Performance-optimised computing – Week 5.

Processes, threads, scheduling - continued

Storage Drives and File IO

Dr. Bakay Árpád – Ericcson

Data Storage Technologies - Compare Performance

- There exist **Local attached** and **Remote** (network accessed) devices
- Major device types: **HDD vs. SSD**

	IOPS (@4K, Q16)	Seek time ms	Random trasfer ms	Seq transfer rate MB/s
HDD	1000	5-10	10-15	100-200
SSD	100-200k	-	0.04 – 0.08	4000-8000

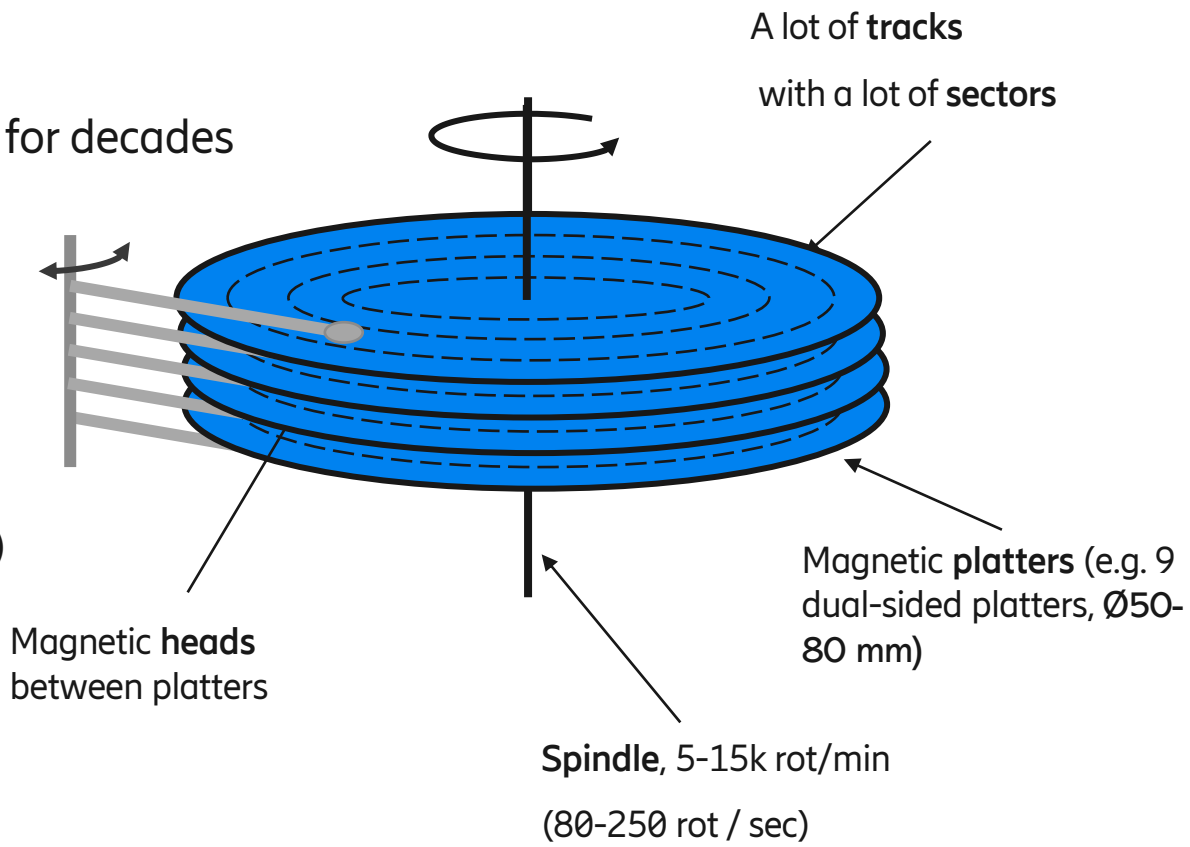
  - Plus: SSD uses less power, generates no noise, tolerant for shock and temperature
- **IOPS – I/O operations / sec,** specified for a given transfer size (typ: 4k), and queue length (1..16...256).
  - Strange/suspicious IOPS values are often published for SSD-s, *better to measure yourself!!*
- **Bus transfer rates**: (SCSI1: 5 Mbps), **SATA3**: 600MBps (dedicated), SAS: 2.4 GBps, Fibre Channel: up to 18 GByte/sec, I-SCSI up to 10 GBps,
- **NVMe SSD**-s use PCI Express bus, which is up to 15GByte/sec/lane (typical SSD transfer speed is 4-8GBps)
- **NVMe over Fabric**: detach SSD drives from host (PCI-E bus), but keep speed (e.g. 8GBps).

