 수 있으며**, 서로 다른 Cgroup 계층끼리 연결되지 않은 **다중 구조를 가진 프로세스**이다.
* **하나 또는 복수의 장치를 묶어 그룹**을 만들고 **그룹 단위로 자원을 분배 및 관리**한다.
* 개별 그룹은 **시스템에서 설정한 값만큼 하드웨어를 사용**할 수 있다.
* **프로세스가 사용하는 하드웨어의 자원은 각각 속해 있는 Cgroup의 통제를 받음**으로써 **특정 프로세스가 호스트 OS의 자원을 모두 사용하는 일을 막는다.**
* 시스템 관리자는 Cgroup을 사용하여 **CPU 시간, 시스템 메모리, 네트워크 대역폭과 같은 자원을 격리하고 제한**할 수 있으며, **설정한 Cgroup을 모니터링 하거나 우선 순위 지정, 특정 자원으로의 액세스 거부 등과 같은 세밀한 제어가 가능**하다.
* 사전 정의된 Cgroup을 **재부팅 후에도 다시 구성하도록 설정**할 수 있다.
* 프로세스와 스레드를 그룹화하여 제어하고 관리하는 기능만 하며 **실제 내부적으로 자원을 분배하는 일은 해당 기능을 구현하고 있는 서브시스템에서 수행**된다.
* Cgroup의 서브시스템은 [1-3. Cgroup의 서브시스템](#_1-3._Cgroup의_서브시스템)에서 자세하게 다룬다.

## **1-3. Cgroup의 서브시스템**

* CPU 시간 또는 Memory와 같은 **단일 자원**을 말하며, 다른 Cgroup에 다양한 수준의 시스템 자원을 할당할 수 있는 **자원 컨트롤러**이다.
* **각 계층에는 하나 이상의 서브시스템에 연결**된다.
* **단일 서브시스템은 하나의 계층에만 연결**할 수 있다.
* 예를 들어 Blkio 서브시스템은 **하나의 계층에만 연결**할 수 있으며, **두 개의 서로 다른 계층에 연결될 수 없다.**
* **단일 계층에는 하나 이상의 서브시스템이 연결**될 수 있다.
* 예를 들어 CPU, Memory 서브시스템은 **다른 계층에 연결되어 있지 않으면** **단일 계층에 이들을 연결**할 수 있다.
* Cgroup 서브시스템의 종류는 **CPU, CPUset, Memory, Devices, Blkio 등이 있다.**
* 본 실험에서 주로 사용되는 Blkio 서브시스템에 대해 [1-4. blkio 서브시스템](#_1-4._blkio_서브시스템)에서 자세한 내용을 다룬다.

**<서브시스템>**

|  |  |
| --- | --- |
| **CPU** | CPU의 사용량을 제한하고, CPU에 Cgroup 작업 액세스를 제공하기 위한 스케줄러 제공 |
| **CPUacct** | CPU의 자원 사용에 대한 분석 및 통계 |
| **CPUset** | CPU 및 메모리 배치 제어 |
| **Memory** | Cgroup 작업에 사용되는 메모리 사용량 제한 |
| **Devices** | 디바이스에 대한 액세스를 허가 또는 제한 |
| **Freezer** | 그룹에 속한 프로세서를 일시적으로 정지 또는 재개 |
| **Net\_cls** | 특정 Cgroup 작업에서 발생하는 패킷을 식별하기 위한 태그 지정 |
| **Blkio** | 특정 블록 디바이스에 대한 접근을 제한 또는 제어 |

## **1-4. blkio 서브시스템**

* **특정 블록 디바이스에 대한 접근을 제한 또는 제어**하기 위한 서브시스템이다.
* **블록 장치의 I/O에 대한 액세스를 제어 및 모니터링**하며, 이러한 가상 파일에 값을 기록하면 **액세스 또는 대역폭이 제한**된다.
* 가상 파일에서 값을 읽을 때 I/O 작업에 대한 정보가 제공된다.

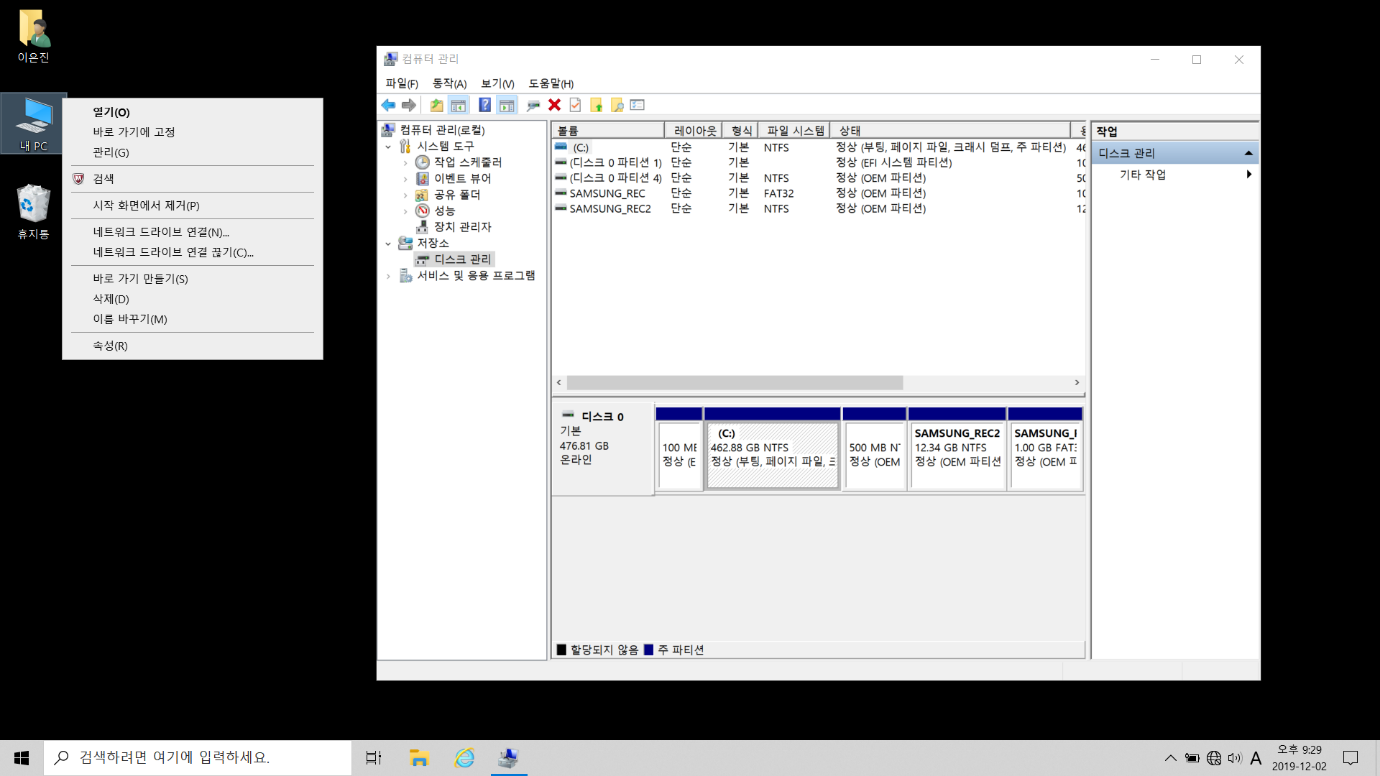
# **2. Ubuntu 부팅 USB 만들기**

## **2-1. Ubuntu를 설치할 파티션 설정하기**

* Ubuntu를 설치하기 위해서 **Ubuntu에 할당할 디스크 용량 크기를 설정** 해야 합니다.
* **디스크 용량이 윈도우에 모두 할당되어 있는 경우** 아래 순서대로 파티션을 나눠주어야 합니다.

1. **디스크 관리 창 열기**

**내 PC -> 우 클릭 -> 관리 클릭 -> 저장소 -> 디스크 관리 클릭**



[그림 1] 디스크 목록 창 화면에 띄우기

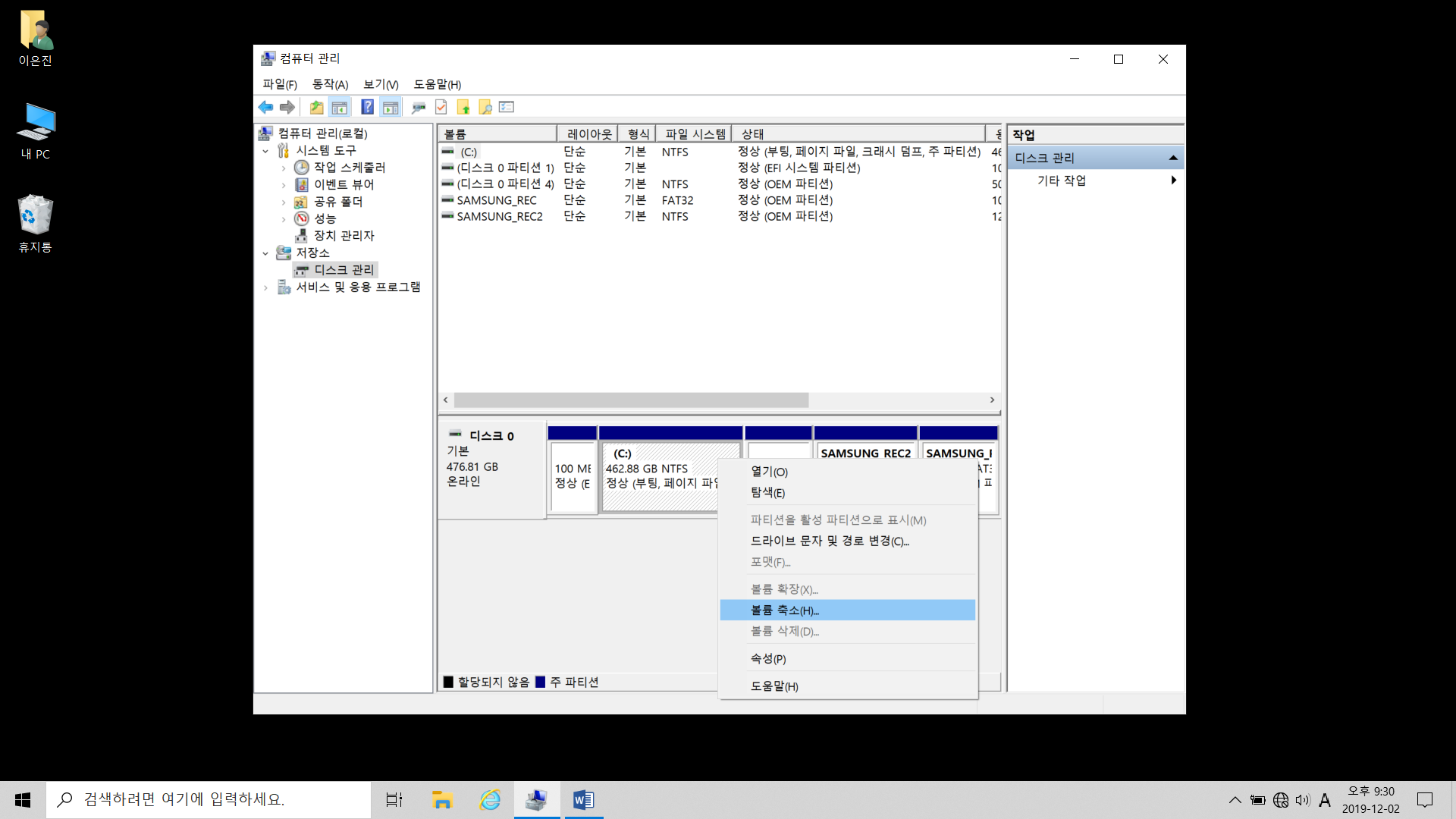
저장소 하위메뉴에 있는 디스크 관리를 누르면 **[그림 1]**과 같이 디스크 목록 창이 뜨게 됩니다.

1. **파티션 나누기**

**[그림 2]**를 보면 **디스크 용량이 윈도우에 모두 할당되어 있는 것을 확인**할 수 있습니다.

C 드라이브에 Ubuntu가 설치될 공간을 마련해주기 위하여 **볼륨 축소**를 눌러줍니다.

**C 드라이브 우 클릭 -> 볼륨 축소 클릭**



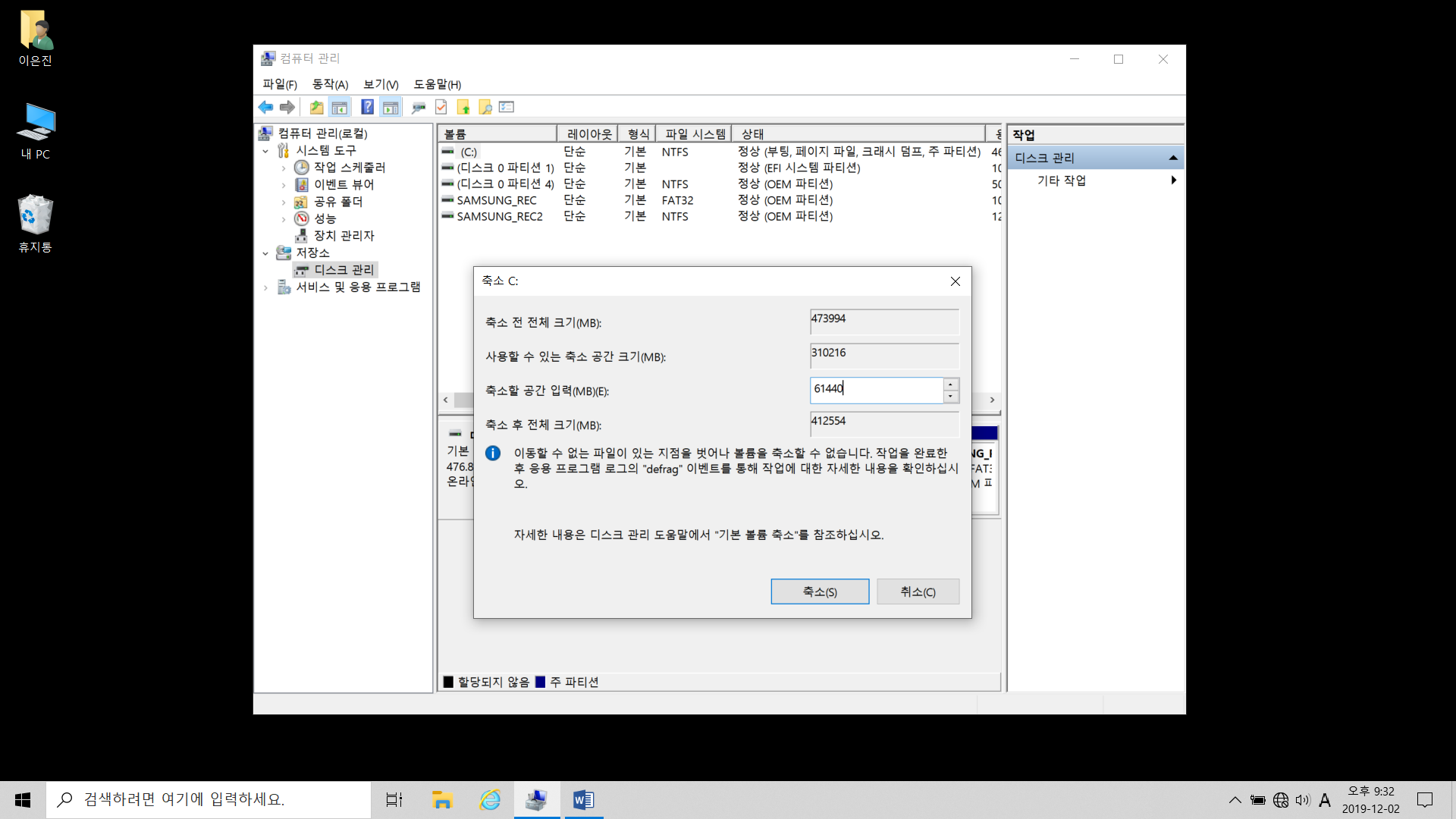
[그림 2] 디스크 용량이 윈도우에 모두 할당되어 있는 것을 확인하고, 볼륨 축소하기

볼륨 축소를 누르면 **[그림 3]**과 같은 창이 열리게 됩니다.

Ubuntu에 할당하기 위하여 60G를 할당 해제해주었습니다.

**단위는 MB로** 예를들어 60G를 할당하려면 MB로 변환한 값 61440MB를 적어줍니다.

**\* Ubuntu에 할당할 용량의 크기는 자유롭게 정하셔도 됩니다.**



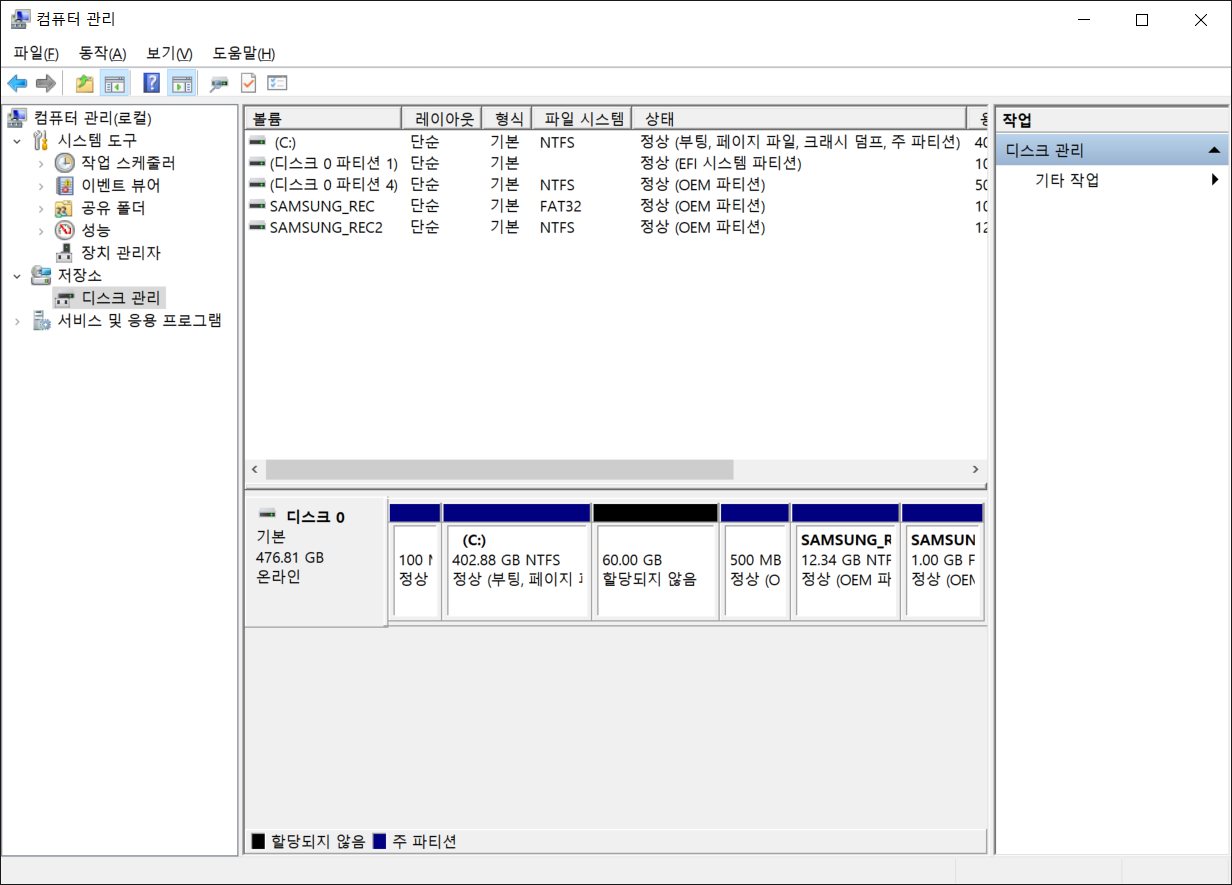
[그림 3] Ubuntu에 용량 할당하기

축소할 공간을 입력했다면 하단에 있는 **축소 버튼**을 눌러줍니다.

