운영체제실습

assignment 5

담당교수 : 김태석 교수님

학　　번 : 2021202058

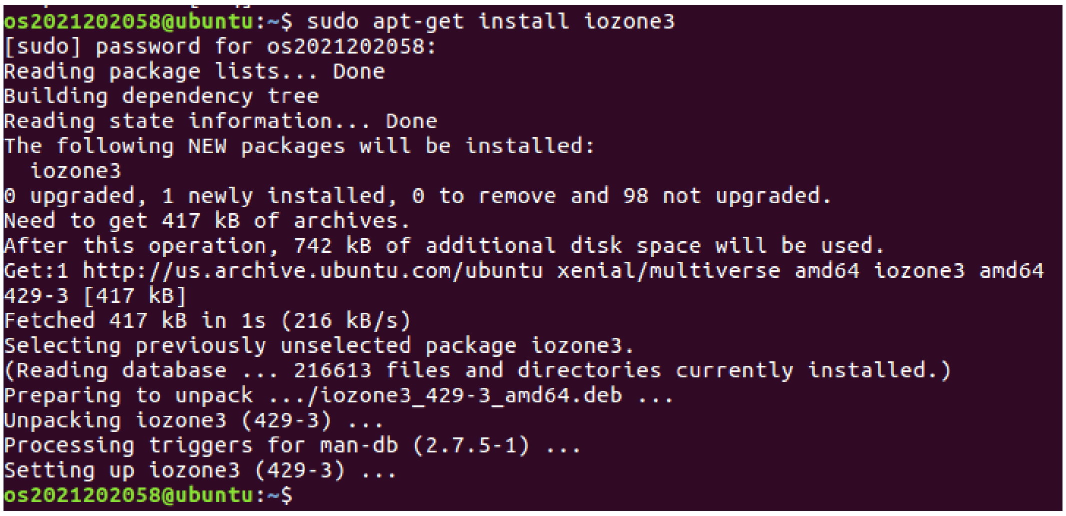
성　　명 : 송채영

1. Introduction

이번 과제는 Linux I/O scheduler 의 성능을 테스트하는 것이다. 사용되는 스케줄러는 각각 noop, CFQ, deadline 이며, 이러한 각 scheduler 의 성능 결과를 테스트하여 얻은 데이터를 표와 그래프로 시각화해서 확인해본다. 뿐만 아니라, 각 scheduler 가 어떤 동작과 어떤 성능을 보이는지를 확인하기 위해 IOZone 을 활용한다.

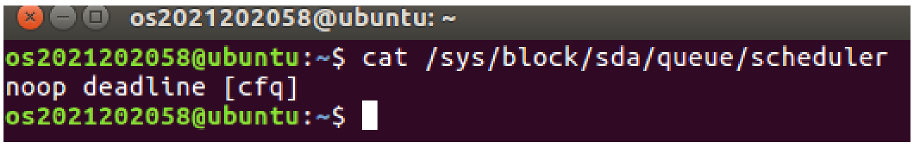
2. Conclusion & Analysis

우선 실험을 진행하기 위해 IOZone 을 설치해주었다.



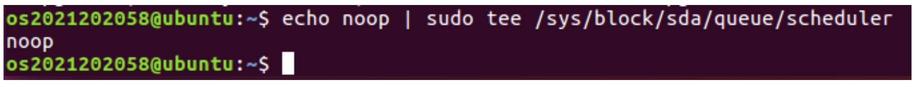
```
os2021202058@ubuntu:~$ sudo apt-get install iozone3
[sudo] password for os2021202058:
Reading package lists... Done
Building dependency tree
Reading state information... Done
The following NEW packages will be installed:
  iozone3
0 upgraded, 1 newly installed, 0 to remove and 98 not upgraded.
Need to get 417 kB of archives.
After this operation, 742 kB of additional disk space will be used.
Get:1 http://us.archive.ubuntu.com/ubuntu xenial/multiverse amd64 iozone3 amd64
429-3 [417 kB]
Fetched 417 kB in 1s (216 kB/s)
Selecting previously unselected package iozone3.
(Reading database ... 216613 files and directories currently installed.)
Preparing to unpack .../iozone3_429-3_amd64.deb ...
Unpacking iozone3 (429-3) ...
Processing triggers for man-db (2.7.5-1) ...
Setting up iozone3 (429-3) ...
os2021202058@ubuntu:~$
```

다음으로 cat /sys/block/sda/queue/scheduler 명령어를 통해 현재 해당하는 scheduler 를 확인하였다.



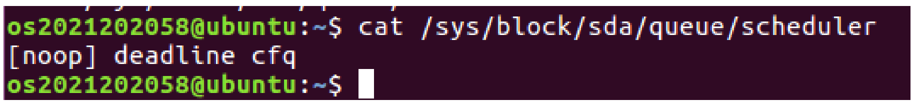
```
os2021202058@ubuntu:~$ cat /sys/block/sda/queue/scheduler
noop deadline [cfq]
os2021202058@ubuntu:~$
```

- Noop



```
os2021202058@ubuntu:~$ echo noop | sudo tee /sys/block/sda/queue/scheduler
noop
os2021202058@ubuntu:~$
```

스케줄러를 noop 으로 변경하였다.

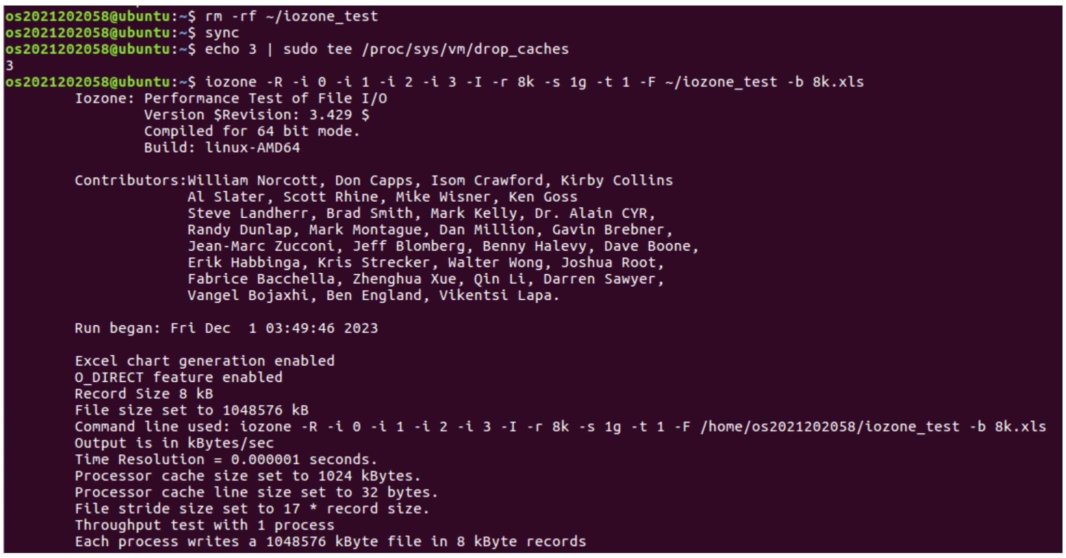


```
os2021202058@ubuntu:~$ cat /sys/block/sda/queue/scheduler
[noop] deadline cfq
os2021202058@ubuntu:~$
```

현재 스케줄러가 noop 인 것을 확인할 수 있다.