Hard drive – Physical Structure

Sector size is 512 bytes – this has not changed for decades

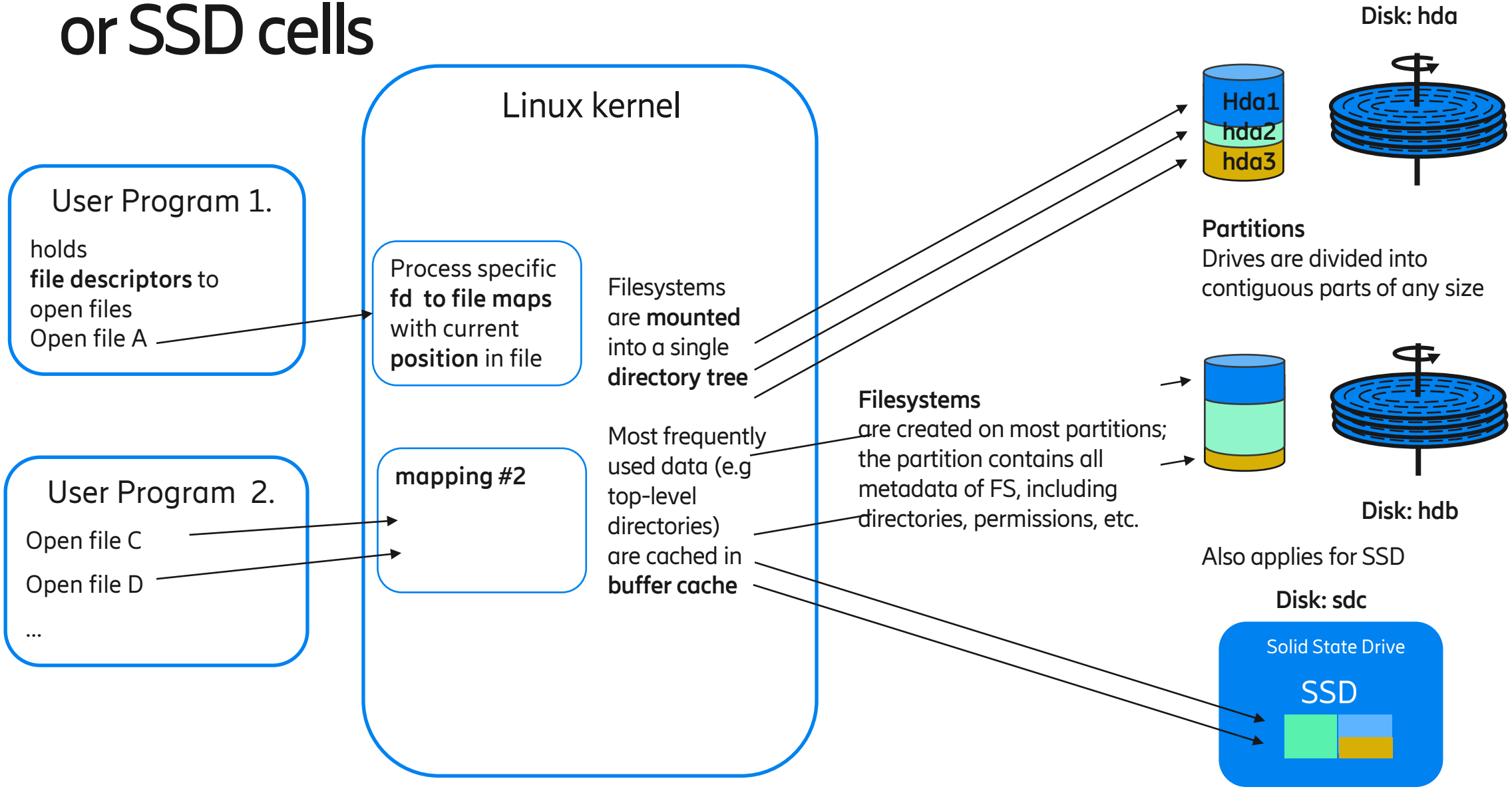
Up to 1.3 TBytes per platter side ->
2.6 * 10^9 sectors ->
e.g.:

- 200 000 tracks!!! (about 100 nm wide)
- 13 000 sectors / track!!! (about 15 µm long)

