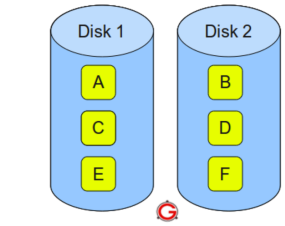
Persistence

- 36 - I/O Devices
- 37 - Hard Disk Drives
- 38 - RAID
- 39 - File and Directories

RAID

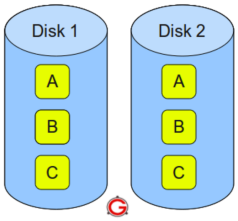
- Redundant array of independent disks
- Goal:
  - Disks are very cheap
  - Failures are very costly
  - Use “extra” disks to ensure reliability
    - If one disk goes down, the data still survives
  - Also allows faster access to data
- Many raid “levels”
  - Different reliability and performance properties

Redundant Array Independent Disks



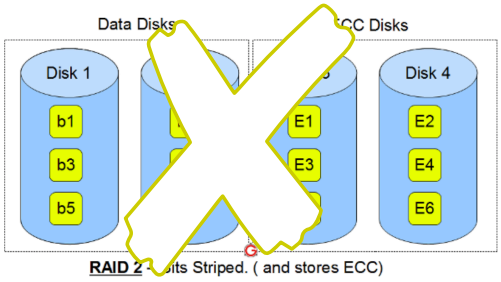
RAID 0 – Blocks Striped. No Mirror. No Parity.

Fast!



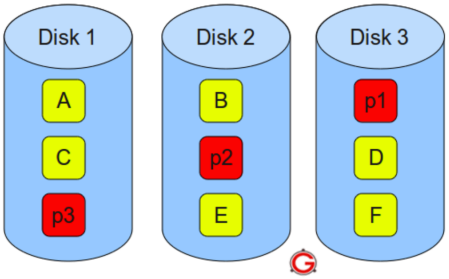
RAID 1 – Blocks Mirrored. No Stripe. No parity.

Redundant!



RAID 2 - its Striped. (and stores ECC)

Weird!

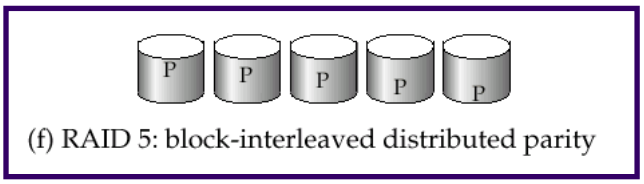


RAID 5 – Blocks Striped. Distributed Parity.

from thegeekstuff.com

RAID Level 5

- Distributed parity "blocks" instead of bits
- Normal operation:
  - "Read" directly from single disk.
    - Load distributed across all 5 disks
  - "Write": Need to read and update the parity block
    - To update 9 to 9'
      - read 9 and P2
      - compute P2' = P2 *xor* 9 *xor* 9'
      - write 9' and P2'

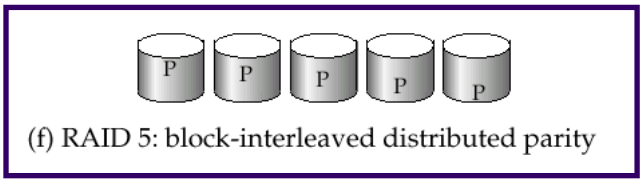


(f) RAID 5: block-interleaved distributed parity

P0	0	1	2	3
4	P1	5	6	7
8	9	P2	10	11
12	13	14	P3	15
16	17	18	19	P4

RAID Level 5

- Failure operation (disk 3 has failed)
  - "Read block 0": Read it directly from disk 2
  - "Read block 1" (which is on disk 3)
    - Read P0, 0, 2, 3 and compute 1 = P0 *xor* 0 *xor* 2 *xor* 3
  - "Write":
    - To update 9 to 9'
      - read 9 and P2
        - Oh… P2 is on disk 3
        - So no need to read or update it
      - Write 9'