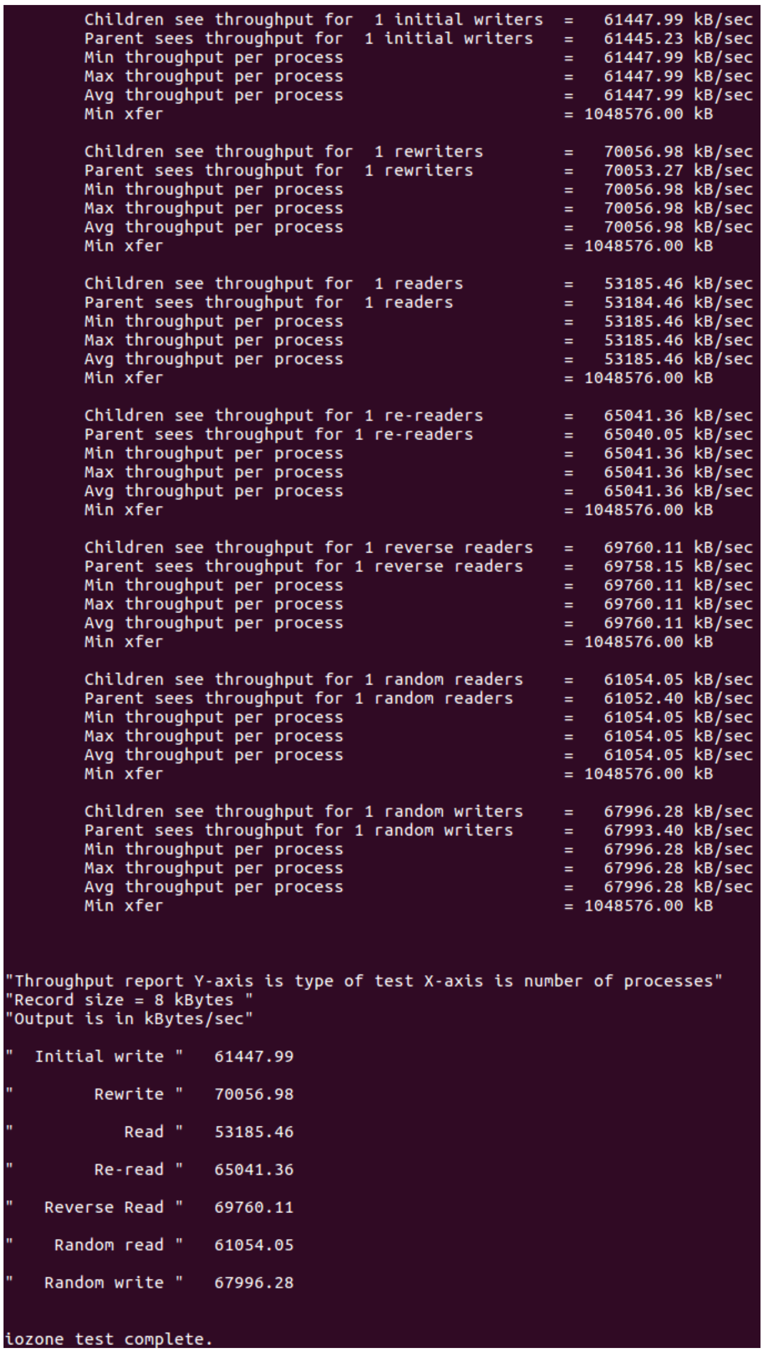
이후 IOZone 을 이용해 test 하는 부분이다. -R 옵션은 excel report 를 생성하고 -I 옵션을 통해 test 를 결정할 수 있다. 0, 1, 2, 3 를 선택했으며 각각 write/re-write read/re-read, random-read/write 에 해당하며, 3 번은 read-backwards 으로 파일 시스템이나 스토리지 장치의 성능을 테스트하기 위한 옵션 중 하나이다. 이 옵션을 사용해 파일을 역순으로 읽는 작업을 수행할 수 있다. **3 번, read-backwards 연산을 추가적으로 선택한 이유는 파일이나 데이터가 역순으로 배치된 경우(예를 들어 시간의 흐름에 따라 역순으로 저장되는 경우)에서도 안정적으로 성능을 유지할 수 있는지에 대한 점과, 그러한 특수한 상황에서의 성능을 테스트해보고 싶어 선택하였다.** -r 옵션을 통해서 record size 를 변경할 수 있고 record size 는(8k, 16k, 32k, 64k, 128k, 256k, 512k, 8m, 16m)에 해당한다. -s 옵션을 사용해 파일 사이즈를 변경할 수 있으며 1g(1GB)으로 변경 없이 사용하였다. -t 옵션을 사용하여 thread or process 의 개수를 1 로 설정하고 -F 옵션을 사용하여 마지막으로 -b 옵션을 사용하여 파일의 경로를 ~/iozone_test 로 해주었고 -b 옵션을 사용하여 파일 이름을 설정해주었다.

8kb

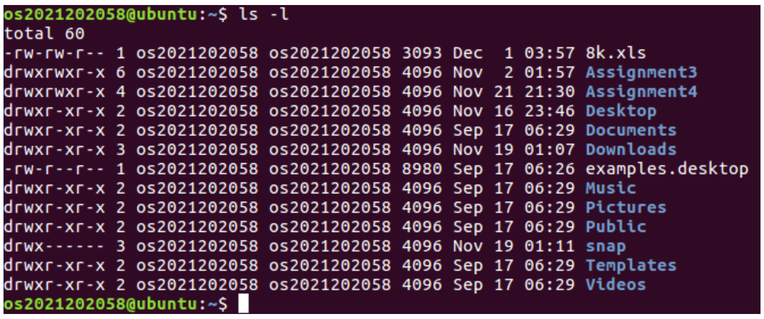


```
os2021202058@ubuntu:~$ rm -rf ~/iozone_test
os2021202058@ubuntu:~$ sync
os2021202058@ubuntu:~$ echo 3 | sudo tee /proc/sys/vm/drop_caches
3
os2021202058@ubuntu:~$ iozone -R -i 0 -i 1 -i 2 -i 3 -I -r 8k -s 1g -t 1 -F ~/iozone_test -b 8k.xls
        Iozone: Performance Test of File I/O
                Version $Revision: 3.429 $
                Compiled for 64 bit mode.
                Build: linux-AMD64

        Contributors:William Norcott, Don Capps, Isom Crawford, Kirby Collins
                     Al Slater, Scott Rhine, Mike Wisner, Ken Goss
                     Steve Landherr, Brad Smith, Mark Kelly, Dr. Alain CYR,
                     Randy Dunlap, Mark Montague, Dan Million, Gavin Brebner,
                     Jean-Marc Zucconi, Jeff Blomberg, Benny Halevy, Dave Boone,
                     Erik Habbinga, Kris Strecker, Walter Wong, Joshua Root,
                     Fabrice Bacchella, Zhenghua Xue, Qin Li, Darren Sawyer,
                     Vangel Bojaxhi, Ben England, Vikentsi Lapa.

        Run began: Fri Dec  1 03:49:46 2023

        Excel chart generation enabled
        O_DIRECT feature enabled
        Record Size 8 kB
        File size set to 1048576 kB
        Command line used: iozone -R -i 0 -i 1 -i 2 -i 3 -I -r 8k -s 1g -t 1 -F /home/os2021202058/iozone_test -b 8k.xls
        Output is in kBytes/sec
        Time Resolution = 0.000001 seconds.
        Processor cache size set to 1024 kBytes.
        Processor cache line size set to 32 bytes.
        File stride size set to 17 * record size.
        Throughput test with 1 process
        Each process writes a 1048576 kByte file in 8 kByte records
```

매 실험 전 캐시 및 버퍼를 비워 실험에 영향을 주는 요소를 제거하였다.