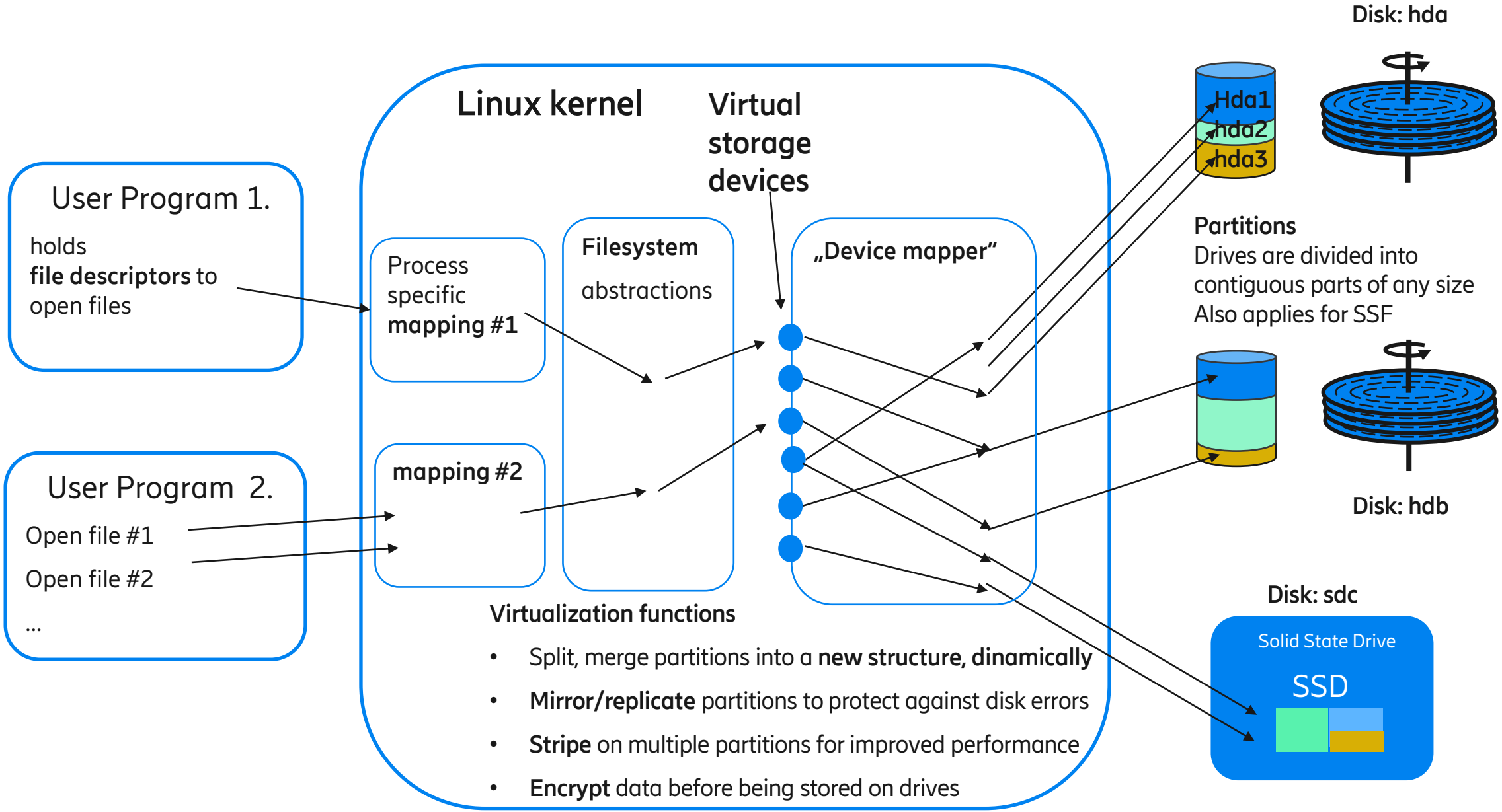


- No similar drawing for SSD-s, everything is in silicon ☹

From userspace files to magnetic traces on disk or SSD cells



Storage Virtualization



Standard File Operations in Unix, Linux, Posix

- Basic, low level, syscall operations using „file descriptor" (integer)
  - int fd = **open**(<pathname>, <flags>, [<create_mode>]);
  - **read**(fd, <buffer>, <count>) / **write**(fd, <buffer>, <count>) / **lseek**(fd)
  - **close**(fd)
  - fd is process-specific (unique within a process)
  - Example: https://gitlab.inf.elte.hu/-/snippets/49
- „POSIX Standard" file ops (implemented by library which uses above file descriptors)
  - fopen(), fread()/fwrite(), fclose() + fflush(), fprintf()/fscanf(), fgetc()/fputc(), fdopen()
  - Differences:
    - **struct FILE*** reresents an open file
      - It is possible to convert either way FILE <-> fd
    - buffered, formatted input/output,
    - Implemented with userspace library (libc) functions (vs. system calls)
- **Orthogonality**: an fd may represent not only files, but also other **„devices"** and **network connections**
  - FILE * stdin / fd 0, FILE* stdout / fd: 1, FILE* stderr / fd: 2 (no need to open/close)
  - Also devices in the **/dev** directory: **console, ptsX**, block devices: **sdX, sdXY**, io ports: **ttyX**, **random/null/zero** etc.

Async IO model for improved performance

- Synchronous: thread is blocked until IO is completed.
- Async IO: submit requests and reap the results to/from an „IO context"

```
int io_setup(int maxevents, io_context_t *ctxp);
...
int io_submit(io_context_t ctx, long nr, struct iocb *ios[]);
...
int io_getevents(io_context_t ctx_id, long min_nr, long nr, struct
io_event *events, struct timespec *timeout);
...
int io_destroy(io_context_t ctx);
```

Async IO Request Buffer

```
struct iocb {
    void *data;
    short aio_lio_opcode;  // IO_CMD_PREAD IO_CMD_PWRITE
    int aio_fildes;   // = fd

    union {
        struct {
            void *buf;
            unsigned long nbytes;
            long long offset;
        } c;
    } u;
};
```

result structure

```
struct io_event {
    void *data;
    struct iocb *obj;
    long long res;
};
```

Storage performance testing with fio

- Install fio (if not already installed)
- Create config file, also with parallel workloads (multiple jobs).