축소 버튼을 누르면 **[그림 4]**와 같이 **60GB의 할당되지 않은 디스크 공간**이 생기게 됩니다.

Ubuntu를 설치할 공간이 마련되었습니다.

[그림 4] 60G의 디스크 공간이 할당된 것을 확인했으면 Ubuntu 설치 준비 완료



## **2-2. Ubuntu 부팅 USB 만들기**

**※ USB에 Ubuntu를 설치하게 되면 기존 USB에 저장돼 있던 파일이 전부 포맷 됩니다.**

* **USB에 Ubuntu 설치하기**

**준비물 : 8GB 이상의 USB**

\* Ubuntu를 다운로드하기에 앞서 **준비한 USB를 컴퓨터에 연결**시켜줍니다.

1. **Ubuntu iso 파일을 다운로드 합니다.**

**\* Ubuntu 공식 다운로드 사이트 :** [**http://releases.ubuntu.com/16.04/**](http://releases.ubuntu.com/16.04/)

Ubuntu 공식 다운로드 사이트에 접속하면 **[그림 1]**과 같은 화면이 뜨게 됩니다.

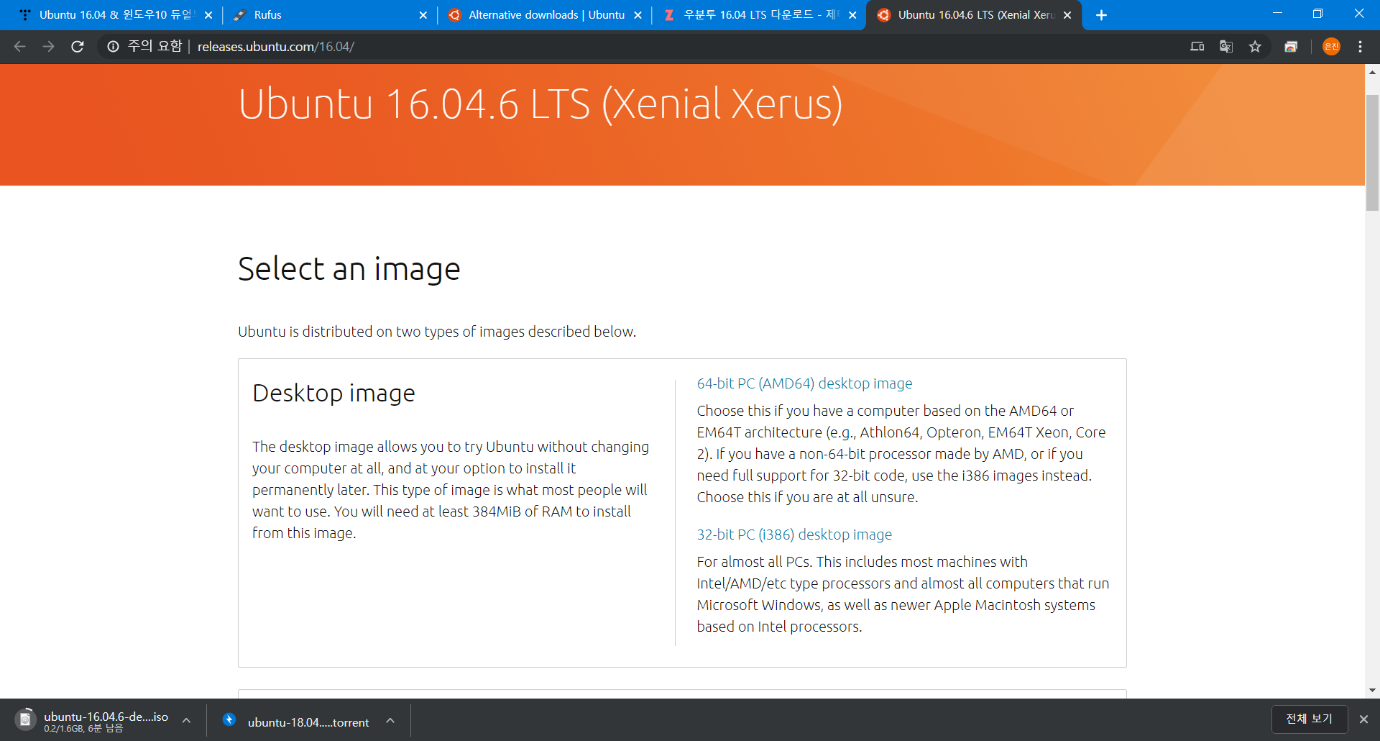
실험을 위해서는 **16.04 LTS 버전**을 다운 받아야합니다.

컴퓨터 시스템 종류를 확인 후 **자신의 컴퓨터에 맞는 버전을 선택**하여 다운로드 합니다.

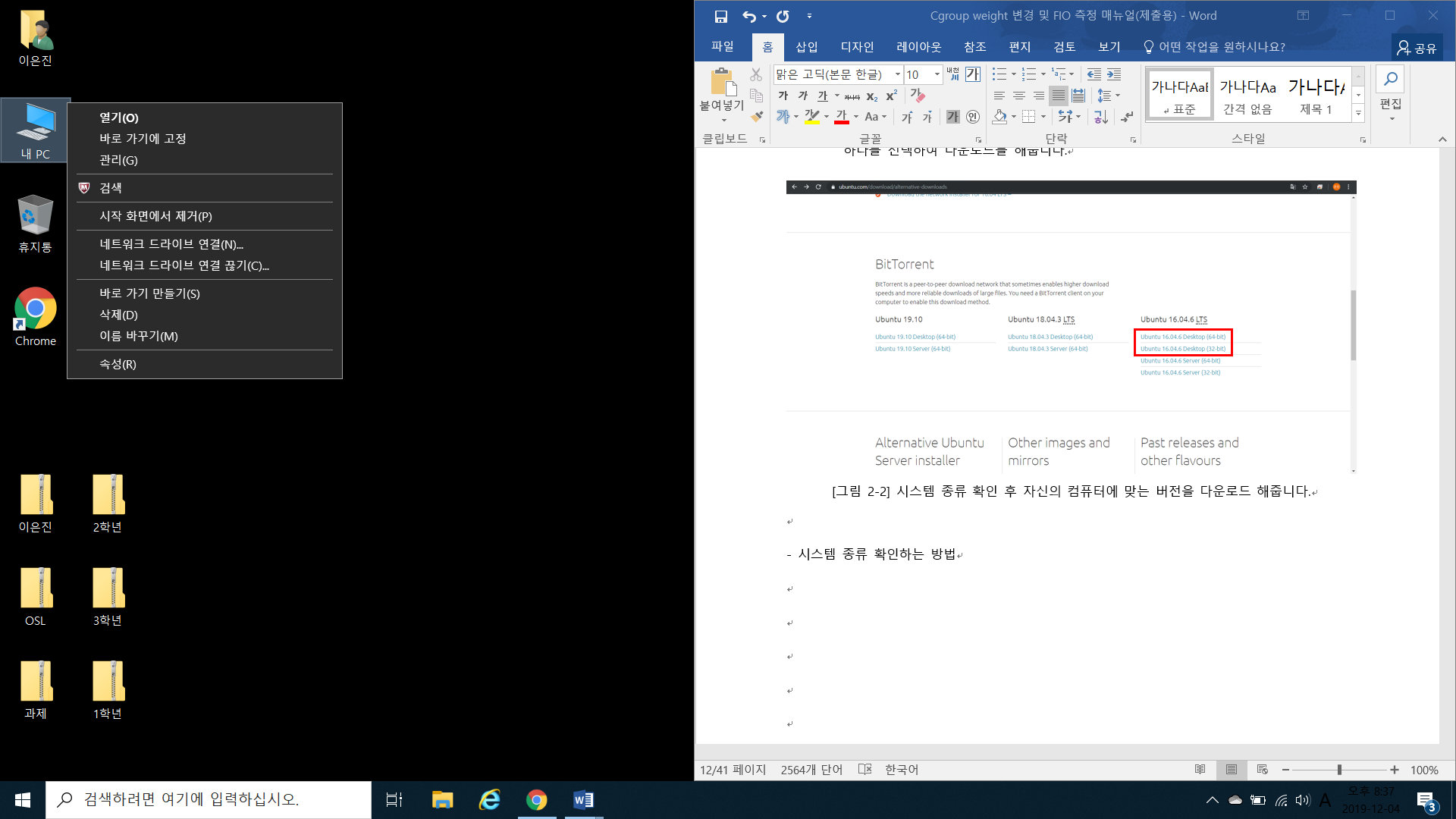
\* **Ubuntu 16.04.6 LTS Desktop (64-bit)** 또는 **Ubuntu 16.04.6 LTS Desktop (32-bit)**

둘 중 **하나를 선택하여 다운로드**를 해줍니다.

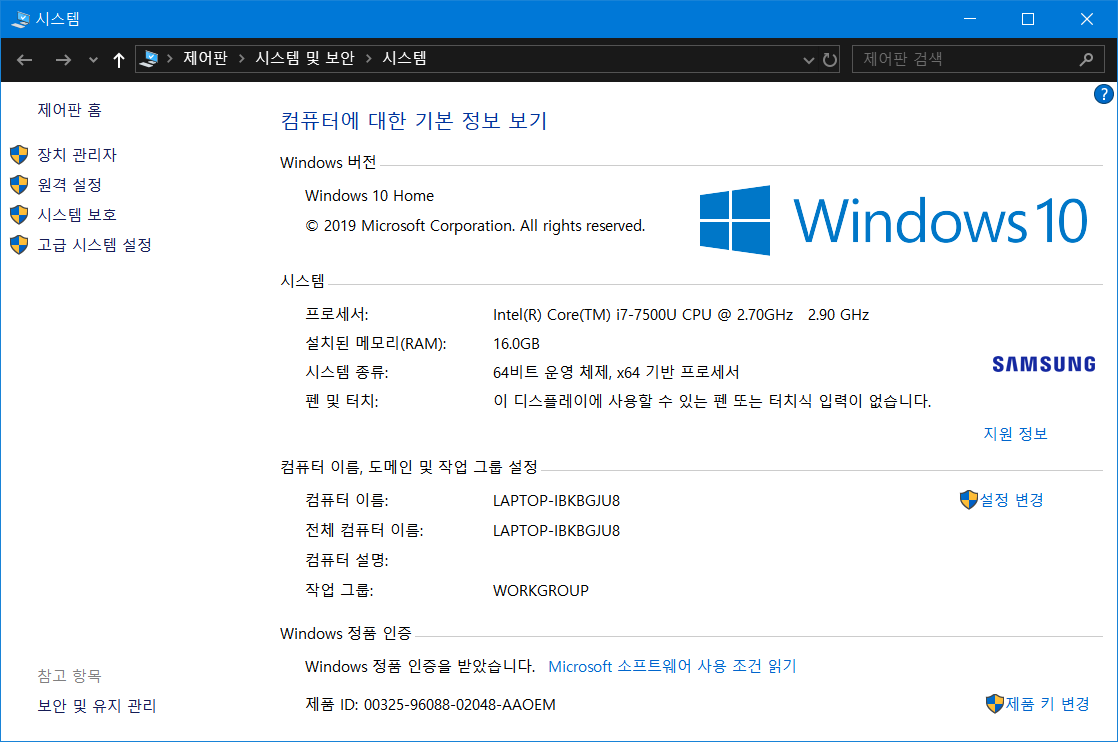
[그림 1] 시스템 종류 확인 후 자신의 컴퓨터에 맞는 버전 다운로드



**<시스템 종류 확인하는 방법>**



내 PC 우 클릭 -> 속성 클릭



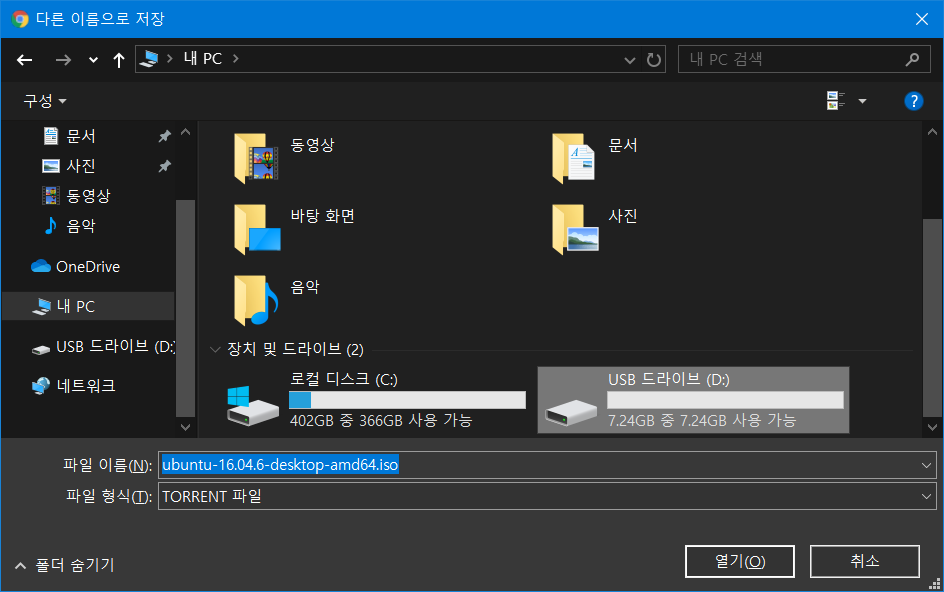
시스템 종류를 확인해보면 64bit인지 32bit인지 알 수 있습니다.

=>

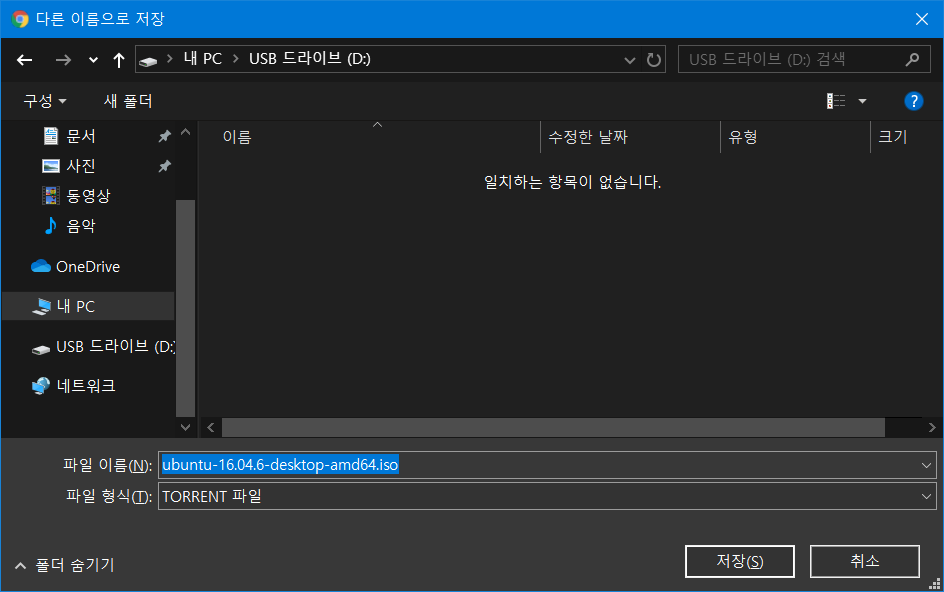
**\* Ubuntu iso 파일의 다운로드 및 압축풀기는 시간이 다소 소요될 수 있습니다.**