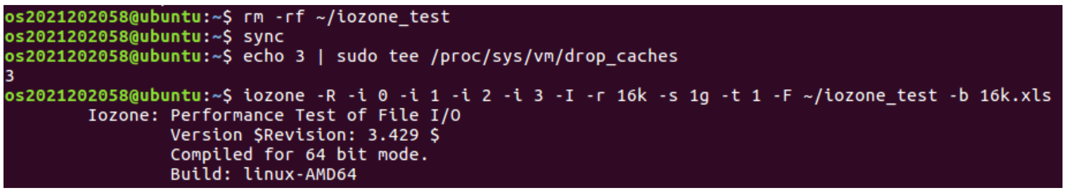
```
        Children see throughput for  1 initial writers  =   61447.99 kB/sec
        Parent sees throughput for  1 initial writers   =   61445.23 kB/sec
        Min throughput per process                      =   61447.99 kB/sec
        Max throughput per process                      =   61447.99 kB/sec
        Avg throughput per process                      =   61447.99 kB/sec
        Min xfer                                        = 1048576.00 kB

        Children see throughput for  1 rewriters        =   70056.98 kB/sec
        Parent sees throughput for  1 rewriters         =   70053.27 kB/sec
        Min throughput per process                      =   70056.98 kB/sec
        Max throughput per process                      =   70056.98 kB/sec
        Avg throughput per process                      =   70056.98 kB/sec
        Min xfer                                        = 1048576.00 kB

        Children see throughput for  1 readers          =   53185.46 kB/sec
        Parent sees throughput for  1 readers           =   53184.46 kB/sec
        Min throughput per process                      =   53185.46 kB/sec
        Max throughput per process                      =   53185.46 kB/sec
        Avg throughput per process                      =   53185.46 kB/sec
        Min xfer                                        = 1048576.00 kB

        Children see throughput for 1 re-readers        =   65041.36 kB/sec
        Parent sees throughput for 1 re-readers         =   65040.05 kB/sec
        Min throughput per process                      =   65041.36 kB/sec
        Max throughput per process                      =   65041.36 kB/sec
        Avg throughput per process                      =   65041.36 kB/sec
        Min xfer                                        = 1048576.00 kB

        Children see throughput for 1 reverse readers   =   69760.11 kB/sec
        Parent sees throughput for 1 reverse readers    =   69758.15 kB/sec
        Min throughput per process                      =   69760.11 kB/sec
        Max throughput per process                      =   69760.11 kB/sec
        Avg throughput per process                      =   69760.11 kB/sec
        Min xfer                                        = 1048576.00 kB

        Children see throughput for 1 random readers    =   61054.05 kB/sec
        Parent sees throughput for 1 random readers     =   61052.40 kB/sec
        Min throughput per process                      =   61054.05 kB/sec
        Max throughput per process                      =   61054.05 kB/sec
        Avg throughput per process                      =   61054.05 kB/sec
        Min xfer                                        = 1048576.00 kB

        Children see throughput for 1 random writers    =   67996.28 kB/sec
        Parent sees throughput for 1 random writers     =   67993.40 kB/sec
        Min throughput per process                      =   67996.28 kB/sec
        Max throughput per process                      =   67996.28 kB/sec
        Avg throughput per process                      =   67996.28 kB/sec
        Min xfer                                        = 1048576.00 kB



"Throughput report Y-axis is type of test X-axis is number of processes"
"Record size = 8 kBytes "
"Output is in kBytes/sec"

"  Initial write "   61447.99

"        Rewrite "   70056.98

"           Read "   53185.46

"        Re-read "   65041.36

"   Reverse Read "   69760.11

"    Random read "   61054.05

"   Random write "   67996.28


iozone test complete.
```



```
os2021202058@ubuntu:~$ ls -l
total 60
-rw-rw-r-- 1 os2021202058 os2021202058 3093 Dec  1 03:57 8k.xls
drwxrwxr-x 6 os2021202058 os2021202058 4096 Nov  2 01:57 Assignment3
drwxrwxr-x 4 os2021202058 os2021202058 4096 Nov 21 21:30 Assignment4
drwxr-xr-x 2 os2021202058 os2021202058 4096 Nov 16 23:46 Desktop
drwxr-xr-x 2 os2021202058 os2021202058 4096 Sep 17 06:29 Documents
drwxr-xr-x 3 os2021202058 os2021202058 4096 Nov 19 01:07 Downloads
-rw-r--r-- 1 os2021202058 os2021202058 8980 Sep 17 06:26 examples.desktop
drwxr-xr-x 2 os2021202058 os2021202058 4096 Sep 17 06:29 Music
drwxr-xr-x 2 os2021202058 os2021202058 4096 Sep 17 06:29 Pictures
drwxr-xr-x 2 os2021202058 os2021202058 4096 Sep 17 06:29 Public
drwx------ 3 os2021202058 os2021202058 4096 Nov 19 01:11 snap
drwxr-xr-x 2 os2021202058 os2021202058 4096 Sep 17 06:29 Templates
drwxr-xr-x 2 os2021202058 os2021202058 4096 Sep 17 06:29 Videos
os2021202058@ubuntu:~$
```

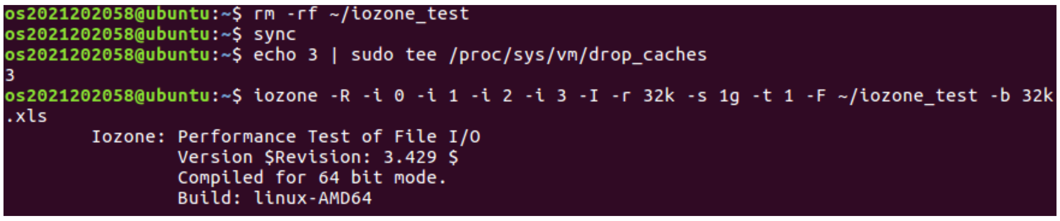
Ls -l 명령어를 사용해 확인해보면 엑셀 파일이 생성된 것을 확인할 수 있다.

16kb



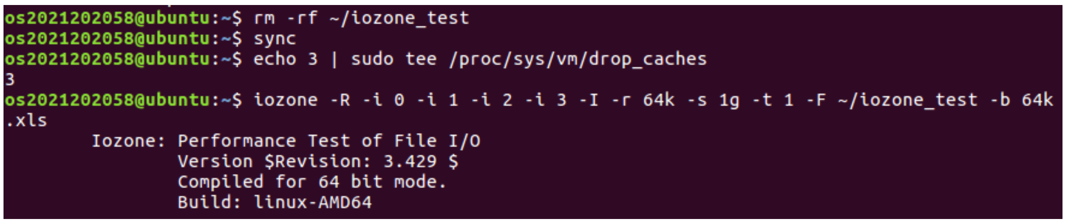
```
os2021202058@ubuntu:~$ rm -rf ~/iozone_test
os2021202058@ubuntu:~$ sync
os2021202058@ubuntu:~$ echo 3 | sudo tee /proc/sys/vm/drop_caches
3
os2021202058@ubuntu:~$ iozone -R -i 0 -i 1 -i 2 -i 3 -I -r 16k -s 1g -t 1 -F ~/iozone_test -b 16k.xls
        Iozone: Performance Test of File I/O
                Version $Revision: 3.429 $
                Compiled for 64 bit mode.
                Build: linux-AMD64
```

32kb



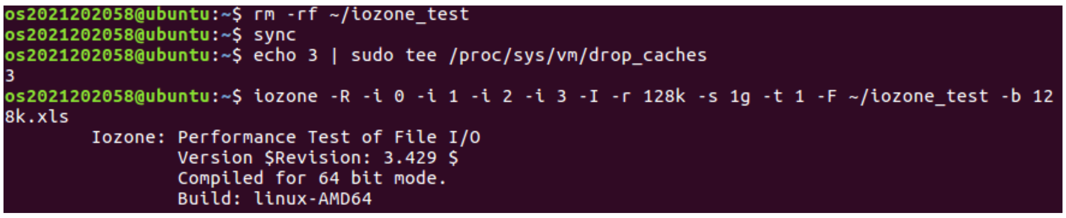
```
os2021202058@ubuntu:~$ rm -rf ~/iozone_test
os2021202058@ubuntu:~$ sync
os2021202058@ubuntu:~$ echo 3 | sudo tee /proc/sys/vm/drop_caches
3
os2021202058@ubuntu:~$ iozone -R -i 0 -i 1 -i 2 -i 3 -I -r 32k -s 1g -t 1 -F ~/iozone_test -b 32k
.xls
        Iozone: Performance Test of File I/O
                Version $Revision: 3.429 $
                Compiled for 64 bit mode.
                Build: linux-AMD64
```