(f) RAID 5: block-interleaved distributed parity

P0	0		2	3
4	P1		6	7
8	9		10	11
12	13		P3	15
16	17		19	P4

Choosing a RAID level

- RAID 0 striping fastest, but no fault tolerance
- Main choice between RAID 1 and RAID 5
- Level 1 better write performance than level 5
  - Level 5: 2 block reads and 2 block writes to write a single block
  - Level 1: only requires 2 block writes
  - Level 1 preferred for high update environments such as log disks
- Level 5 lower storage cost
  - Usable storage for Level 1 only 50% of raw disk capacity
  - Level 5 is preferred for applications with low update rate, and large amounts of data

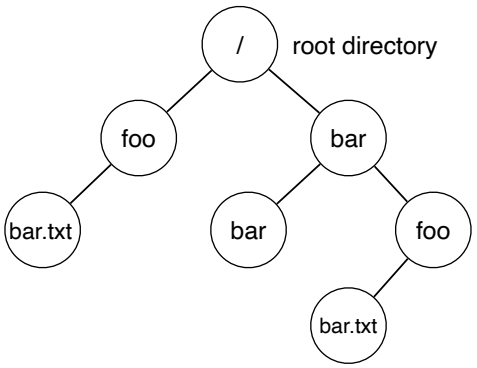
Persistence

- 36 - I/O Devices
- 37 - Hard Disk Drives
- 38 - RAID
- 39 - File and Directories

Persistence

- Keep data intact even if power loss
  - hard drive
  - solid-state device
- Two key abstractions
  - File
    - linear array of bytes
    - each has *inode number (inumber)*
  - Directory
    - stored like a file, has inumber
    - contains list of (user-readable name, low-level name) pairs
    - each entry refers to other file or directory.
- Assume a directory entry (“foo”, 10)
  - File “foo” w/ inode number (“inumber”) 10

Directory Hierarchy



Example Directory Tree

Valid files (absolute pathname) :
/foo/bar.txt
/bar/foo/bar.txt

Valid directory :
/
/foo
/bar
/bar/bar
/bar/foo/

Sub-directories

Creating Files

- Use `open()` system call with `O_CREAT` flag

```
int fd = open("foo", O_CREAT | O_WRONLY | O_TRUNC);
```

  - `O_CREAT` : create file
  - `O_WRONLY` : only write to that file while opened
  - `O_TRUNC` : truncate file (remove existing content)

- `open()` system call returns *file descriptor*
  - *File descriptor* is an integer, and is used to access files

Reading and Writing Files

- Reading and writing '`foo`':

```
prompt> echo hello > foo
prompt> cat foo
hello
prompt>
```

  - `echo` : redirect output to `foo`
  - `cat` : dump contents of file to screen

Reading and writing files…

```
prompt> strace cat foo
…
open("foo", O_RDONLY|O_LARGEFILE)  = 3
read(3, "hello\n", 4096)           = 6
write(1, "hello\n", 6)             = 6 // file descriptor 1: standard out
hello
read(3, "", 4096)                  = 0 // 0: no bytes left in the file
close(3)                           = 0
…
prompt>
```

- `open` (file descriptor, flags)
  - return file descriptor ("3" in example)
  - descriptors 0, 1, 2 are standard input, output, and error
- `read` (file descriptor, buffer pointer, buffer size)
  - returns number bytes read
- `write` (file descriptor, bufer pointer, buffer size)
  - return number of bytes written

Non-sequential Reading and Writing

- open files have *current offset*
  - specifies byte position where next read or write will begin
- updated
  - implicitly: a read of `N` bytes adds `N` to current offset
  - explicitly: `lseek()`

```
off_t lseek(int fildes, off_t offset, int whence);
```

  - `fildes` : file descriptor
  - `offset` : location in file
  - whence : how the seek is performed

From the man page:

```
If whence is SEEK_SET, the offset is set to offset bytes.
If whence is SEEK_CUR, the offset is set to its current
location plus offset bytes.
If whence is SEEK_END, the offset is set to the size of the
file plus offset bytes.
```


`fsync()`

- file system *buffers* writes in memory
  - overwrites
  - write combining
  - push
  - *data can be lost*
- some apps require more timely guarantees
  - DBs….
  - `fsync()` returns once the data has made it's way to the disk

```
int fd = open("foo", O_CREAT | O_WRONLY | O_TRUNC);
assert (fd > -1)
int rc = write(fd, buffer, size);
assert (rc == size);
rc = fsync(fd);
assert (rc == 0);
```