다운로드를 누르게 되면 아래와 같은 창이 뜨게 됩니다. Ubuntu를 USB에 다운로드 하기 위하여

**내 PC -> 연결한 USB 드라이브 -> 열기** 순서로 클릭해줍니다.



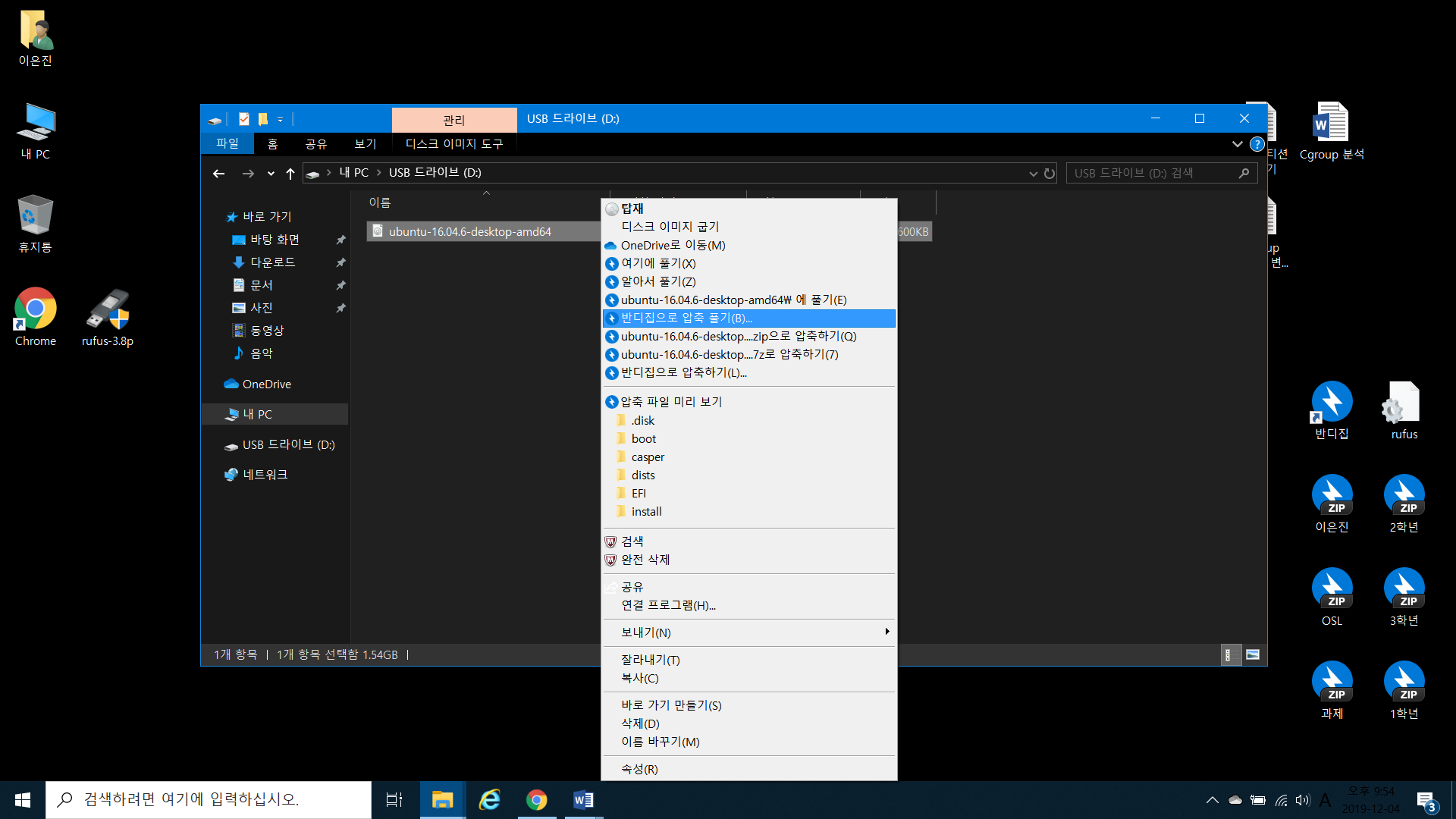
그 다음 **저장을 눌러 Ubuntu를 다운로드** 해줍니다.



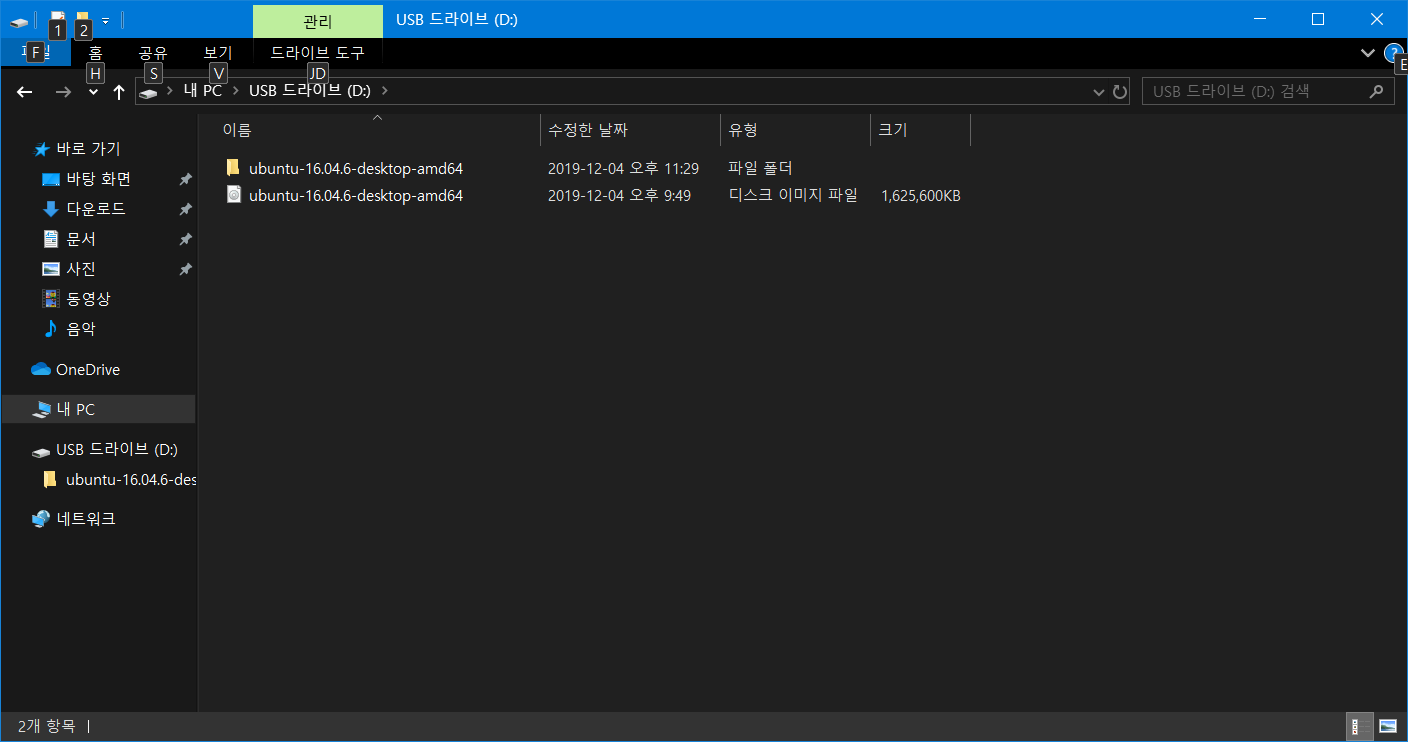
1. **Ubuntu iso 압축 풀기**

**USB에 Ubuntu iso가 정상적으로 설치된 것을 확인**할 수 있습니다.

설치된 **Ubuntu iso의 압축을 풀어줍니다.**



압축을 푼 뒤 **[그림 5]**와 같이 **Ubuntu 파일 폴더가 생성된 것을 확인**할 수 있습니다.



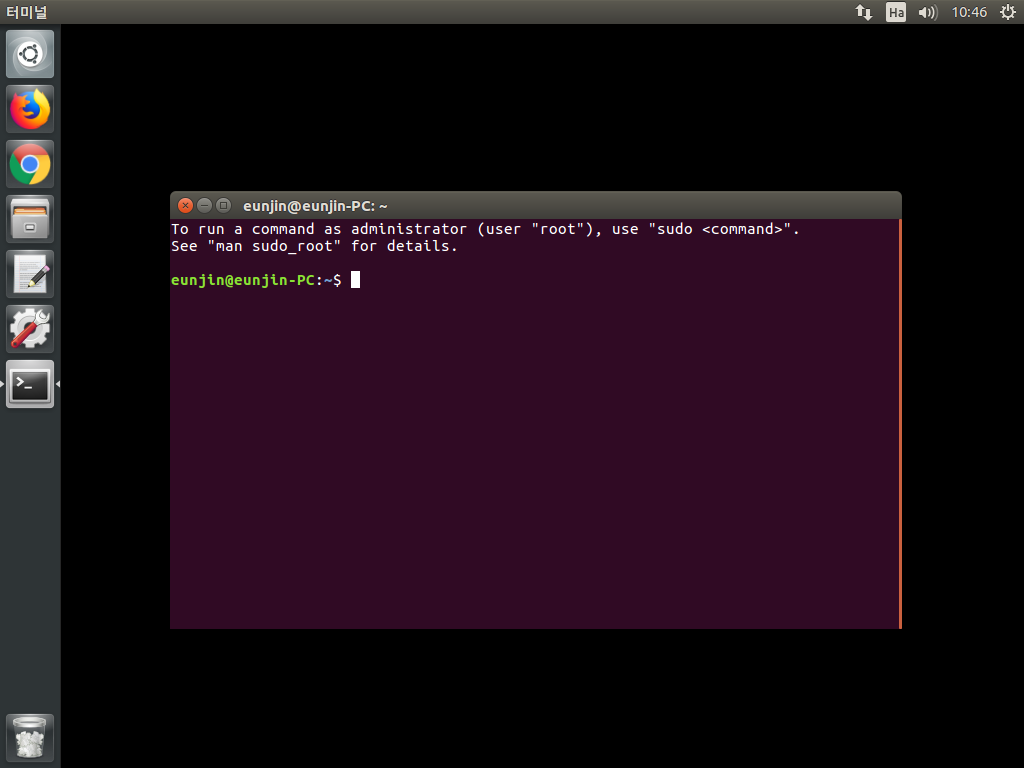
[그림 5] 부팅 USB 만들기 완료

Ubuntu 파일 폴더가 생성된 것을 확인했다면 **부팅 USB 만들기가 완료**되었습니다.

# **3. Ubuntu 사용법 및 실험 준비**

## **3-1. Ubuntu 터미널 사용법**

* **터미널 열기**

**마우스 우클릭 → 터미널 열기 클릭**

[그림 1] 터미널 열기

## **3-2. 루트 계정 전환 및 Linux, Ubuntu 버전 확인**

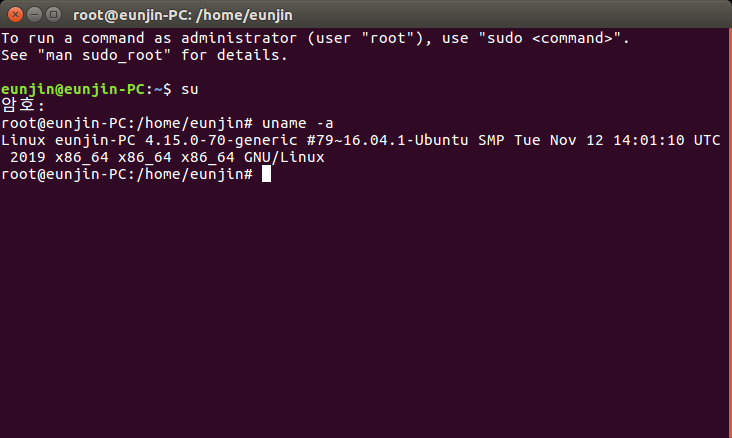
* **루트 계정 전환 (루트(관리자) 계정 = #, 일반 사용자 계정 = $)**

**su 입력 → 암호 입력**

* **일반 사용자 계정($)에서 루트 계정(#) 사용하기**

**sudo 입력 : ex) sudo uname -a**

* **Linux, Ubuntu 버전 확인**

**uname** **-a**

[그림 1] Cgroups 실험에서 CFQ 스케줄러를 사용하기 위해 “Ubuntu 16.04 LTS” 버전 사용

## **3-3. Linux 주요 명령어 설명**

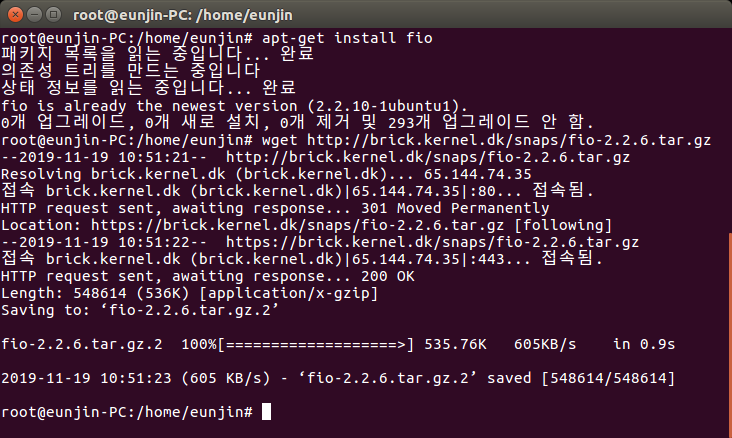
* **cat :** 텍스트로 작성된 파일을 화면에 출력
* **mount :** 파일시스템 구조 내에 있는 일련의 파일들을 사용자가 이용할 수 있도록 만듦
* **mkdir :** 새로운 디렉토리 생성
* **rmdir :** 디렉토리 삭제
* **rm :** 파일 삭제
* **ls :** 디렉토리의 목록 확인
* **echo :** 문자열 화면 출력
* **cd :** 원하는 디렉토리로 이동

# **4. FIO 툴 설치 방법**

**\* FIO-3.7 버전에서 ‘ioengine=libaio’이 실행이 되지 않아 FIO-2.2.6 버전을 사용하였습니다.**

* **I/O 측정을 위한 Fio 툴 설치**

**1) apt-get install fio**

**2) wget http://brick.kernel.dk/snaps/fio-2.2.6.tar.gz**

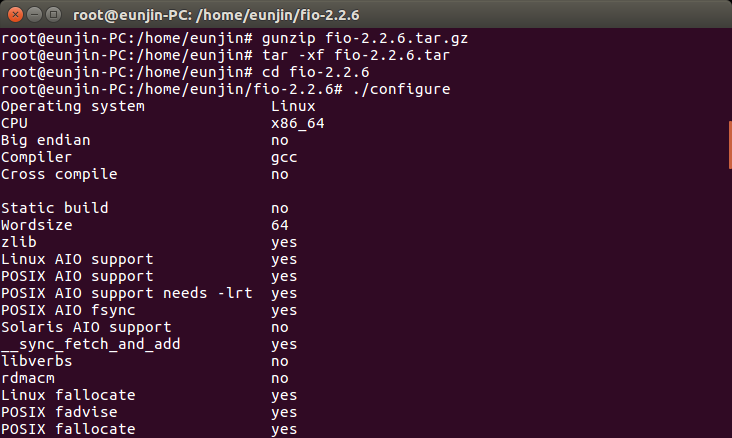
[그림 1] FIO 설치 화면

* **Fio zip 압출 풀기 및 생성**

**3) gunzip fio-2.2.6.tar.gz : 압축을 풀어줍니다.**

**4) tar -xf fio-2.2.6.tar**

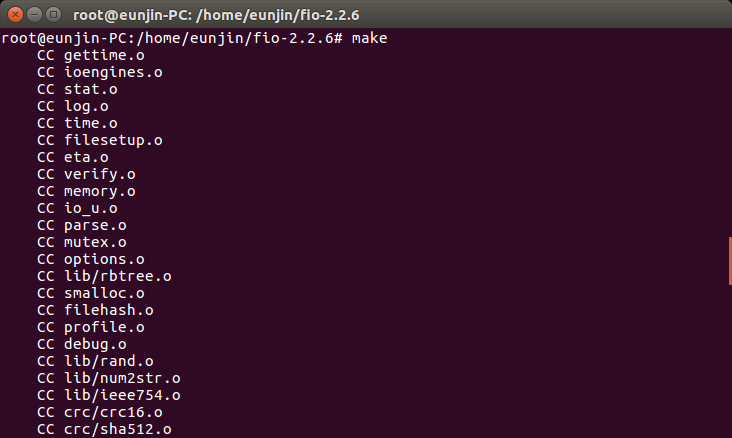
**5) cd fio-2.2.6 : fio-2.2.6 디렉토리로 이동합니다.**

**6) ./configure**

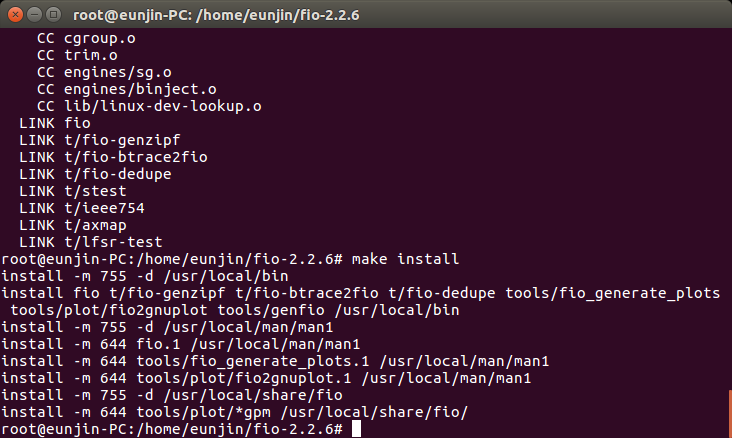
[그림 2] 설치된 FIO의 압축 풀기 및 사용 준비

**7) make**

**8) make install**



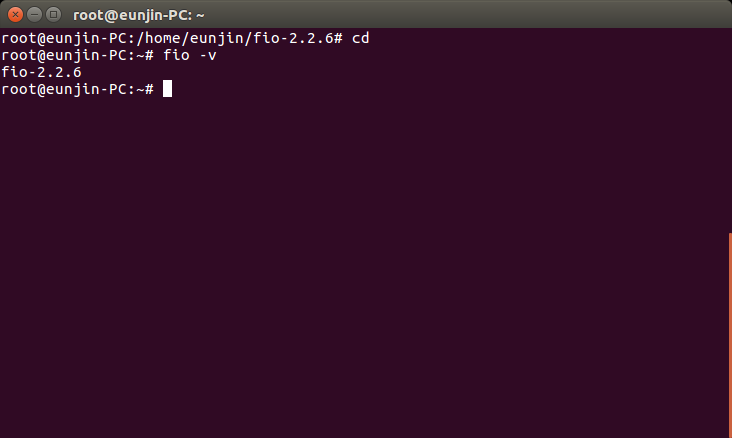
[그림 3] make 및 make install이 완료되었다면, FIO를 사용할 준비 완료



* **fio-2.2.6 디렉토리에서 빠져나오기**

**cd**

* **Fio 버전 확인**

**fio -v**

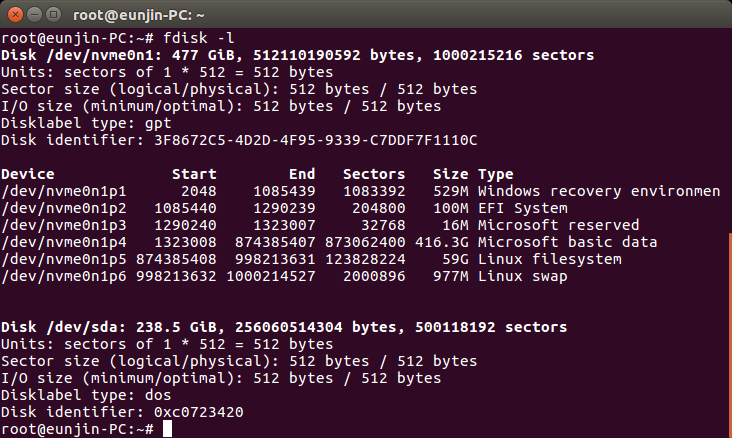
[그림 4] FIO가 정상적으로 설치된 것을 확인할 수 있습니다.