64kb



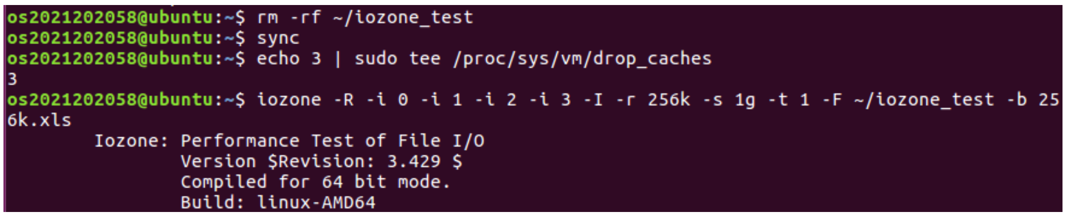
```
os2021202058@ubuntu:~$ rm -rf ~/iozone_test
os2021202058@ubuntu:~$ sync
os2021202058@ubuntu:~$ echo 3 | sudo tee /proc/sys/vm/drop_caches
3
os2021202058@ubuntu:~$ iozone -R -i 0 -i 1 -i 2 -i 3 -I -r 64k -s 1g -t 1 -F ~/iozone_test -b 64k
.xls
        Iozone: Performance Test of File I/O
                Version $Revision: 3.429 $
                Compiled for 64 bit mode.
                Build: linux-AMD64
```

128kb



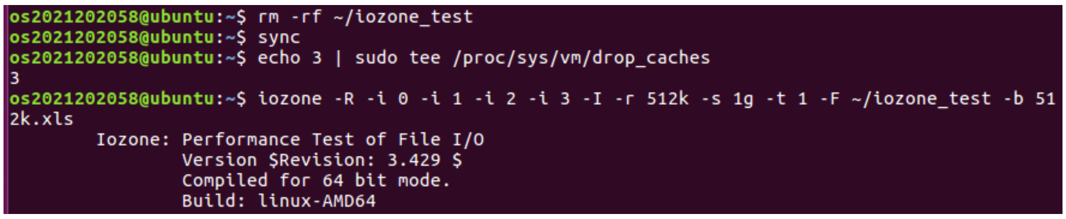
```
os2021202058@ubuntu:~$ rm -rf ~/iozone_test
os2021202058@ubuntu:~$ sync
os2021202058@ubuntu:~$ echo 3 | sudo tee /proc/sys/vm/drop_caches
3
os2021202058@ubuntu:~$ iozone -R -i 0 -i 1 -i 2 -i 3 -I -r 128k -s 1g -t 1 -F ~/iozone_test -b 12
8k.xls
        Iozone: Performance Test of File I/O
                Version $Revision: 3.429 $
                Compiled for 64 bit mode.
                Build: linux-AMD64
```

256kb



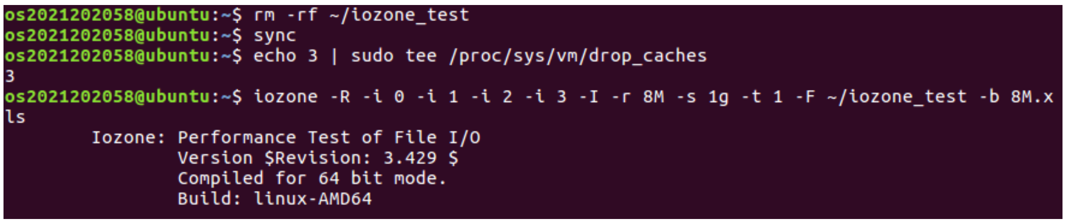
```
os2021202058@ubuntu:~$ rm -rf ~/iozone_test
os2021202058@ubuntu:~$ sync
os2021202058@ubuntu:~$ echo 3 | sudo tee /proc/sys/vm/drop_caches
3
os2021202058@ubuntu:~$ iozone -R -i 0 -i 1 -i 2 -i 3 -I -r 256k -s 1g -t 1 -F ~/iozone_test -b 25
6k.xls
        Iozone: Performance Test of File I/O
                Version $Revision: 3.429 $
                Compiled for 64 bit mode.
                Build: linux-AMD64
```

512kb

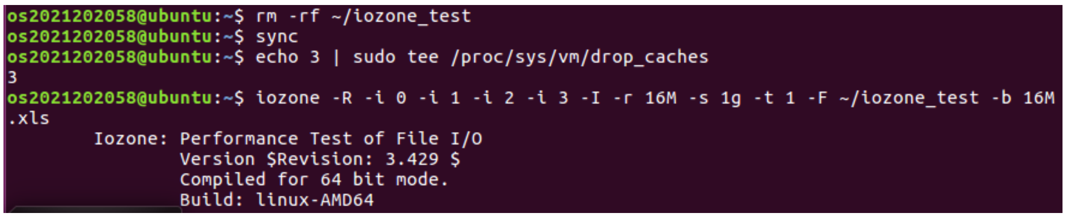


```
os2021202058@ubuntu:~$ rm -rf ~/iozone_test
os2021202058@ubuntu:~$ sync
os2021202058@ubuntu:~$ echo 3 | sudo tee /proc/sys/vm/drop_caches
3
os2021202058@ubuntu:~$ iozone -R -i 0 -i 1 -i 2 -i 3 -I -r 512k -s 1g -t 1 -F ~/iozone_test -b 51
2k.xls
        Iozone: Performance Test of File I/O
                Version $Revision: 3.429 $
                Compiled for 64 bit mode.
                Build: linux-AMD64
```

8M


```
os2021202058@ubuntu:~$ rm -rf ~/iozone_test
os2021202058@ubuntu:~$ sync
os2021202058@ubuntu:~$ echo 3 | sudo tee /proc/sys/vm/drop_caches
3
os2021202058@ubuntu:~$ iozone -R -i 0 -i 1 -i 2 -i 3 -I -r 8M -s 1g -t 1 -F ~/iozone_test -b 8M.x
ls
        Iozone: Performance Test of File I/O
                Version $Revision: 3.429 $
                Compiled for 64 bit mode.
                Build: linux-AMD64
```

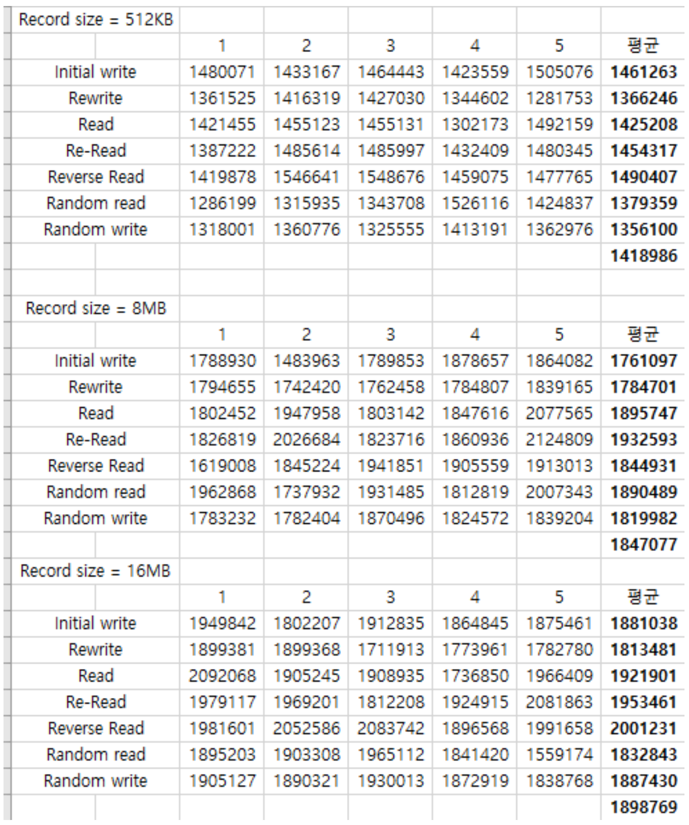
16M


```
os2021202058@ubuntu:~$ rm -rf ~/iozone_test
os2021202058@ubuntu:~$ sync
os2021202058@ubuntu:~$ echo 3 | sudo tee /proc/sys/vm/drop_caches
3
os2021202058@ubuntu:~$ iozone -R -i 0 -i 1 -i 2 -i 3 -I -r 16M -s 1g -t 1 -F ~/iozone_test -b 16M
.xls
        Iozone: Performance Test of File I/O
                Version $Revision: 3.429 $
                Compiled for 64 bit mode.
                Build: linux-AMD64
```

Record size 를 변경해가며 실행한 결과사진이다. 매 test 전 캐시 및 버퍼를 지웠다.