Other FS APIs

- `rename (char *old, char *new)` (*atomic*)

```
prompt> mv foo bar        // mv uses the system call rename()
```

```
int fint fd = open("foo.txt.tmp", O_WRONLY|O_CREAT|O_TRUNC);
write(fd, buffer, size); // write out new version of file
fsync(fd);
close(fd);
rename("foo.txt.tmp", "foo.txt");
```

- `stat()`, `fstat()` : show metadata

```
struct stat {
    dev_t st_dev;         /* ID of device containing file */
    ino_t st_ino;         /* inode number */
    mode_t st_mode;       /* protection */
    nlink_t st_nlink;     /* number of hard links */
    uid_t st_uid;         /* user ID of owner */
    gid_t st_gid;         /* group ID of owner */
    dev_t st_rdev;        /* device ID (if special file) */
    off_t st_size;        /* total size, in bytes */
    blksize_t st_blksize; /* blocksize for filesystem I/O */
    blkcnt_t st_blocks;   /* number of blocks allocated */
    time_t st_atime;      /* time of last access */
    time_t st_mtime;      /* time of last modification */
    time_t st_ctime;      /* time of last status change */
};
```


Stat, cont.

```
prompt> echo hello > file
prompt> stat file

File: 'file'
Size: 6 Blocks: 8 IO Block: 4096 regular file
Device: 811h/2065d Inode: 67158084 Links: 1
Access: (0640/-rw-r-----) Uid: (30686/ root) Gid: (30686/ remzi)
Access: 2011-05-03 15:50:20.157594748 -0500
Modify: 2011-05-03 15:50:20.157594748 -0500
Change: 2011-05-03 15:50:20.157594748 -0500
```

- info kept in `inode`

Removing files

- `rm` calls `unlink()`

```
prompt> strace rm foo
…
unlink("foo")              = 0      // return 0 upon success
…
prompt>
```

- `link(old *char, *new)    // "hard" link`
  - adds a new hard path to get to same file
  - inode tracks number of links, deletes inode when zero

Symbolic ("soft") links

- implemented as third type of file/dir
  - file contains a static path to redirect accesses
  - can link directories (hard links cannot)
  - inode is not updated, so symbolic links can be *orphaned*

Links cont.

```
hyperion:~/x> echo "hello" > file
hyperion:~/x> ln -s file file2
hyperion:~/x> ln file file3
hyperion:~/x> ll
total 16K
drwxr-xr-x  2 keleher keleher 4.0K Mar 26 10:24 .
drwxr-xr-x 23 keleher keleher 4.0K Mar 26 10:23 ..
-rw-r--r--  2 keleher keleher    6 Mar 26 10:23 file
lrwxrwxrwx  1 keleher keleher    4 Mar 26 10:24 file2 -> file
-rw-r--r--  2 keleher keleher    6 Mar 26 10:23 file3
hyperion:~/x> stat file
  File: file
  Size: 6          Blocks: 8          IO Block: 4096   regular file
Device: 802h/2050d Inode: 159776897   Links: 2
Access: (0644/-rw-r--r--)  Uid: ( 1000/ keleher)   Gid: ( 1000/ keleher)
Access: 2024-03-26 10:23:57.714939940 -0400
Modify: 2024-03-26 10:23:57.714939940 -0400
Change: 2024-03-26 10:24:19.990594144 -0400
 Birth: 2024-03-26 10:23:57.714939940 -0400
hyperion:~/x> mv file nfile
hyperion:~/x> cat nfile
hello
hyperion:~/x> cat file2
cat: file2: No such file or directory
hyperion:~/x> cat file3
hello
hyperion:~/x>
```


Making and mounting file systems

- `mkfs` : make empty file system
  - creates a root directory on given partition
- `mount()`
  - maps file system on top of an existing directory

```
prompt> mount -t ext3 /dev/sda1 /home/users
prompt> ls /home/users
a b
```