# **5. Cgroup blkio weight 100, 200, 400, 800 실험**

## **5-1. 디스크 추가**

### **1. 디스크 확인**

* **Ubuntu에 연결돼 있는 디스크 목록 확인**

**fdisk -l**

[그림 1] SSD 장착 여부 확인

**fdisk -l** 명령어를 사용하면 **[그림1]**과 같이 **현재 Ubuntu에 연결되어 있는 디스크의 목록**을 확인할 수 있습니다.

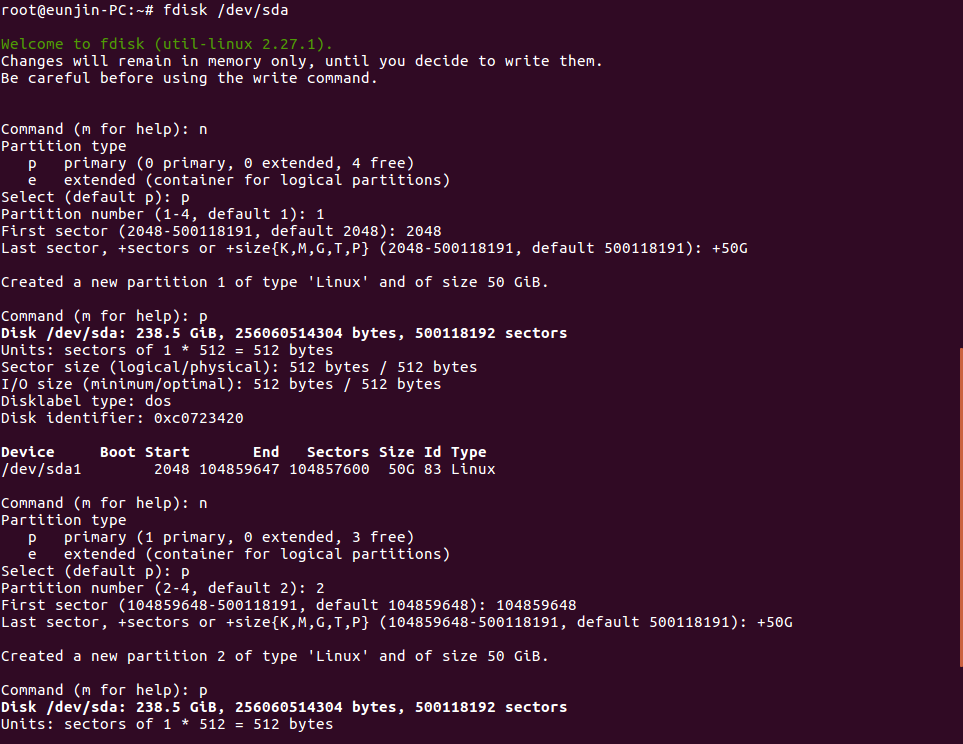
**[그림 1]**에 나타난 **sda는** **장착된 SSD의 이름**입니다.

**\* SSD의 이름은 sda, sdb, sdc 등 장착된 메모리에 따라 다르게 표시될 수 있습니다.**

### **2. 파티션 생성**

**\*** Cgroup blkio의 **weight의 값을 4개로 설정**하기위해 **파티션을 4개**로 나누었습니다.

**fdisk /dev/sda**



[그림 2] 파티션을 4개로 나누는 과정

**새로운 파티션 생성**

**\* Select (default p) : p**

**= 주 파티션 (p)**

**1번 파티션**

**2048부터~**

**50G 할당**

**파티션 설정 확인**

**1) n을 눌러 새로운 파티션을 생성합니다.**

**2) p (primary)를 눌러 주파티션으로 생성합니다.**

**3) 1번 파티션을 선택합니다.**

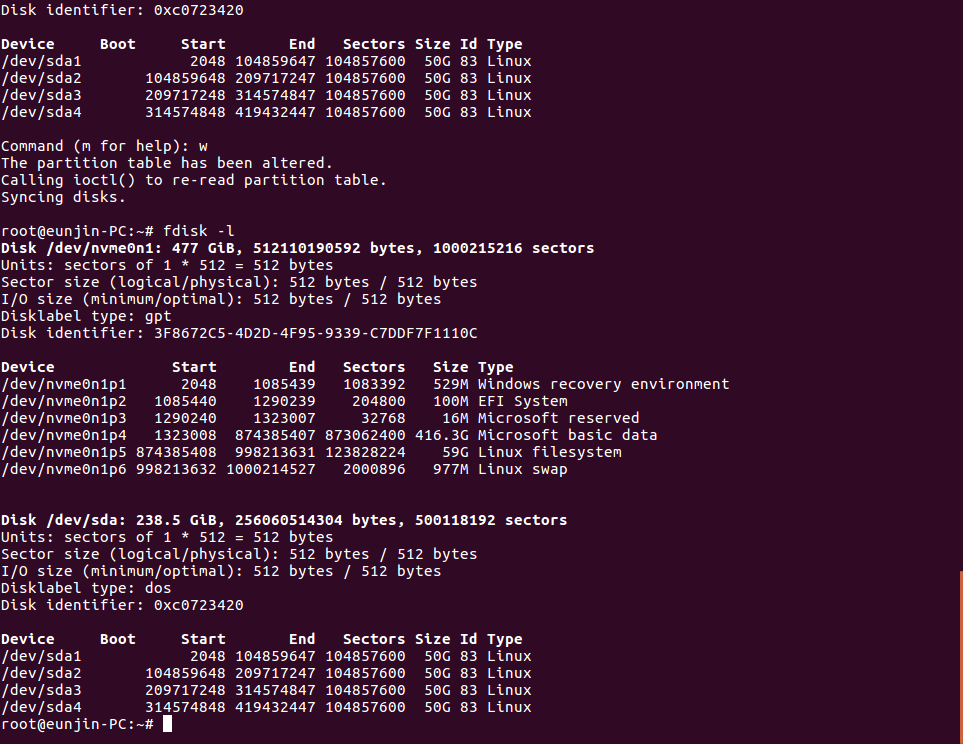
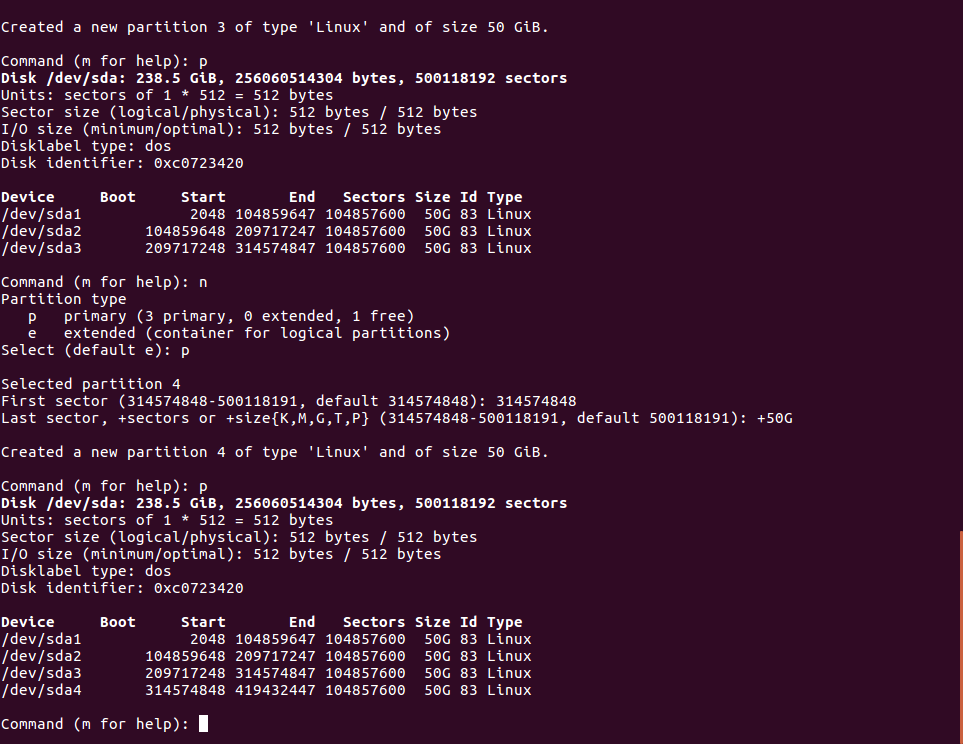
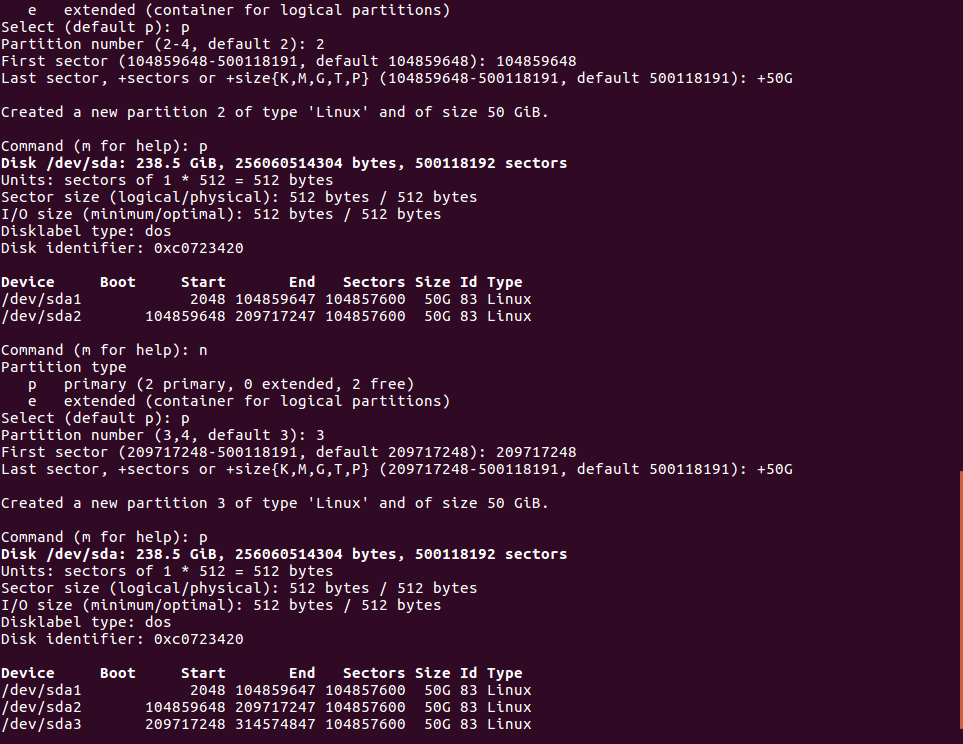
**4) 2048부터 +50G를 해줌으로써 104859647까지 사용하게 됩니다.**

**5) p를 눌러 파티션이 설정됨을 확인합니다.**

**- /dev/sda1 이라는 새로운 파티션이 생성됐음을 알 수 있습니다.**

**6) 위와 같은 방법으로 4번 반복해줍니다.**

[그림 3] 파티션을 4개로 나누는 과정



**저장 후 종료**

**7) [그림 3]과 같이 /dev/sda1~/dev/sda4 총 새로운 4개의 파티션이 생성됐음을 알 수 있습니다.**

### **3. 파티션 포맷**

**\* 파티션 포맷을 진행할 때는** **파티션 나누기가 완료된 상태**이기 때문에 sda가 아닌

**sda1, sda2, sda3, sda4를 입력하여 진행**해야 합니다.

* **새로운 파티션에 file system 포맷**

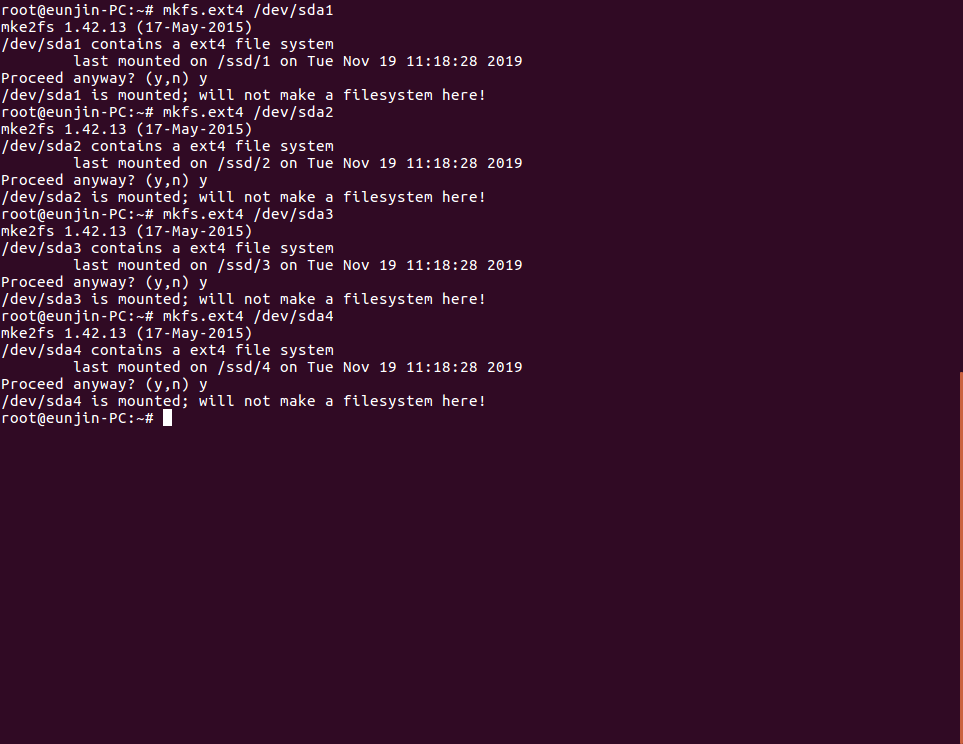
**mkfs.ext4 /dev/sda1 : sda1을 ext4로 포맷해줍니다.**

**mkfs.ext4 /dev/sda2 : sda2를 ext4로 포맷해줍니다.**

**mkfs.ext4 /dev/sda3 : sda3을 ext4로 포맷해줍니다.**

**mkfs.ext4 /dev/sda4 : sda4를 ext4로 포맷해줍니다.**

[그림 4] 새로 생성한 4개의 각 파티션을 ext4로 포맷하는 과정



**\* 새로 생성한 4개의 파티션에 각각 ext4 file system으로 모두 포맷해줍니다.**

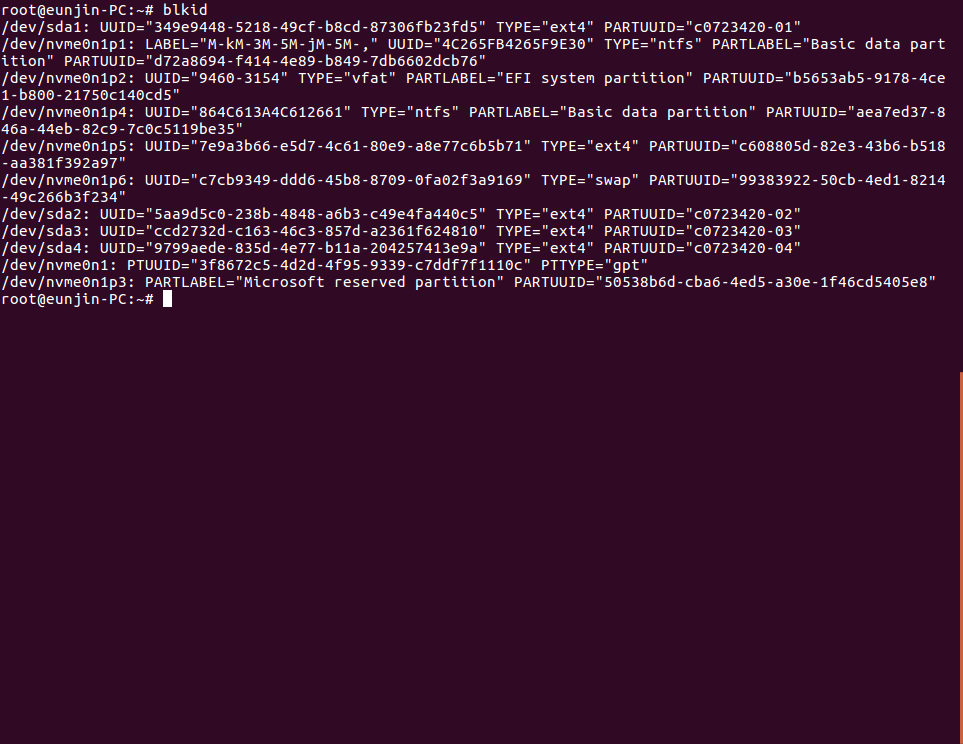
**(**Ubuntu에서 지원하는 **file system은** **bfs, ext2, ext3, ext4, ntfs, fat, vfat** 등이 있지만, **현재 Linux에서 가장 보편적**으로 사용되고 있는 **ext4 file system**으로 포맷해주었습니다.**)**