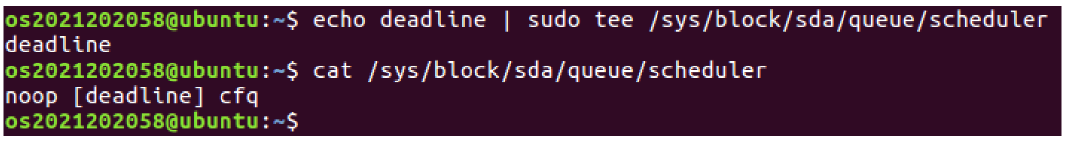
아래의 사진은 noop scheduler 의 성능 결과를 표로 나타낸 것이다. 실험을 5 회 실시하여 평균값을 구하였다.

			NOOP			
Record size = 8KB						
	1	2	3	4	5	평균
Initial write	61447.99	62263.26	65450.32	61838.14	62352.17	**62670.37**
Rewrite	70056.98	69439.32	62631.44	74260.16	67510.61	**68779.7**
Read	53185.46	54150.91	60692.88	58471.6	60734.37	**57447.04**
Re-Read	65041.36	53969.02	65178.33	67162.28	60751.51	**62420.5**
Reverse Read	69760.11	49511.79	71357.3	66692.18	63078.98	**64080.07**
Random read	61054.05	67944.48	60142	64981.36	70644.33	**64953.24**
Random write	67996.28	64097.01	71902.47	60572.63	60250.57	**64963.79**
						63616.39
Record size = 16KB						
	1	2	3	4	5	평균
Initial write	119358.1	108899.3	112972.2	124363	125280.3	**118174.6**
Rewrite	124039.9	126098.6	132049.7	126668.4	132027.2	**128176.8**
Read	135320.6	104206.3	126178.3	128364.9	104695.4	**119753.1**
Re-Read	120988.4	103101.5	110138.2	122341.8	111344	**113582.8**
Reverse Read	120725.7	91311.24	127809.5	134125.3	128904.3	**120575.2**
Random read	93639.9	115881.1	127802.7	110655.3	111611.4	**111918.1**
Random write	139386.3	133802.1	119932.4	136730.4	143388.5	**134647.9**
						120975.5
Record size = 32KB						
	1	2	3	4	5	평균
Initial write	211708.8	173030.9	225302.2	218991.7	252238.6	**216254.5**
Rewrite	234510.2	201352.3	197418.3	233733.7	233867.2	**220176.3**
Read	235539.8	183578.8	195770.8	205409.3	223161.4	**208692**
Re-Read	221224	181293.2	213221.1	215527.1	216820.3	**209617.1**
Reverse Read	197105.4	183669	207344.7	216347	201126.5	**201118.5**
Random read	192618.6	194748.5	192811.4	203999	203414.6	**197518.4**
Random write	212147.8	185573	202300.6	217187.3	203466.8	**204135.1**
						208216
Record size = 64KB						
	1	2	3	4	5	평균
Initial write	413182.9	472191.2	438957.9	459372.3	473789.6	**451498.8**
Rewrite	446026.8	394083.6	398515.1	385963.2	403810.7	**405679.9**
Read	386639.5	370634.3	391096.4	417066.4	382810.7	**389649.5**
Re-Read	358475.5	374564.1	422805.4	374652.1	454498.2	**396999.1**
Reverse Read	446693.1	372800.6	355534.4	385408.1	395045.2	**391096.3**
Random read	380152	415815.6	405025.6	346163	412930.7	**392017.4**
Random write	357991	352027.2	389248.1	399224.7	404534.8	**380605.1**
						401078
Record size = 128KB						
	1	2	3	4	5	평균
Initial write	608398.8	601139.5	639028.8	627270.1	599679.8	**615103.4**
Rewrite	595140.3	618701.9	622756	632287.8	617564.6	**617290.1**
Read	607407.1	628348.4	618291.8	581039.9	650055.1	**617028.5**
Re-Read	693052.4	568582.9	661638.1	563549	613816.1	**620127.7**
Reverse Read	674357	599415.8	629310.1	618823.1	677904.5	**639962.1**
Random read	652322.6	591206.6	599849.6	598343.8	563922.4	**601129**
Random write	563951.3	565154.3	635925.3	561390.9	586120.5	**582508.5**
						613307
Record size = 256KB						
	1	2	3	4	5	평균
Initial write	976910.8	999260.6	973160.8	1012789	1052854	**1002995**
Rewrite	932767.3	973161	964344.3	976912.5	976480.4	**964733.1**
Read	933800.8	1112165	993528.4	1080692	990790.3	**1022195**
Re-Read	935540.2	1030662	1041751	1014118	1033540	**1011122**
Reverse Read	1082769	1062471	1064092	1099041	1013571	**1064389**
Random read	948334.3	980302.7	943831.4	875506.7	944010.9	**938397.2**
Random write	1001557	1006541	943356	958270.4	948274	**971599.7**
						996490.2



Record size = 512KB						
	1	2	3	4	5	평균
Initial write	1480071	1433167	1464443	1423559	1505076	**1461263**
Rewrite	1361525	1416319	1427030	1344602	1281753	**1366246**
Read	1421455	1455123	1455131	1302173	1492159	**1425208**
Re-Read	1387222	1485614	1485997	1432409	1480345	**1454317**
Reverse Read	1419878	1546641	1548676	1459075	1477765	**1490407**
Random read	1286199	1315935	1343708	1526116	1424837	**1379359**
Random write	1318001	1360776	1325555	1413191	1362976	**1356100**
						1418986
Record size = 8MB						
	1	2	3	4	5	평균
Initial write	1788930	1483963	1789853	1878657	1864082	**1761097**
Rewrite	1794655	1742420	1762458	1784807	1839165	**1784701**
Read	1802452	1947958	1803142	1847616	2077565	**1895747**
Re-Read	1826819	2026684	1823716	1860936	2124809	**1932593**
Reverse Read	1619008	1845224	1941851	1905559	1913013	**1844931**
Random read	1962868	1737932	1931485	1812819	2007343	**1890489**
Random write	1783232	1782404	1870496	1824572	1839204	**1819982**
						1847077
Record size = 16MB						
	1	2	3	4	5	평균
Initial write	1949842	1802207	1912835	1864845	1875461	**1881038**
Rewrite	1899381	1899368	1711913	1773961	1782780	**1813481**
Read	2092068	1905245	1908935	1736850	1966409	**1921901**
Re-Read	1979117	1969201	1812208	1924915	2081863	**1953461**
Reverse Read	1981601	2052586	2083742	1896568	1991658	**2001231**
Random read	1895203	1903308	1965112	1841420	1559174	**1832843**
Random write	1905127	1890321	1930013	1872919	1838768	**1887430**
						1898769

- Deadline



```
os2021202058@ubuntu:~$ echo deadline | sudo tee /sys/block/sda/queue/scheduler
deadline
os2021202058@ubuntu:~$ cat /sys/block/sda/queue/scheduler
noop [deadline] cfq
os2021202058@ubuntu:~$
```

현재 스케줄러를 deadline 으로 바꿔준 후 명령어를 통해 올바르게 변경한 것을 확인하였다.