  - Select ioengine: sync (traditional IO) / libaio (async IO) / etc-
  - Select read/write method: read/write randread/randwrite
  - Specify run time
  - Specify target file (or device names + size of tested area)
  - Specify parallelism
  - **Note** Buffer caches (see next slide) may bias the results
    -> use invalidate=1
- Results printed:
  - throughput (MB/s)
  - Total I/O (Gbytes)
  - IOPS -> not directly printed, calculate IOS / jobs / time

fio_test.conf

```
[global]
bs=4K
iodepth=256
direct=1
ioengine=sync
group_reporting
time_based
runtime=10
numjobs=4
name=raw-randread
rw=write

[job1]
filename=testfile.bin
```

Meet the Disk Buffer Cache!

- Linux uses almost all free memory as a cache for data on disks
  - When processes request memory, the buffer cache is reduced
- Buffer cache speeds up storage reads and writes significantly
- It caches fixed size „disk blocks“ (i.e. allocation units), which are typically 4 kbytes (8 sectors)
  - `stat -f /  -> ... block size: 4096 ...`
- Buffer Cache contains „clean“ and „dirty blocks“, the latter ones still need to be written to storage
  - This speeds up writes, but data may be lost before persisted
  - `sync` command or `fsync()` system will save all dirty buffers

- This may be misleading for fio and other performance measurement tools!
  - To clean all buffer caches, use: `echo 1 > /proc/sys/vm/drop_caches`

Compute/ Storage Performance Monitoring with iostat

- iostat -c : show CPU stats
  - %user, %system, %iowait, %idle

- iostat –d [<device>]
  - transactions / sec
  - read /s write /s,
  - discards / s
  - kBytes total

- Use <interval> for periodic display

Excercises:

1. Use fio and iostat for disk performance testing on Linux

- Install fio (if not already installed)
- Create a fio config file
- Run fio
- Use iostat –d to check storage device usage

2. Homegrown program for testing Disk/File IO

- https://gitlab.inf.elte.hu/-/snippets/49
- Build with gcc: **gcc –O1 –o disk_io_test disk_io_test.c**
- Run:
  - Write test: ./disk_io_test –w –f my_testfile –s 100000000 expect 1-1.5 Gb/s (without sync)
  - Read test: ./disk_io_test –f my_testfile –s 100000000 expect 2-3 Gb/s (from cache, 0.1-0.15 from disk
  - Use –y option for sync after every 100th reads/writes
  - Use –f MEMORY to compare with plain memory reads/writes expect 20-25 Gb/s for reads, 3-3.5 Gb/s for writes

That's it for today!