### **4. 자동 마운트**

* **mount 할 디스크 UUID 및 TYPE 확인**

**blkid**

[그림 5] SSD 자동 마운트 하기



**\*** /dev/sda1~/dev/sda4 4개의 **UUID 및 TYPE을 확인**합니다.

* **생성한 4개의 파티션을 마운트 할 디렉토리를 생성해줍니다.**

**mkdir /ssd/1** **: /dev/sda1 마운트 할 위치생성 → /ssd/1**

**mkdir /ssd/2 : /dev/sda2 마운트 할 위치생성 → /ssd/2**

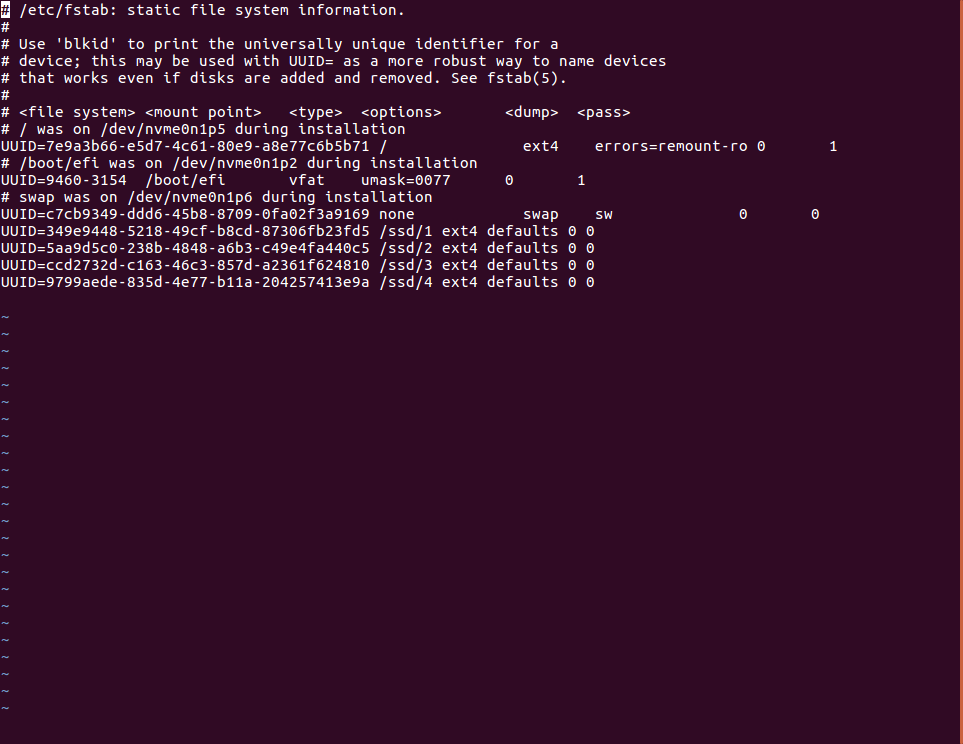
**mkdir /ssd/3 : /dev/sda3 마운트 할 위치생성 → /ssd/3**

**mkdir /ssd/4 : /dev/sda4 마운트 할 위치생성 → /ssd/4**

* **fstab 파일에 파티션 추가**

**vi /etc/fstab** **: 명령어를 입력하면 아래 화면처럼 vi가 바로 실행됩니다.**

[그림 6] 위에서 확인한 UUID 및 TYPE을 형식에 맞춰 적어줍니다.



**<형식>**

**UUID=00000000-0000-0000-000000000000 /ssd/1 ext4 defaults 0 0**

**(각 디스크의 UUID → 마운트 할 디렉토리 → 타입 → 별 다른 사항이 없으면 defaults 0 0)**

**sda1 → /ssd/1 디렉토리에 마운트**

**sda2 → /ssd/2 디렉토리에 마운트**

**sda3 → /ssd/3 디렉토리에 마운트**

**sda4 → /ssd/4 디렉토리에 마운트**

* **vi 사용법**

**맨 아래로 이동** **:** Shift + L

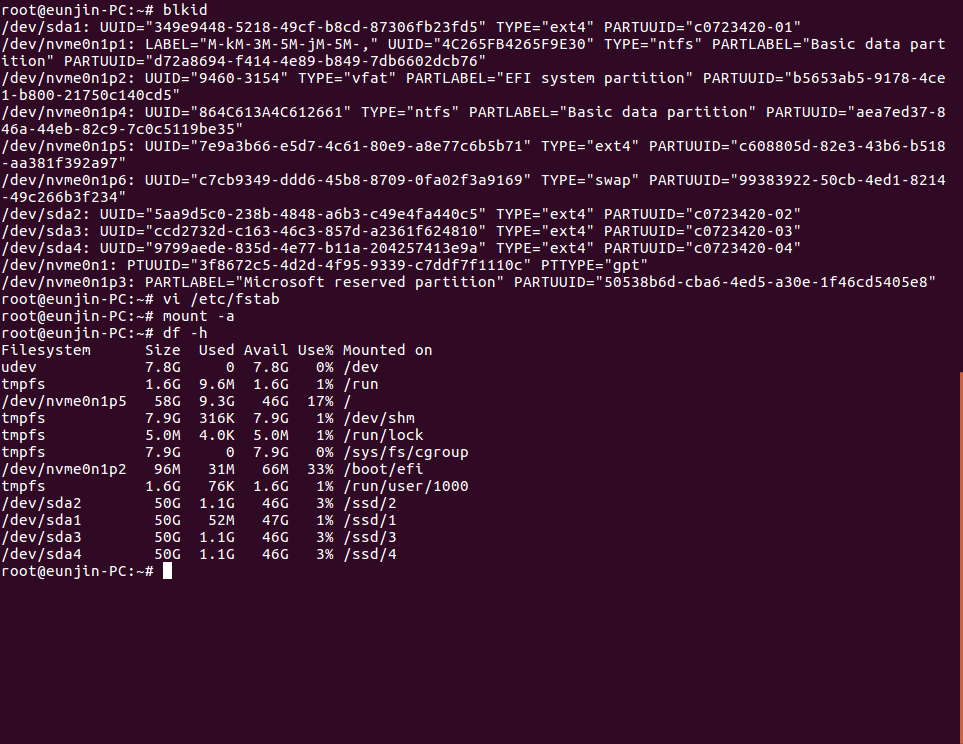
**입력** **:** a

**입력모드 종료** **:** ESC

**저장 후 종료** **:** :wq(콜론wq)

* **마운트 확인**

**mount -a**

**df -h**

[그림 7] 4개의 파티션이 마운트 된 것을 확인할 수 있습니다.

\* /dev/sda1~/dev/sda4 생성한 4개의 파티션이 **/ssd/1~/ssd/4 디렉토리에 마운트 된 것을 확인**할 수 있습니다.

## **5-2. Cgroup Block I/O 컨트롤러 사용**

### **1. Block I/O 컨트롤러 실행 조건**

* **블록 IO 컨트롤러 활성화**

**CONFIG\_BLK\_CGROUP=Y**

* **블록 레이어에서 조절 활성화**

**CONFIG\_BLK\_DEV\_THROTTLING=Y**

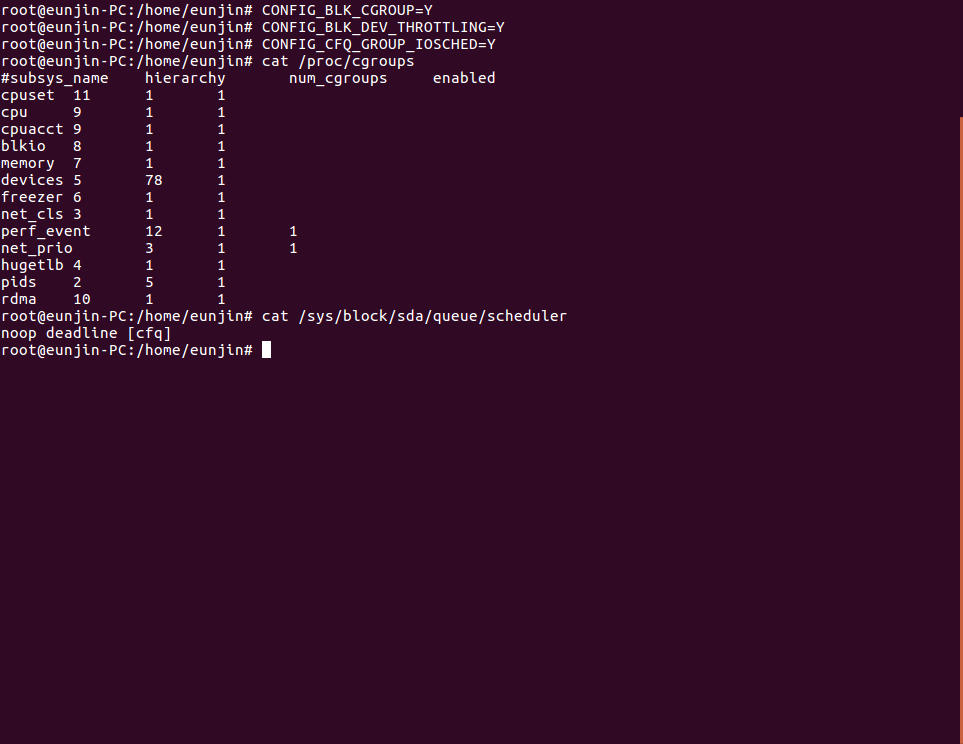
* **CFQ에서 그룹 스케줄링 사용 활성화**

**CONFIG\_CFQ\_GROUP\_LOSCHED=Y**

* **커널이 Cgroup을 지원하는지 확인 (blkio의 enabled가 1로 되어 있으면 완료)**

**cat /proc/cgroups**

* **스케줄러 확인 (CFQ로 되어 있으면 완료)**

**cat /sys/block/sda/queue/scheduler**

[그림 1] Cgroup의 서브시스템인 blklio를 사용하기 위한 필수 준비 단계입니다.

**\* 스케줄러가 noop 또는 deadline으로 돼있을 경우 CFQ로 바꿔줍니다.**

**echo cfq > /sys/block/sda/queue/scheduler**

### **2. Cgroup 마운트 및 Block I/O 컨트롤러 사용**

* **Cgroup 루트 디렉토리와 blkio 하위 시스템 마운트**

**mount -t tmpfs cgroup\_root /sys/fs/cgroup**

**mkdir /sys/fs/cgroup/blkio**

**mount -t cgroup -o blkio none /sys/fs/cgroup/blkio**

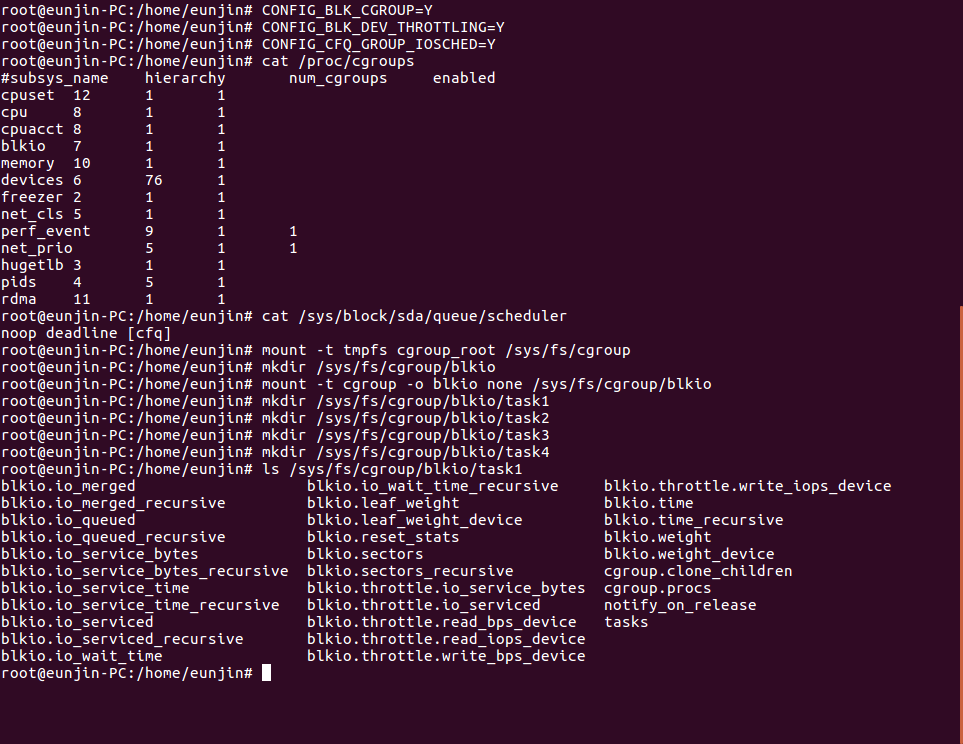
**\* -o 옵션으로 subsystem의 이름을 쓰면 특정 Cgroup과 subsystem이 연결됩니다.**

* **blkio 컨트롤러에서 4개의 Cgroup 생성**

**mkdir /sys/fs/cgroup/blkio/task1** : task1 그룹을 생성해줍니다.

**mkdir /sys/fs/cgroup/blkio/task2** : task2 그룹을 생성해줍니다.

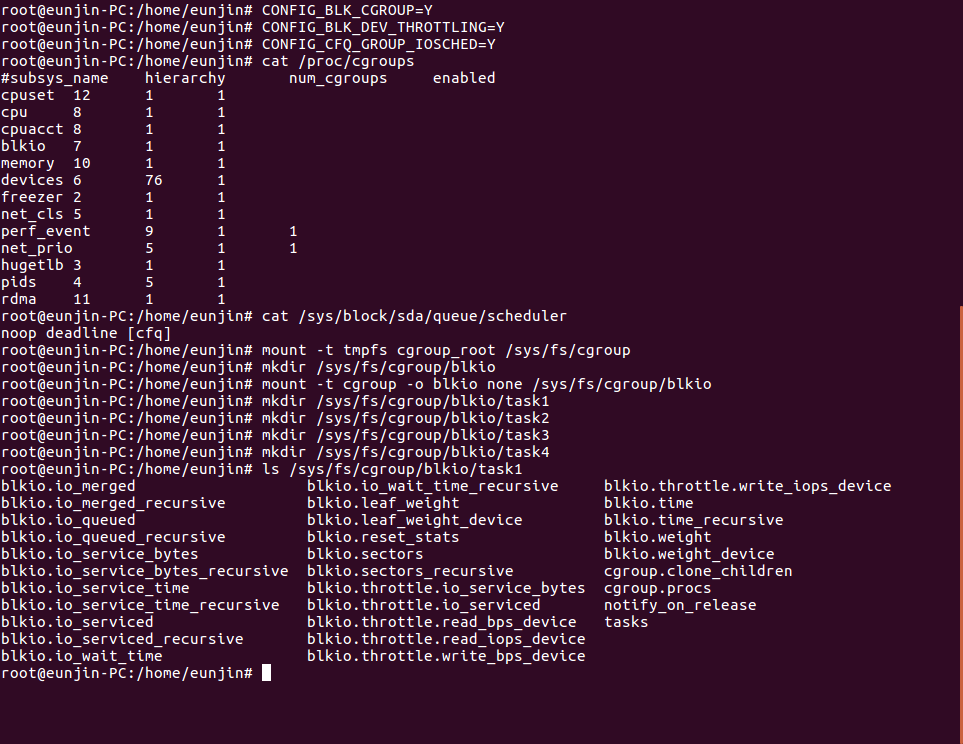
**mkdir /sys/fs/cgroup/blkio/task3** : task3 그룹을 생성해줍니다.

**mkdir /sys/fs/cgroup/blkio/task4** : task4 그룹을 생성해줍니다.

[그림 2] 에러가 뜨지 않으면 정상적으로 마운트 및 생성이 완료된 것입니다.

* **Cgroup task1 디렉토리의 특수파일**

- Cgroup은 **디렉토리 생성과 동시에 특수 파일을 배치**합니다.

** ls /sys/fs/cgroup/blkio/task1** : task1 디렉토리 목록을 확인할 수 있습니다.

[그림 3] task1 디렉토리에 아무것도 포함시키지 않았지만, 이미 많은 특수 파일이 생성되어 있는 것을 확인할 수 있습니다.

### **3. Cgroup에 프로세스 할당**

* **프로세스 목록 확인[[1]](#footnote-1)**

**cat /sys/fs/cgroup/blkio/tasks**

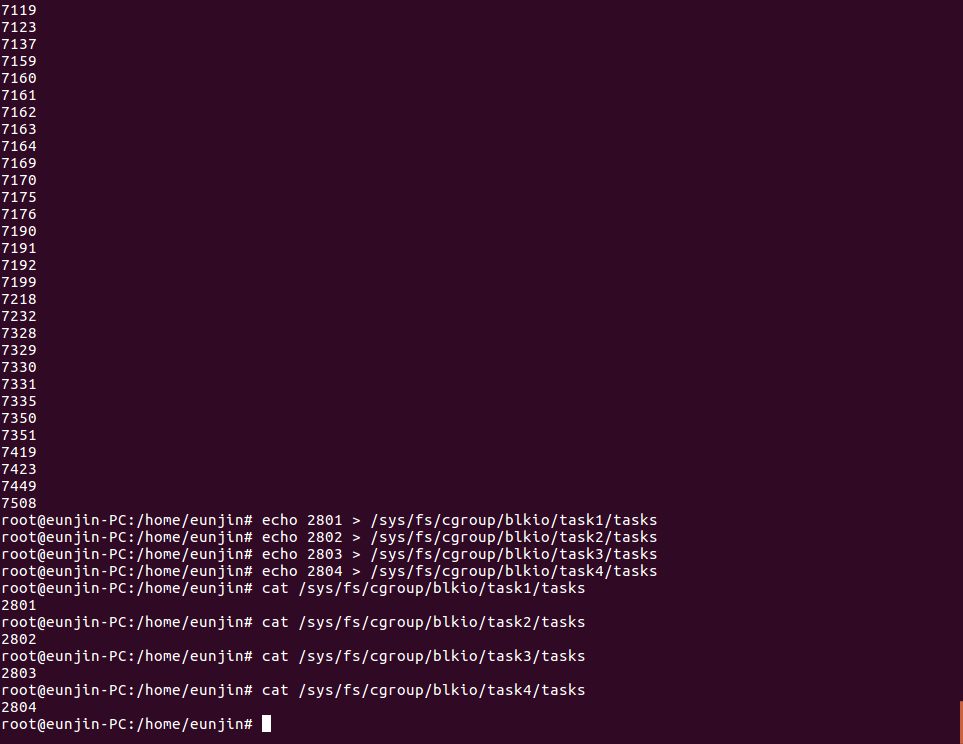
[그림 4] 위에서 생성한 4개의 task 그룹에 TID를 할당해주기 위한 Cgroup TID 확인 단계입니다.