Noop scheduler 에서 수행했던 과정을 동일하게 진행하였다. Record size 를 8kb, 16kb, 32kb, 64kb, 128kb, 256kb, 512kb, 8mb, 16mb 로 변경하며 수행하였으며 매 test 전 캐시 및 버퍼를 지웠다.

아래의 사진은 deadline scheduler 의 성능 결과를 표로 나타낸 것이다. 실험을 5 회 실시하여 평균값을 구하였다.

Record size = 8KB						
	1	2	3	4	5	평균
Initial write	61268.84	68486.38	66154.63	73715.27	69171.03	**67759.23**
Rewrite	69004.17	71179.78	65298.82	65301.82	70007.03	**68158.33**
Read	56805.42	58588.23	49106.06	61824.32	59489.79	**57162.76**
Re-Read	66344.23	57412.78	67469.52	55695.46	48968.61	**59178.12**
Reverse Read	48734.74	59710.79	66423.17	62402.64	57943.63	**59043**
Random read	66401.41	65466.91	65309.5	63633.43	55427.36	**63247.72**
Random write	59259.86	63455.25	65909.05	61647.05	60673.76	**62188.99**
						62391.16

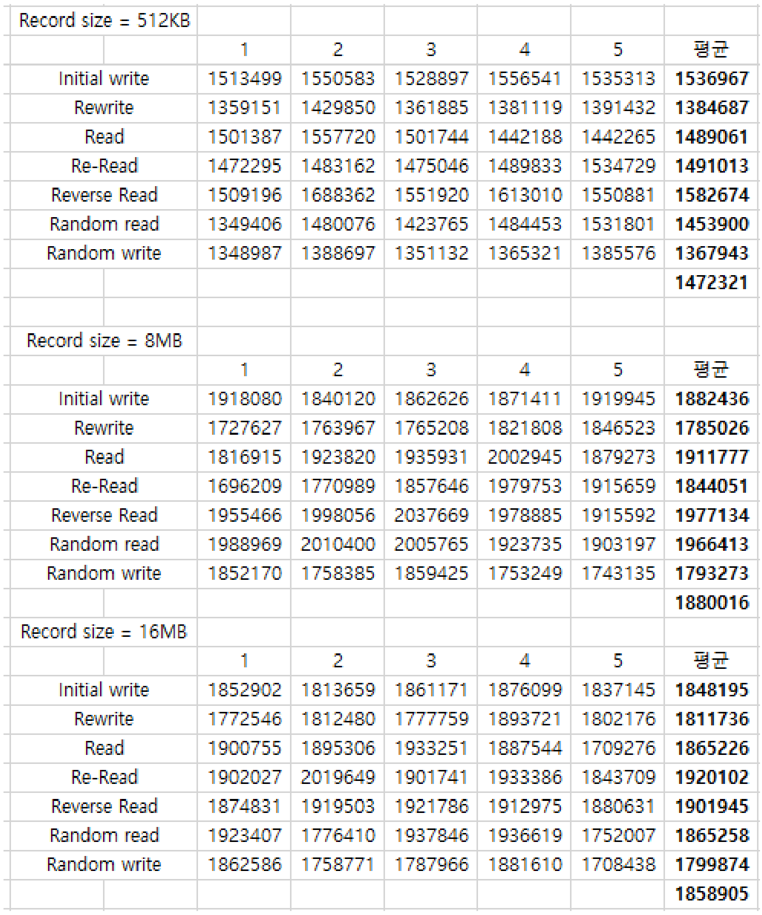
Record size = 16KB						
	1	2	3	4	5	평균
Initial write	128339.9	136001.8	138193.5	140116.2	139336.4	**136397.6**
Rewrite	100650.6	131817.6	107109.3	130787	131198.8	**120312.7**
Read	114532.2	128038.7	96527.45	132042	99644.15	**114156.9**
Re-Read	119356.4	132385.7	136093.5	134579.5	120014.4	**128485.9**
Reverse Read	100742.3	134431	128733.7	115408.4	126574.9	**121178.1**
Random read	87855.2	119822.2	107377.3	104932	123604.7	**108718.3**
Random write	126021.9	117777.7	116146	122186	120880.7	**120602.5**
						121407.4

Record size = 32KB						
	1	2	3	4	5	평균
Initial write	289208.7	272284.8	262138.7	228539.3	236178.8	**257670.1**
Rewrite	188889.7	230160.3	233778.5	248199.1	209084	**222022.3**
Read	183332.5	196265.8	228603.7	247007.7	207837.9	**212609.5**
Re-Read	254760.4	194504	173422.8	236032.5	173124.4	**206368.8**
Reverse Read	210872.5	259060.7	229853	213812.8	222503.1	**227220.4**
Random read	192275.2	258164	188585.6	190635.6	212848.9	**208501.9**
Random write	200929	179108.3	208546.5	233483.3	224232.2	**209259.8**
						220521.8

Record size = 64KB						
	1	2	3	4	5	평균
Initial write	438146.1	460605.6	466598.9	489709.4	447849.3	**460581.8**
Rewrite	297343.2	394732.4	335096.5	420880.4	358973	**361405.1**
Read	382853.8	334768.5	434726.3	291260.8	312004.1	**351122.7**
Re-Read	308962.1	302321.7	303748.4	422927.3	411404.9	**349872.9**
Reverse Read	423418.3	397090.3	394297.6	358415.5	354467.1	**385537.7**
Random read	343676.8	373513.9	446151.9	322860.4	320021.3	**361244.9**
Random write	292944.6	413129.3	450956.9	291044.8	303314.6	**350278.1**
						374291.9

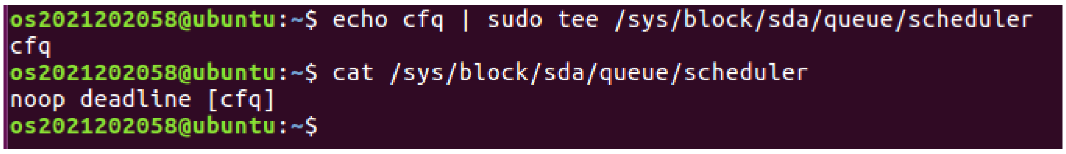
Record size = 128KB						
	1	2	3	4	5	평균
Initial write	641288.1	684196.7	679431.8	666946.9	593978.9	**653168.5**
Rewrite	662804.5	645480.6	612920.6	677423.7	576607.9	**635047.5**
Read	668991.5	614961.6	635864.1	642212.7	617420.6	**635890.1**
Re-Read	653860.6	616074.8	659900.1	663321.1	600674.2	**638766.2**
Reverse Read	660266.3	686379.1	638996.1	647237.6	632929	**653161.6**
Random read	587458.6	626627.9	607041.8	593984.3	575019.1	**598026.3**
Random write	676777.4	572234	634110.9	558198.6	617433.1	**611750.8**
						632258.7

Record size = 256KB						
	1	2	3	4	5	평균
Initial write	1128873	1110083	1113524	1097594	1000502	**1090115**
Rewrite	998394.7	910008.9	870738.8	908041.9	897648.1	**916966.5**
Read	991716.1	961573.6	985961.4	939541.9	1111874	**998133.4**
Re-Read	963838.1	1011975	986053.1	971213.8	1005178	**987651.7**
Reverse Read	934636.4	1021193	1142391	1138202	999731	**1047231**
Random read	975820.8	954210.4	963105.9	1057371	987176.4	**987537**
Random write	863006.3	874745.6	1038558	874576.3	925648.9	**915307**
						991848.7