Linux machine

```
hyperion:~> mount
sysfs on /sys type sysfs (rw,nosuid,nodev,noexec,relatime)
proc on /proc type proc (rw,nosuid,nodev,noexec,relatime)
udev on /dev type devtmpfs (rw,nosuid,relatime,size=8087356k,nr_inodes=2021839,mode=755,inode64)
devpts on /dev/pts type devpts (rw,nosuid,noexec,relatime,gid=5,mode=620,ptmxmode=000)
tmpfs on /run type tmpfs (rw,nosuid,nodev,noexec,relatime,size=1629064k,mode=755,inode64)
/dev/sda2 on / type ext4 (rw,relatime,errors=remount-ro)
securityfs on /sys/kernel/security type securityfs (rw,nosuid,nodev,noexec,relatime)
tmpfs on /dev/shm type tmpfs (rw,nosuid,nodev,inode64)
tmpfs on /run/lock type tmpfs (rw,nosuid,nodev,noexec,relatime,size=5120k,inode64)
cgroup2 on /sys/fs/cgroup type cgroup2 (rw,nosuid,nodev,noexec,relatime)
pstore on /sys/fs/pstore type pstore (rw,nosuid,nodev,noexec,relatime)
bpf on /sys/fs/bpf type bpf (rw,nosuid,nodev,noexec,relatime,mode=700)
systemd-1 on /proc/sys/fs/binfmt_misc type autofs (rw,relatime,fd=29,pgrp=1,timeout=0,minproto=5,maxproto=5,direct,pipe_ino=21621
mqueue on /dev/mqueue type mqueue (rw,nosuid,nodev,noexec,relatime)
debugfs on /sys/kernel/debug type debugfs (rw,nosuid,nodev,noexec,relatime)
hugetlbfs on /dev/hugepages type hugetlbfs (rw,relatime,pagesize=2M)
tracefs on /sys/kernel/tracing type tracefs (rw,nosuid,nodev,noexec,relatime)
fusectl on /sys/fs/fuse/connections type fusectl (rw,nosuid,nodev,noexec,relatime)
configfs on /sys/kernel/config type configfs (rw,nosuid,nodev,noexec,relatime)
none on /run/credentials/systemd-sysusers.service type ramfs (ro,nosuid,nodev,noexec,relatime,mode=700)
tracefs on /sys/kernel/debug/tracing type tracefs (rw,nosuid,nodev,noexec,relatime)
/var/lib/snapd/snaps/core20_2182.snap on /snap/core20/2182 type squashfs (ro,nodev,relatime,errors=continue,x-gdu.hide)
/var/lib/snapd/snaps/snapd_20671.snap on /snap/snapd/20671 type squashfs (ro,nodev,relatime,errors=continue,x-gdu.hide)
/var/lib/snapd/snaps/lxd_24061.snap on /snap/lxd/24061 type squashfs (ro,nodev,relatime,errors=continue,x-gdu.hide)
/var/lib/snapd/snaps/core20_2105.snap on /snap/core20/2105 type squashfs (ro,nodev,relatime,errors=continue,x-gdu.hide)
/dev/sdb1 on /media/littlessd type ext4 (rw,relatime,stripe=256)
/dev/sdc1 on /media/bigssd type ext4 (rw,relatime,stripe=512)
binfmt_misc on /proc/sys/fs/binfmt_misc type binfmt_misc (rw,nosuid,nodev,noexec,relatime)
lxcfs on /var/lib/lxcfs type fuse.lxcfs (rw,nosuid,nodev,relatime,user_id=0,group_id=0,allow_other)
tmpfs on /run/snapd/ns type tmpfs (rw,nosuid,nodev,noexec,relatime,size=1629064k,mode=755,inode64)
nsfs on /run/snapd/ns/lxd.mnt type nsfs (rw)
/var/lib/snapd/snaps/snapd_21184.snap on /snap/snapd/21184 type squashfs (ro,nodev,relatime,errors=continue,x-gdu.hide)
tmpfs on /run/user/1000 type tmpfs (rw,nosuid,nodev,relatime,size=1629060k,nr_inodes=407265,mode=700,uid=1000,gid=1000,inode64)
hyperion:~>
```