* **tesk1~4 그룹에 특정 프로세스 할당**

**echo 2801 > /sys/fs/cgroup/blkio/task1/tasks** **: task1에 2801 TID 할당**

**echo 2802 > /sys/fs/cgroup/blkio/task2/tasks** **: task2에 2802 TID 할당**

**echo 2803 > /sys/fs/cgroup/blkio/task3/tasks** **: task3에 2803 TID 할당**

**echo 2804 > /sys/fs/cgroup/blkio/task4/tasks** **: task4에 2804 TID 할당**

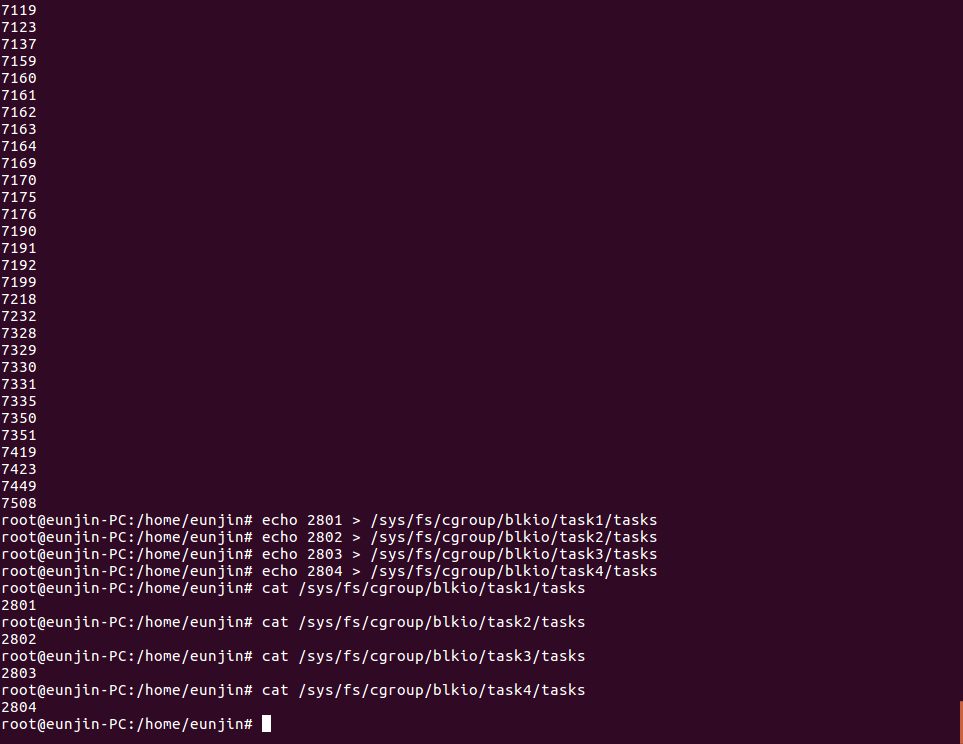
[그림 5] 위에서 확인한 Cgroup TID 목록에서 임의의 4개의 TID를 task1~4 그룹에 할당시켜줍니다.

* **tesk1~4 그룹에 지정한 프로세스가 할당됐는지 확인**

**cat /sys/fs/cgroup/blkio/task1/tasks**

**cat /sys/fs/cgroup/blkio/task2/tasks**

**cat /sys/fs/cgroup/blkio/task3/tasks**

**cat /sys/fs/cgroup/blkio/task4/tasks**

[그림 6] 4개의 task1~4 그룹에 지정해준 프로세스가 할당되어 있는 것을 확인할 수 있습니다.

* **프로세스 목록 확인**

**cat /sys/fs/cgroup/blkio/tasks**

[그림 7] Cgroup TID 목록에 2801~2804가 없는 것을 확인할 수 있습니다.

### **4. weight 비율 할당**

* **task1~4 그룹에 weight 비율 할당**

**echo 100 > /sys/fs/cgroup/blkio/task1/blkio.weight**

**echo 200 > /sys/fs/cgroup/blkio/task2/blkio.weight**

**echo 400 > /sys/fs/cgroup/blkio/task3/blkio.weight**

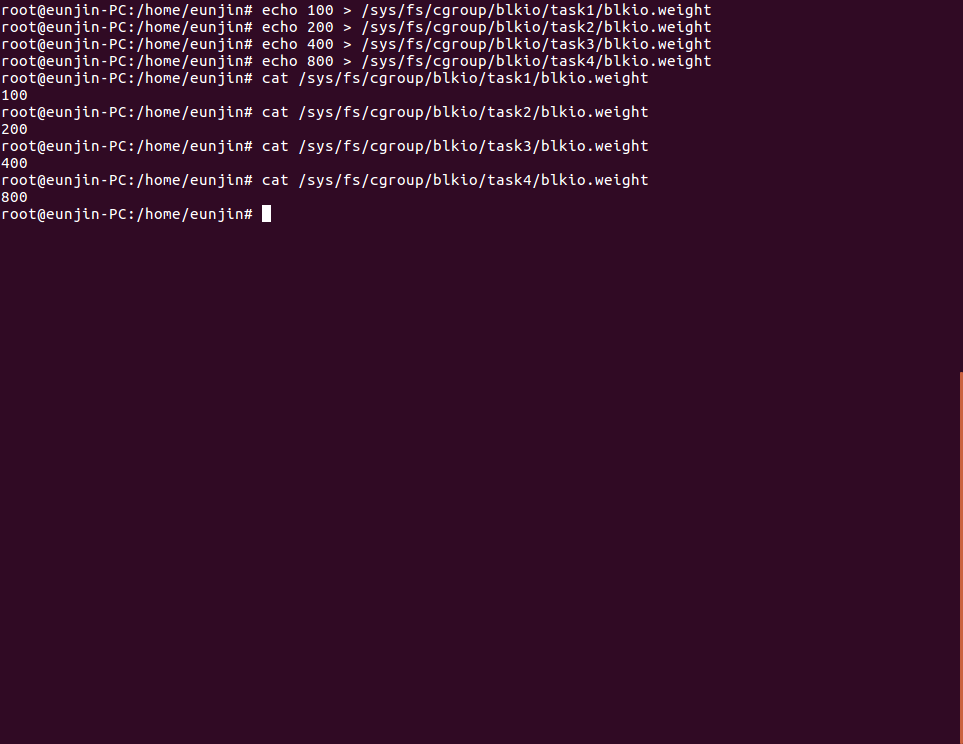
**echo 800 > /sys/fs/cgroup/blkio/task4/blkio.weight**

* **tesk1~4 그룹에 weight 비율이 할당됐는지 확인**

**cat /sys/fs/cgroup/blkio/task1/blkio.weight**

**cat /sys/fs/cgroup/blkio/task2/blkio.weight**

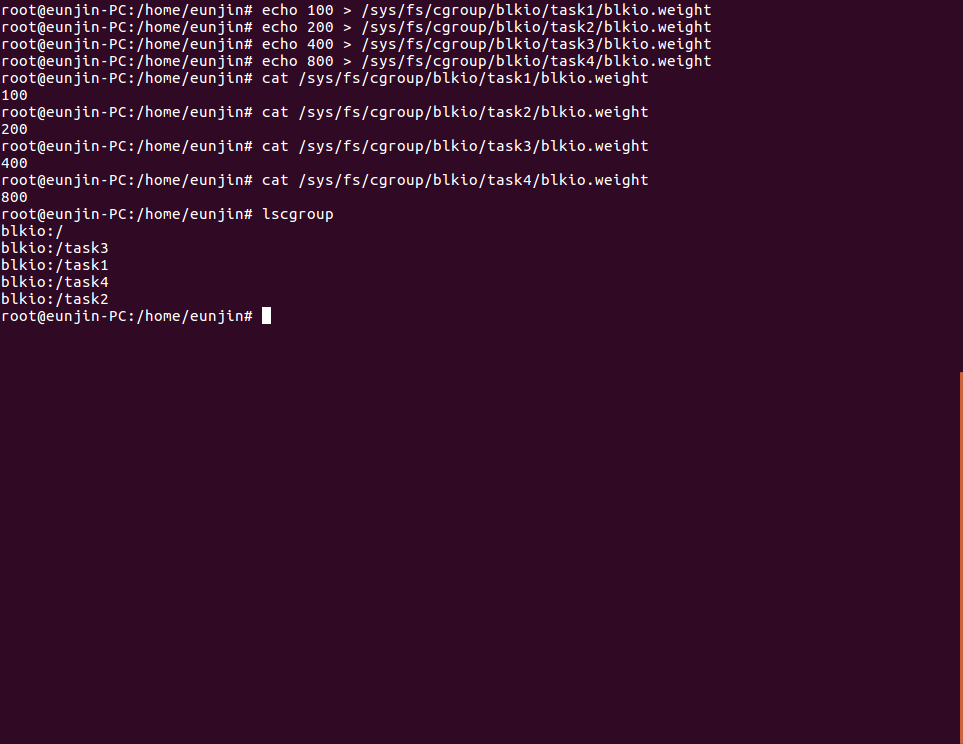
**cat /sys/fs/cgroup/blkio/task3/blkio.weight**

**cat /sys/fs/cgroup/blkio/task4/blkio.weight**

[그림 8] 4개의 task 그룹에 각각 weight 비율을 할당하고, 확인합니다.

**\* task1~4 그룹에 각각 weight 100, 200, 400, 800을 할당해주었습니다.**

* **task1~4 그룹과 blkio 서브시스템 마운트 확인**

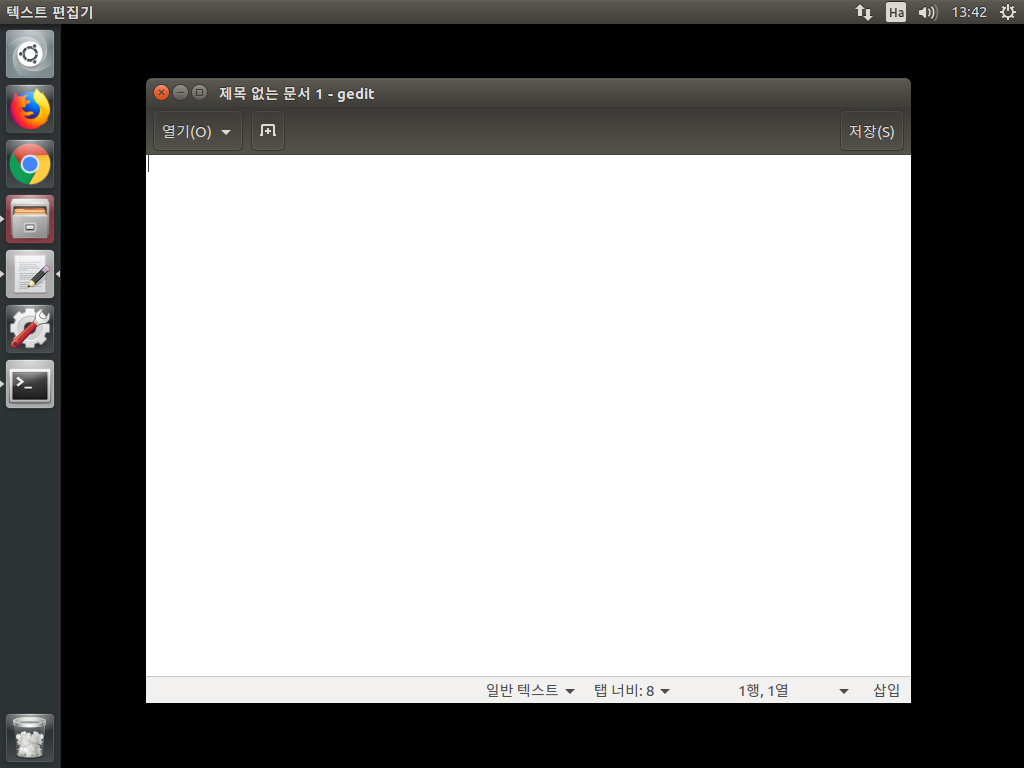
**lscgroup** : task1~4 그룹이 blkio에 마운트 되어 있는지 확인합니다.

[그림 9] 4개의 task 그룹이 Cgroup 서브시스템인 blkio에 마운트 되어 있는 것을 확인할 수 있습니다.

## **5-3. FIO 툴을 사용해 IOPS 및 BW 측정**

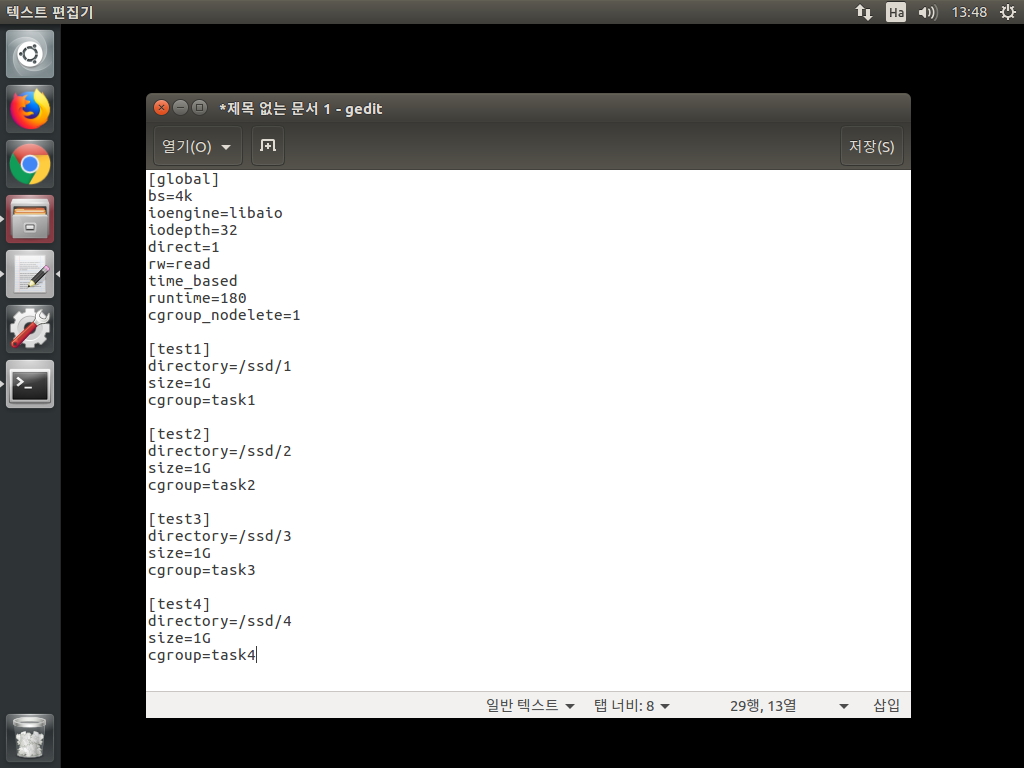
### **1. FIO Job File 생성**

* **텍스트 편집기 열기**

**내 컴퓨터에서 검색 → 텍스트 편집기 검색 → 클릭**

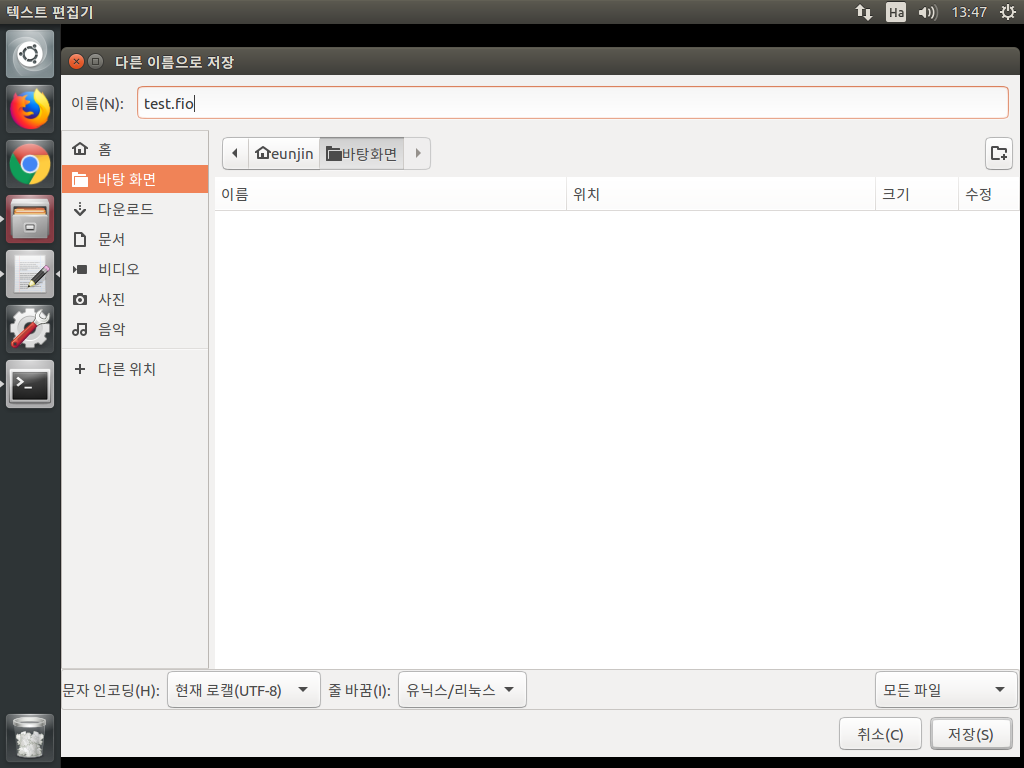
[그림 1] 빈 텍스트 파일이 열린 것을 확인할 수 있습니다.

* **Job File 작성**

\* 내용 및 설명은 [7-2. FIO Job File 명령어 설명](#_7-2._FIO_Job)에서 다뤄집니다.

[그림 2] 실행할 Job File 옵션을 정의합니다.

* **Job File 저장**

**오른쪽 위에 있는 저장 버튼 클릭 → 이름 입력 단, 확장자는 .fio로 해주기 → 저장할 위치는 바탕화면으로 해주기 → 오른쪽 아래 있는 저장 버튼 클릭**

[그림 3] Job File 바탕화면에 저장하기

* **바탕화면에 파일이 저장되어 있으면 완료**

[그림 4] test 파일이 정상적으로 저장된 것을 확인할 수 있습니다.