Record size = 512KB						
	1	2	3	4	5	평균
Initial write	1513499	1550583	1528897	1556541	1535313	**1536967**
Rewrite	1359151	1429850	1361885	1381119	1391432	**1384687**
Read	1501387	1557720	1501744	1442188	1442265	**1489061**
Re-Read	1472295	1483162	1475046	1489833	1534729	**1491013**
Reverse Read	1509196	1688362	1551920	1613010	1550881	**1582674**
Random read	1349406	1480076	1423765	1484453	1531801	**1453900**
Random write	1348987	1388697	1351132	1365321	1385576	**1367943**
						1472321
Record size = 8MB						
	1	2	3	4	5	평균
Initial write	1918080	1840120	1862626	1871411	1919945	**1882436**
Rewrite	1727627	1763967	1765208	1821808	1846523	**1785026**
Read	1816915	1923820	1935931	2002945	1879273	**1911777**
Re-Read	1696209	1770989	1857646	1979753	1915659	**1844051**
Reverse Read	1955466	1998056	2037669	1978885	1915592	**1977134**
Random read	1988969	2010400	2005765	1923735	1903197	**1966413**
Random write	1852170	1758385	1859425	1753249	1743135	**1793273**
						1880016
Record size = 16MB						
	1	2	3	4	5	평균
Initial write	1852902	1813659	1861171	1876099	1837145	**1848195**
Rewrite	1772546	1812480	1777759	1893721	1802176	**1811736**
Read	1900755	1895306	1933251	1887544	1709276	**1865226**
Re-Read	1902027	2019649	1901741	1933386	1843709	**1920102**
Reverse Read	1874831	1919503	1921786	1912975	1880631	**1901945**
Random read	1923407	1776410	1937846	1936619	1752007	**1865258**
Random write	1862586	1758771	1787966	1881610	1708438	**1799874**
						1858905

- CFQ



```
os2021202058@ubuntu:~$ echo cfq | sudo tee /sys/block/sda/queue/scheduler
cfq
os2021202058@ubuntu:~$ cat /sys/block/sda/queue/scheduler
noop deadline [cfq]
os2021202058@ubuntu:~$
```

현재 스케줄러를 cfq 로 바꿔준 후 명령어를 통해 올바르게 변경한 것을 확인하였다.

Noop scheduler 에서 수행했던 과정을 동일하게 진행하였다. Record size 를 8kb, 16kb, 32kb, 64kb, 128kb, 256kb, 512kb, 8mb, 16mb 로 변경하며 수행하였으며 매 test 전 캐시 및 버퍼를 지웠다.

아래의 사진은 cfq scheduler 의 성능 결과를 표로 나타낸 것이다. 실험을 5 회 실시하여 평균값을 구하였다.

Record size = 8KB						
	1	2	3	4	5	평균
Initial write	59740.24	68591.73	70218.7	67990.55	68134.57	**66935.16**
Rewrite	65162.2	67220.19	56718	66270.97	63774.93	**63829.26**
Read	64706.65	55113.43	59132.96	66049.08	67715.52	**62543.53**
Re-Read	66926.66	63344.71	54890.55	68055.16	55899.94	**61823.4**
Reverse Read	58837.69	57468.47	61732.02	52707.81	66910.52	**59531.3**
Random read	65468.09	66132.15	65605.46	50687.93	61796.64	**61938.05**
Random write	63371.95	56132.34	66610.79	66971.54	64995.07	**63616.34**
						62888.15

Record size = 16KB						
	1	2	3	4	5	평균
Initial write	124524.4	132296.3	110899.8	128865.6	109768.8	**121271**
Rewrite	138200.4	139984.5	117946.5	133541.6	121886.5	**130311.9**
Read	105697.2	129420.2	107378.3	87723.76	110944.7	**108232.8**
Re-Read	132293.3	127238.3	131790.5	96708.98	127991.5	**123204.5**
Reverse Read	120326	130071.3	121179.6	109458	134482.7	**123103.5**
Random read	114586.7	118486.1	103315.2	112848.3	120380.4	**113923.3**
Random write	123825.4	123295.4	114257.9	128404.6	122024.8	**122361.6**
						120344.1

Record size = 32KB						
	1	2	3	4	5	평균
Initial write	232220.1	197226.3	271864.2	250147.6	192797.2	**228851.1**
Rewrite	225452.5	170997.5	235207.2	221210.8	252008.4	**220975.3**
Read	239525.6	148558.8	167569.5	163789.9	232875	**190463.7**
Re-Read	254442.2	204561.8	173190.9	252058.9	199362.1	**216723.2**
Reverse Read	246203.6	263564.3	160684.7	231117.4	240524.8	**228419**
Random read	185421.4	209833.5	156112.8	183755.5	206930.9	**188410.8**
Random write	202111.1	202157.2	212967	216761.1	223504.3	**211500.1**
						212191.9

Record size = 64KB						
	1	2	3	4	5	평균
Initial write	514048.2	298449.1	510894.4	304657.1	512935.2	**428196.8**
Rewrite	328646.2	451223.4	468464.6	297407.4	286238.3	**366396**
Read	344152.5	448714.6	379924.2	295296.6	406328.3	**374883.2**
Re-Read	291346.1	325518.4	388986	294202.9	285580.3	**317126.7**
Reverse Read	292723.8	293909.5	426877.5	447840.5	293996.4	**351069.5**
Random read	337961.5	379824.6	300231.3	279340.7	380619.1	**335595.4**
Random write	351361.7	298439.2	427400.7	326263.6	297223.8	**340137.8**
						359057.9

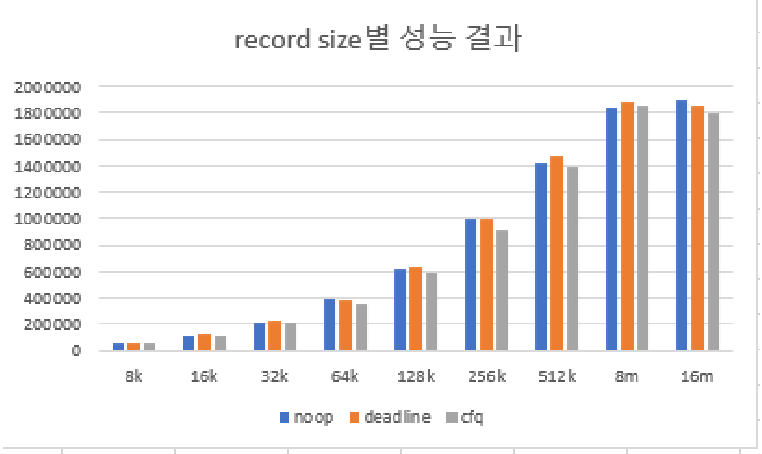
Record size = 128KB						
	1	2	3	4	5	평균
Initial write	648963.6	522853.6	547867.7	590855.2	573625.5	**576833.1**
Rewrite	630285.7	694037.4	599176.3	698634.3	633089.4	**651044.6**
Read	616706.8	553424.7	553564.7	556922.3	552745.2	**566672.7**
Re-Read	642724.8	547510.1	553202.8	552207.9	545950.5	**568319.2**
Reverse Read	592888.4	587759.3	563724.1	631037.4	603366.1	**595755.1**
Random read	533310	604080.7	530225.2	573883	651718.1	**578643.4**
Random write	524847.5	694396.4	622473.1	614086.8	543894.3	**599939.6**
						591029.7