### **2. FIO Job File 명령어 설명**

* **본 실험에서의 Job File 내용**

|  |  |
| --- | --- |
| [global]  bs=4k  ioengine=libaio  iodepth=32  direct=1  rw=read  time\_based  runtime=180  cgroup\_nodelete=1  [test1]  directory=/ssd/1  size=1G  cgroup=task1 | [test2]  directory=/ssd/2  size=1G  cgroup=task2  [test3]  directory=/ssd/3  size=1G  cgroup=task3  [test4]  directory=/ssd/4  size=1G  cgroup=task4 |

**[global] :** **약속된 이름 (변경 불가능)**

**(global에 정의되어 있는 옵션들은 공통 옵션)**

**bs=4k : 테스트할 블록의 크기**

**ioengine=libaio : 프로세스가 I/O를 발생하는 방법 지정**

**(libaio : Linux AIO)**

**iodepth=32 : AIO를 사용할 때 동시에 발생할 I/O 수의 상한선**

**(대량의 I/O가 발생해 경합하는 것을 방지하기 위함)**

**direct=1 : Direct I/O 사용**

**rw=read : 테스트 종류**

**read : 순차읽기, write : 순차 쓰기**

**randread : 랜덤 읽기, randwrite : 랜덤 쓰기**

**time\_based : runtime 시간 만큼 I/O를 실행 (파일 사이즈 필요 X)**

**runtime=180 : 테스트 진행 시간 (단위 : 초)**

**cgroup\_nodelete=1 : Fio 작업 완료 후 cgroup을 남겨두기 위함 (기본값 false(cgroup 삭제))**

**[test1] : 테스트 이름 (변경 가능)**

**directory=/ssd/1 : 디렉토리 경로(SSD 마운트 한 디렉토리)**

**size=1G : Job 당 총 파일 사이즈**

**(Job이 완료하기가지 I/O를 수행할 데이터양)**

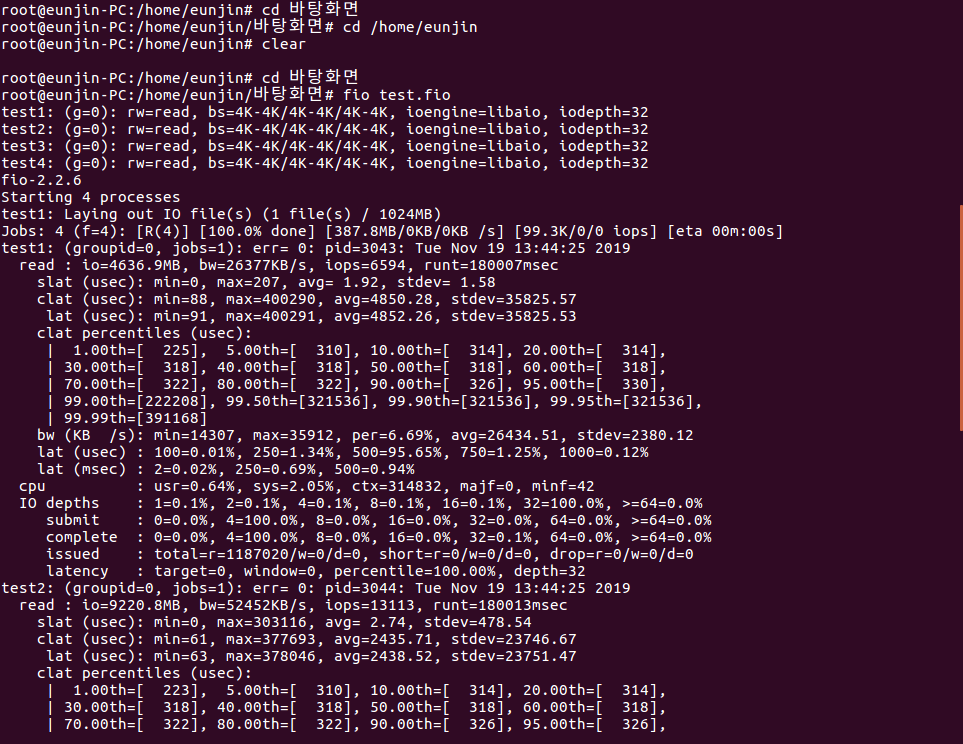
**cgroup=task1 : cgroup과 task1 파일 연결**

### **3. FIO 실행 방법**

* **Fio job file을 저장한 바탕화면으로 이동**

**cd 바탕화면**

* **Fio 실행**

**fio test.fio**

[그림 5] 위에서 생성한 test Job File을 저장한 위치로 이동한 뒤 실행해줍니다.

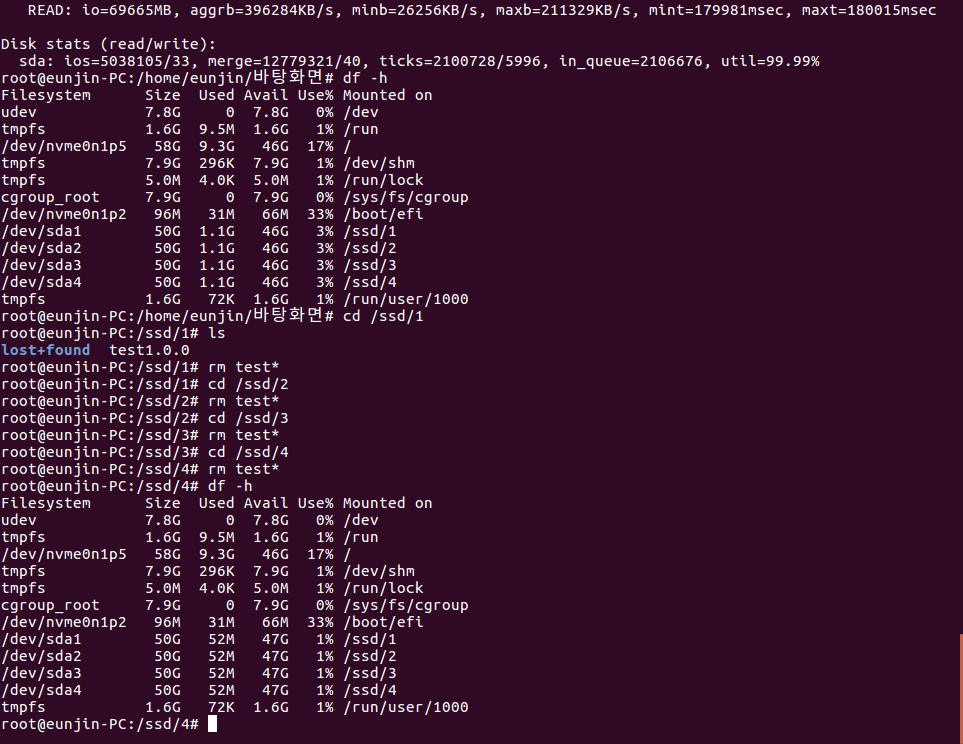
\* fio test.fio 를 실행하면 **[그림 5]**와 같은 실행결과 화면이 나타납니다.

**본 실험에서는 IOPS, BW(KB/s)를 기록하며 실험을 진행**하였습니다.

**[그림 5]**의 경우 **[test1] 즉, task1 그룹의 IOPS = 6594, BW = 26377KB/s**

**[test2] 즉, task2 그룹의 IOPS = 13113, BW = 52452KB/s** 입니다.

* **디스크 마운트 확인 (용량 확인)**

**df -h**

[그림 6] FIO를 실행함으로써 디스크가 사용되었음을 확인할 수 있습니다.

* **test 파일 삭제**

**cd /ssd/1** **: /ssd/1 디렉토리로 이동**

**ls : /ssd/1 디렉토리에 있는 목록 확인**

**rm test\* : test로 시작하는 모든 파일 삭제**

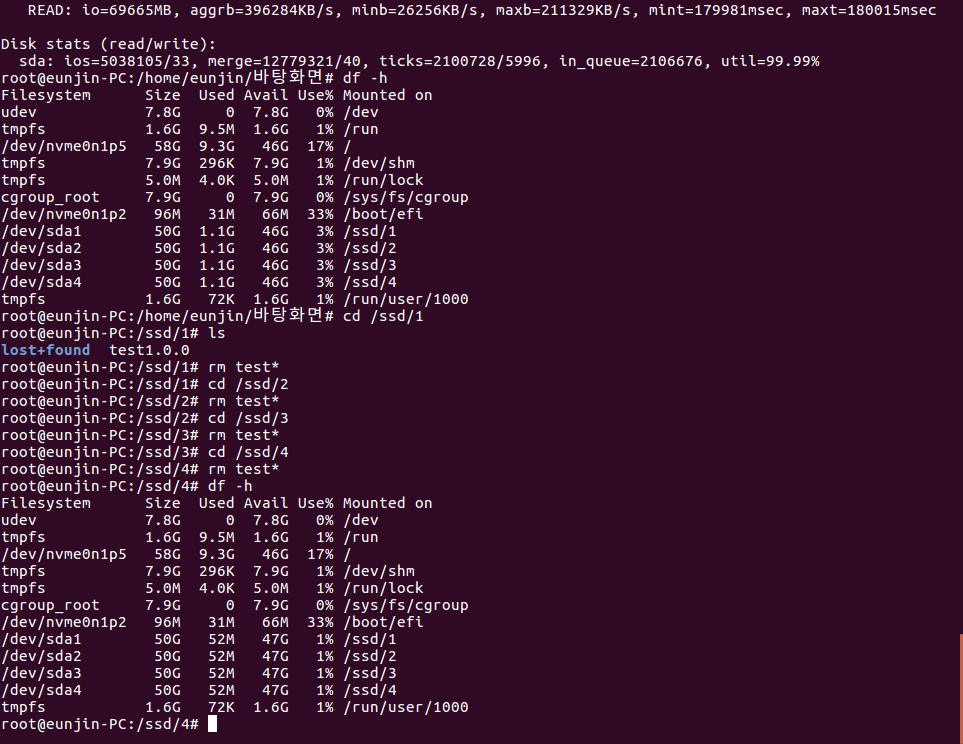
**cd /ssd/2**

**rm test\***

**cd /ssd/3**

**rm test\***

**cd /ssd/4**

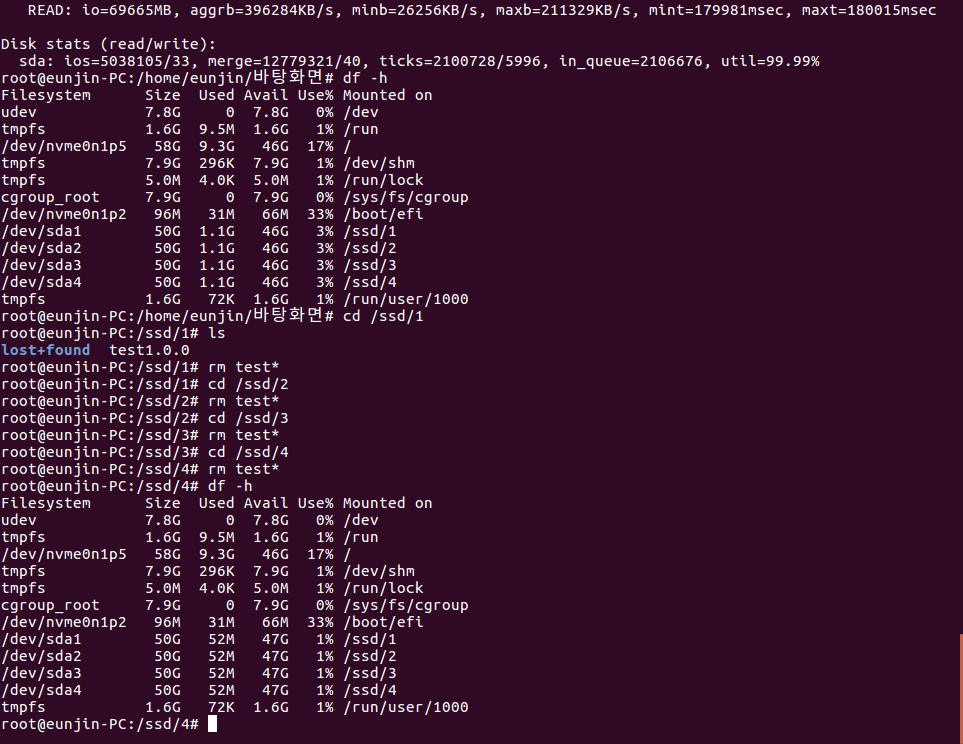
**rm test\***

[그림 7] /ssd/1~/ssd/4 각각의 디렉토리에 생성되어 있는 test을 전부 삭제해줍니다.

* **디스크 마운트 확인 (용량 확인)**

**df -h**

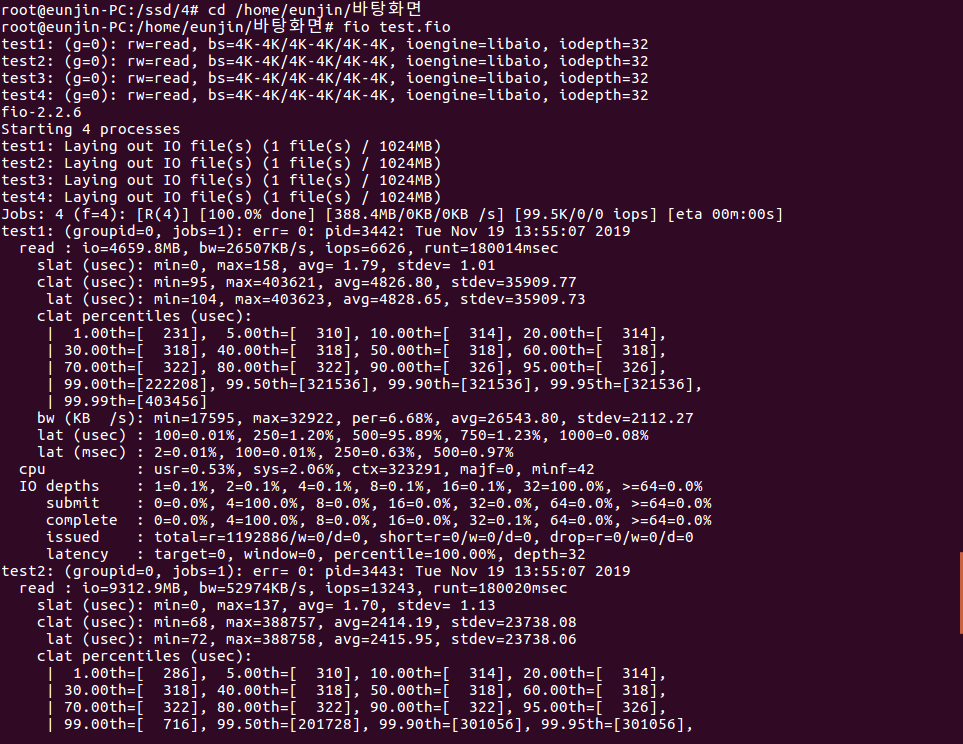
[그림 8] test 파일들을 삭제해줌으로써 디스크가 처음 상태로 돌아온 것을 확인할 수 있습니다.



* **다시 Fio 실행**

**cd /home/eunjin/바탕화면 : Fio job file이 저장돼 있는 위치로 이동**

**fio test.fio : fio 실행**



## **5-4. 실험 결과 및 고찰**

### **1. 실험 결과**

* **FIO 결과 그래프 (BW)**
* **FIO 결과 그래프 (IOPS)**
* **실험 설명**

weight의 비율이 100일 때, 200일 때, 400일 때, 800일 때를 비교하며 IOPS 및 BW를 측정하였습니다. read, write, random read, random write 4개의 옵션을 바꿔가며 실험하였습니다. IOPS와 BW의 평균값을 구하기 위하여 각 옵션 및 weight 당 5번씩 총 80번의 FIO 벤치마크를 돌려주어 실험을 진행하였습니다.

### **2. 고찰**

이번 실험을 통해 Cgroup weight 비율에 따른 디스크 I/O 성능 차이를 직접 눈으로

확인할 수 있었습니다. weight 비율이 커질수록 배수로 I/O 성능이 높아지는 것을 확인했고,

BW 역시 높아지는 것을 확인할 수 있었습니다. 직접 실험을 해봄으로 인해 이론으로 공부할

때 놓쳤던 부분을 새로이 더 알게 되어 더욱 흥미로운 실험이었습니다.

# **6. Cassandra DB를 이용한 Cgroup 성능 측정**

## **6-1. Cassandra DB 란?**

* **Apache**에서 개발한 **자바 기반 데이터베이스**이다.
* **분산형 자유-오픈소스** **NoSQL 데이터베이스** 관리 시스템이다..
* **대용량**과 **고성능 트렌젝션**을 요구하는 **SNS에 주로 사용**된다.
* **대량의 데이터**를 **분산된 서버에서 관리**해야 할 경우에 적합하다.
* 관계형 데이터베이스 보다 덜 제한적인 **일관성 모델**을 **사용**한다.
* **key, value 형태로 저장**하며, **key를 사용**하여 **데이터에 접근 및 관리**한다.

## **6-2. Cassandra DB 특징**

* 자바로 작성되었지만 **Python, C# 등 여러 프로그래밍 언어를 지원**한다.
* 데이터 간의 복잡한 관계 정의 등이 필요 없다.
* **대용량의 데이터 트렌젝션**에 대해 **고성능 처리**가 가능하다
* 단일 장애점 없이 **고성능을 제공**한다.
* **Consistent Hashing 방식**을 이용하여 데이터 분산, 로드 밸런싱을 위한 알고리즘을 더해 **요청이 적은 서비스를 요청이 많은 서버 구간으로 옮기는 방식**으로 구현한다.
* **대량의 요청**이 소수의 **서버에 몰리는 것**을 막아 **장애를 방지**하고 성능을 높인다.
* 데이터는 **메모리에 먼저** 쓰여지고, **일정 크기의 데이터**가 되면 **디스크로 옮겨진다**.