Record size = 256KB						
	1	2	3	4	5	평균
Initial write	850659.6	917820.4	840975.4	1049612	886092.3	**909031.9**
Rewrite	960633.8	910608.1	832627.9	928117.4	864101.4	**899217.7**
Read	914803.4	920051.3	797792.4	1017172	955707.9	**921105.4**
Re-Read	1092144	1034037	988853.9	854682.6	837797.8	**961503.1**
Reverse Read	862769.1	1147786	1025122	920129.4	815747.1	**954310.8**
Random read	976485.9	930274.9	915431.8	825483.4	876931.7	**904921.5**
Random write	913238.6	913679.5	855811.3	796823.5	787429.6	**853396.5**
						914783.8

Record size = 512KB						
	1	2	3	4	5	평균
Initial write	1445001	1366823	1436303	1399559	1548136	**1439164**
Rewrite	1256291	1284374	1245932	1286208	1478102	**1310181**
Read	1318007	1322777	1356588	1304808	1542691	**1368974**
Re-Read	1329885	1506458	1378586	1369091	1460913	**1408987**
Reverse Read	1338031	1540623	1368829	1400432	1591385	**1447860**
Random read	1310532	1427745	1309780	1327865	1508347	**1376854**
Random write	1420617	1332740	1431167	1263147	1428648	**1375264**
						1389612
Record size = 8MB						
	1	2	3	4	5	평균
Initial write	1878697	1886091	1909592	1870363	1793648	**1867678**
Rewrite	1835656	1775391	1794293	1849768	1762633	**1803548**
Read	2051623	1840744	1835903	1814782	1939050	**1896420**
Re-Read	2076268	1795766	1813709	1815882	1824264	**1865178**
Reverse Read	1998064	1908338	1841497	1926627	1946237	**1924153**
Random read	1801151	1777010	1899271	1897333	1869760	**1848905**
Random write	1830276	1759536	1742918	1824727	1708391	**1773170**
						1854150
Record size = 16MB						
	1	2	3	4	5	평균
Initial write	1828030	1848470	1836470	1880429	1844925	**1847665**
Rewrite	1685318	1825070	1748280	1805160	1609235	**1734612**
Read	1608845	1659195	1961723	1587016	1886352	**1740626**
Re-Read	1657746	1728447	1949610	1914442	1782696	**1806588**
Reverse Read	1633362	1910197	1932439	1849304	1931243	**1851309**
Random read	1586404	1901955	1875844	1694529	1917792	**1795305**
Random write	1793327	1834982	1767970	1772196	1887615	**1811218**
						1798189

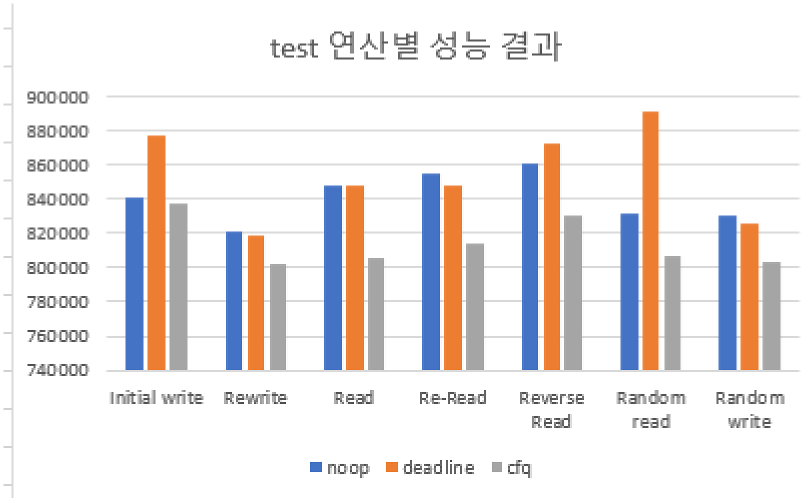
위 표를 그래프로 나타내어 비교하면 다음과 같다. 첫 번째 사진은 record size 별로 비교한것이며, 두 번째 사진은 테스트 연산 별로 비교해본 것이다. 평균값을 바탕으로 그래프를 구성하였다.

size	8k	16k	32k	64k	128k	256k	512k	8m	16m
noop	63616.39	120975.5	208216	401078	613307	996490.2	1418986	1847077	1898769
deadline	62391.16	121407.4	220521.8	374291.9	632258.7	991848.7	1472321	1880016	1858905
cfq	62888.15	120344.1	212191.9	359057.9	591029.7	914783.8	1389612	1854150	1798189



우선 record size 가 클수록 성능이 좋게 나오는 것을 볼 수 있다. 하지만 record size 가 8m, 16m 인 경우 큰 차이를 보이진 않으며 8m 인 경우가 더 좋게 나오는 경우도 존재하는 것을 확인할 수 있다. 3 개의 스케줄러가 모두 같은 추세를 보인다.

	noop	deadline	cfq
Initial write	841121.6	877441.2	836744.4
Rewrite	821038.5	818280.4	801686.1
Read	847252.8	848260.1	805273.3
Re-Read	855304.4	847456	814429.7
Reverse Read	860192.8	872812.4	830280.3
Random read	830966.8	891179	806636.4
Random write	830523.2	825727	803043.6



다음으로 test 연산 별 성능을 비교해본 결과이다. 전체적으로 deadline 스케줄러의 성능이 좋게 나왔으며, cfq 의 성능이 다른 것들에 비해 낮게 나왔다. 실제로 linux 에서 test 를 진행하면서도 cfq 부분은 시간이 다른 스케줄러보다 오래 걸렸던 것 같다. 우선 noop 스케줄러의 경우 random access 하는 device 를 위한 스케줄러이기 때문에 random read 연산이 다른 read 연산보다 더 좋은 성능이 나올 것이라고 예상했지만 꼭 그렇지만은 않았다. 애초에 noop 스케줄러는 I/O request 를 큐에 쌓아 두고 처리하기 때문에 reverse read 연산의 경우 성능이 낮게 나올 것이라 예상했지만 해당 연산을 하는 횟수가 거의 없기 때문에 위와 같은 결과가 나왔다고 예상한다. Deadline 스케줄러의 경우 읽기 우선 정책을 사용하기 때문에 read 의 성능이 보다 좋게 나올 것이라 예상하였지만 이 부분 역시 항상 같은 결과가 나오지 않았다. 마지막으로 cfq 스케줄러의 경우 모든 연산에서 가장 좋지 않은 성능을 보이는 것을 확인할 수 있었다. 이는 cfq 스케줄러에 비해 다른 스케줄러가 성능적으로 더 우수한 경우를 test 했기 때문이라고 생각한다.

3. 고찰

마지막 과제를 진행하면서 내가 생각했던 것과 매우 다른 결과가 나와 실험이 제대로 된 것이 맞는지 여러 번 확인하게 됐던 것 같다. 이론적으로 인지하고 있는 부분과 실제 실험 결과 값의 차이가 좀 있었던 것 같다. 마지막 결과 사진에서 볼 수 있듯이 cfq 스케줄러의 경우 어느 연산에서도 우수한 결과를 보이지 못했다. 아마 상황에 따라 최적의 성능을 낼 수 있는 적합한 스케줄러가 cfq 가 아니었기 때문에 성능차이가 있었다고 예상하는데, 그때그때 알맞은 스케줄러를 고려해서 사용하면 좋을 것 같다. 또한 매 실험 전 캐시 및 버퍼를 지우는 과정을 하였음에도 불구하고 완전히 동일한 환경이 만들어지진 않았을 것이라고 예상한다. 실험을 돌릴 때 마다 다르게 실행되는 백그라운드 프로세스 등으로 인해 확실하고 완전한 결과를 보지 못한 점이 아쉬움으로 남는다. 지난 학기 컴퓨터구조 수업 때 역시 벤치마크와 관련된 과제를 진행하였는데 이번 기회를 통해 다시 한 번 공부할 수 있어 좋았으며 운영체제 강의로만 배웠던 부분을 실습을 통해 직접 진행해볼 수 있어 좋았다.

4. Reference

- 운영체제실습 강의자료 참조