## **6-3. Cassandra DB 설치 방법**

### **1. JDK 설치**

**\* Ubuntu 16.04 LTS와 함께 패키지 된 버전을 사용하였습니다.**

* **자바 설치여부 확인**

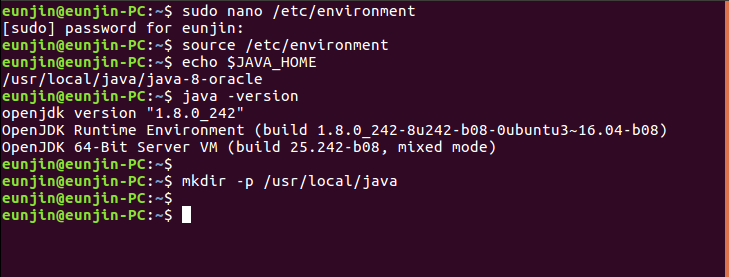
**java -version : 자바 버전 확인**



[그림 1] 자바가 설치되어 있지 않는 것을 확인할 수 있습니다.

1

* **자바가 이미 설치되어 있는 경우**



[그림 2] 1.8.0\_242 버전의 자바가 설치되어 있는 것을 확인할 수 있습니다.

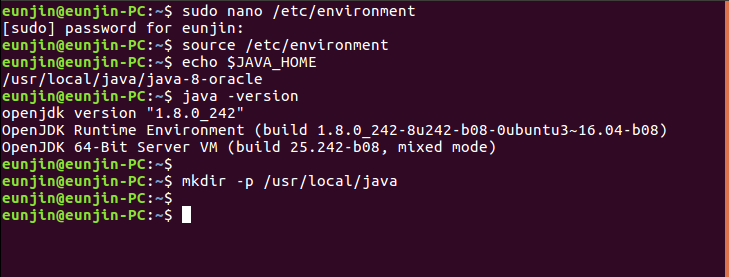
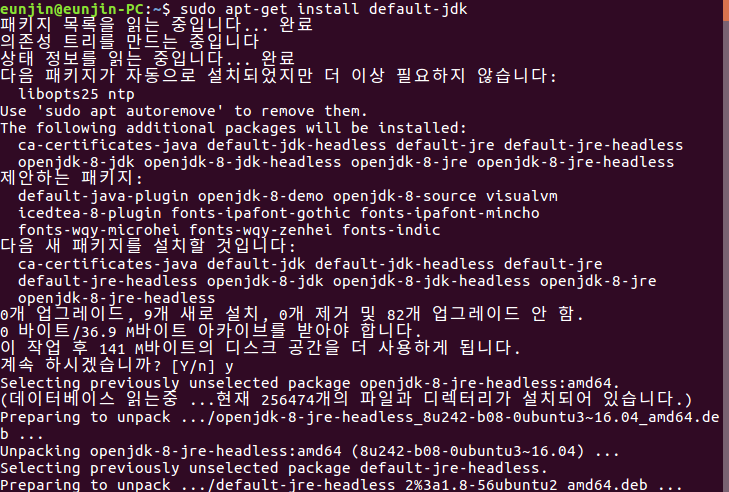
**\* 자바가 이미 설치되어 있는 경우 바로** [**Cassandra DB 설치**](#_2._Cassandra_DB) **or** [**환경 변수 설정**](#_1-1._환경_변수)**을 하시면 됩니다.**

* **JDK 설치**

**1)** **sudo apt-get install default-jdk**

**2) 계속 하시겠습니까? → y 선택**

[그림 4] JDK 설치하는 과정을 볼 수 있습니다



[그림 2] 자바가 설치되어 있는 것을 확인할 수 있습니다.

#### 1-1. 환경 변수 설정

* **설치한 자바의 경로 확인 후 복사**

1. **설치한 자바의 경로 확인**

**sudo update-alternatives –-config java**

1. **[그림 5] 빨간 박스에 있는 설치된 자바의 경로 복사**

**/usr/lib/jvm/java-8-openjdk-amd64 복사**

[그림 5] JAVA\_HOME 환경 변수 설정 시 입력해야 할 경로를 복사합니다.



**\* 설치된 자바의 경로는 다르게 표시될 수 있습니다.**

* **JAVA\_HOME 환경 변수 설정**

1. **환경 파일 열기**

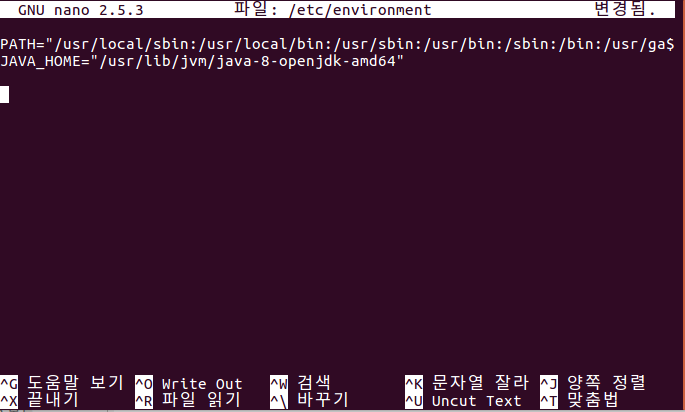
**sudo nano /etc/environment**

1. **환경 변수 설정**

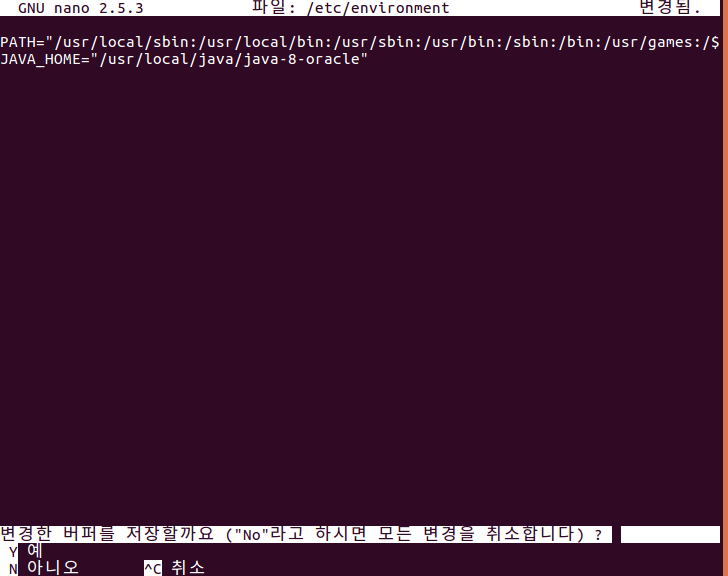
**[그림 5]에서 복사한 경로를 마지막 줄에 다음과 같은 형식으로 작성**

**→ JAVA\_HOME=”/usr/lib/jvm/java-8-openjdk-amd64”**

**( 위 형식의 강조 표시된 부분에 복사한 경로를 입력합니다. )**



[그림 6] 복사한 경로를 마지막줄에 형식에 맞추어 작성합니다.

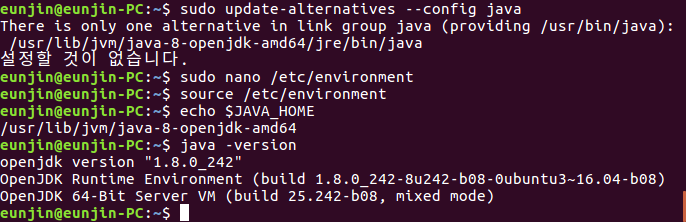
1. **Ctrl + x를 눌러 끝내기**
2. **변경한 버퍼를 저장할까요? → y**
3. **저장할 파일 이름 : /etc/environment → Enter**

* **환경 파일 reload**

**source /etc/environment : 환경 파일을 다시 로드합니다.**

* 환경 변수 설정 테스트

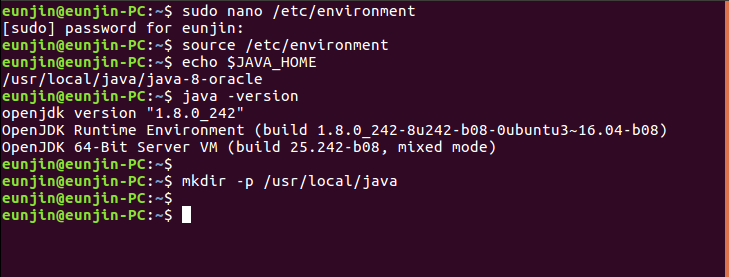
**echo $JAVA\_HOME : 방금 설정한 경로를 반환합니다.**



[그림 7] 환경 변수 설정이 완료된 것을 확인할 수 있습니다.

* **자바 설치 확인**

**java -version : 자바 버전 확인**

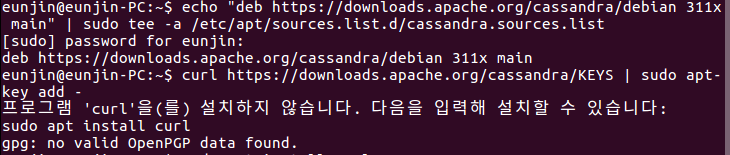


**[그림 2]** 1.8.0\_242 버전의 자바가 정상적으로 설치되었습니다.

### **2. Cassandra DB 설치**

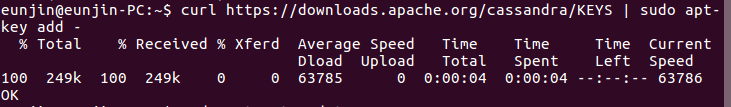
* **Apache Cassandra 3.11 버전 저장소 추가**

**echo “deb https://downloads.apache.org/cassandra/debian 311x main” | sudo tee -a /etc/apt/sources.list.d/cassandra.sources.list**



[그림 1] Cassandra 저장소 /etc/apt/sources.list.d/Cassandra.souroes.list 디렉토리 설치하기

* **Apache Cassandra 저장소 키 추가**

 **curl https://downloads.apache.org/cassandra/KEYS | sudo apt-key add –**

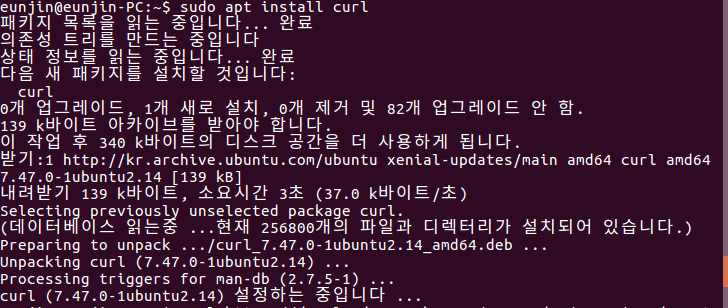
[그림 2] Apache Cassandra 저장소에 키를 추가하기

**\* “프로그램 curl을 설치하지 않았습니다.” 라는 문구가 뜬다면 하단을 참고하세요.**

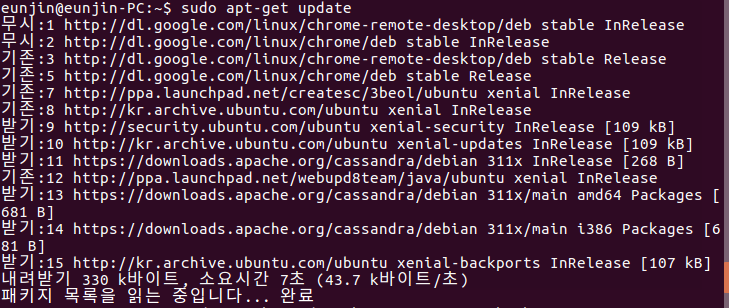
* **curl 명령어 설치**

**sudo apt install curl 명령어 입력 후 상단의**

**“Apache Cassandra 저장소 키 추가” 과정 실행**



* **리포지토리 업데이트**

 **sudo apt-get update : 패키지 색인 업데이트**

[그림 3] 리포지토리가 정상적으로 업데이트 된 것을 확인할 수 있습니다.

**\* 하단과 같은 오류가 발생하면 공개 키  A278B781FE4B2BDA를 하단의 형식으로 추가하세요.**

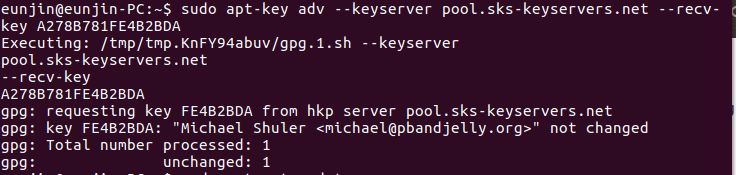
**GPG error: http://www.apache.org 311x InRelease: The following signatures couldn't be verified because the public key is not available: NO\_PUBKEY A278B781FE4B2BDA**

* **GPG 오류 해결**

**sudo apt-key adv –keyserver pool.sks-keyservers.net --recv-key**

**A278B781FE4B2BDA**

**명령어 입력 후 상단의 “리포지토리 업데이트” 과정 실행**

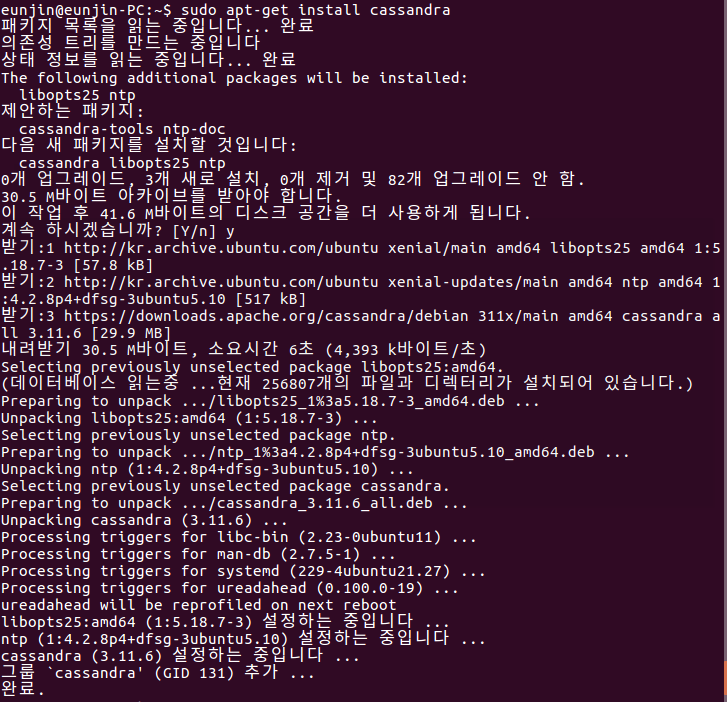


* **Cassandra 설치**

**1) sudo apt-get install Cassandra**

**2) 계속 하시겠습니까? → y 선택**

[그림 4] Cassandra 설치가 완료된 것을 확인할 수 있습니다.



### **3. Cassandra DB 설치 테스트**

* **Cassandra 시작**

**sudo service Cassandra start**

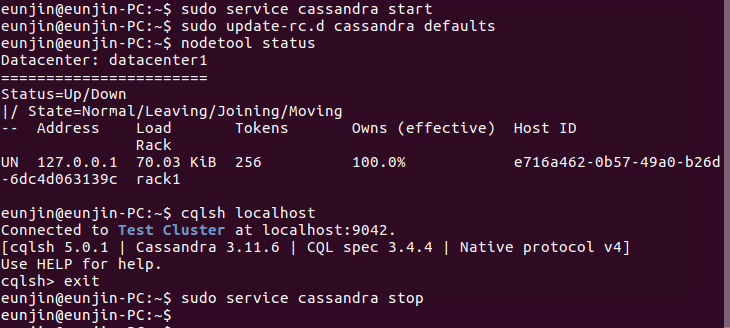
* **부팅 시 Cassandra 자동 시작 (원하지 않는 사람은 건너뛰시면 됩니다.)**

**sudo update-rc.d Cassandra defaults**

* **nodetool을 사용하여 현재 노드에서 Cassandra 상태 표시**

**nodetool status : Cassandra가 실행 중인지 확인할 수 있습니다.**

[그림 5] Cassandra가 정상적으로 작동하는 것을 확인할 수 있습니다.

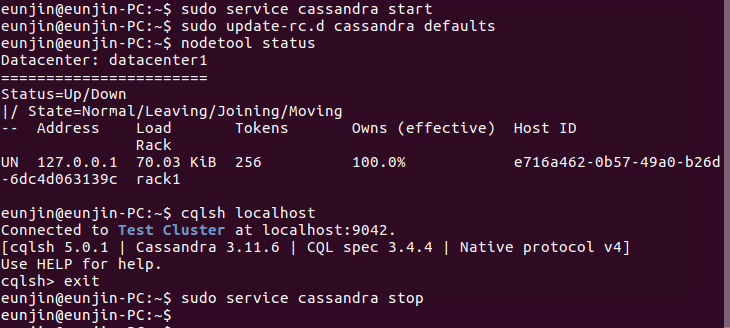


**\* nodetool status 명령어를 입력했을 때 나오는 실행결과 화면에서 UN은 Cassandra가 정상적으로 작동하고 있음을 의미합니다.**

* **cqlsh 쉘 연결**

**1)** **cqlsh localhost**

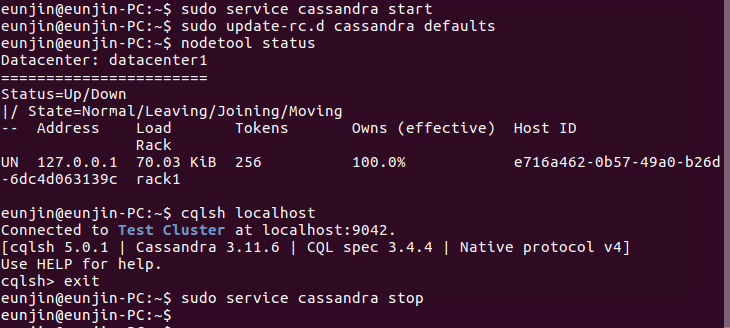
**2) 마지막 줄 cqlsh>에 exit 입력 후 종료 : cqlsh을 종료합니다.**



[그림 6] Cassandra가 정상적으로 작동하는 것을 확인할 수 있습니다.

* **Cassandra 종료**

**sudo servise Cassandra stop**



[그림 6] Cassandra를 종료합니다.

## **6-4. Cassandra DB를 이용한 Cgroup 성능 측정**

## **6-5. 실험 결과 및 고찰**

### **1. 실험 결과**

### **2. 고찰**

1. 정확히 말하면 /sys/fs/cgroup/blkio에 속해 있는 스레드의 스레드 ID(TID) 목록을 확인하는 것입니다. [↑](#footnote-ref-1)