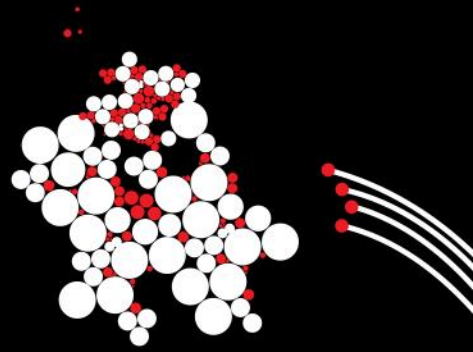
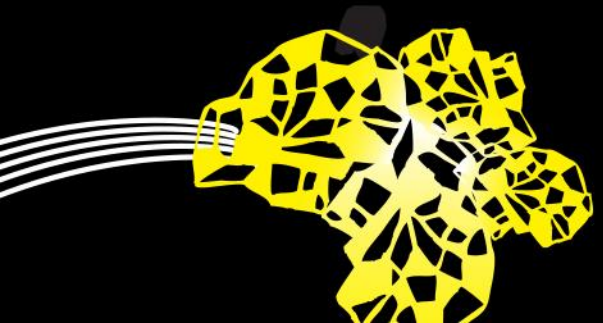
UNIVERSITY OF TWENTE.



OPERATING SYSTEMS

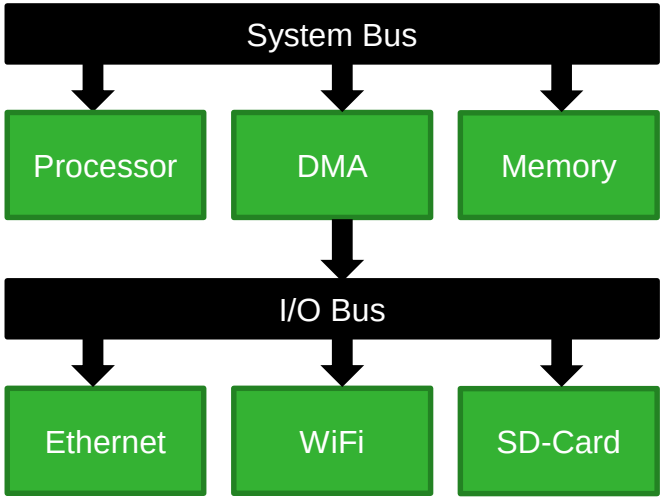
INPUT/OUTPUT

ERIK TEWS <e.tews@utwente.nl>



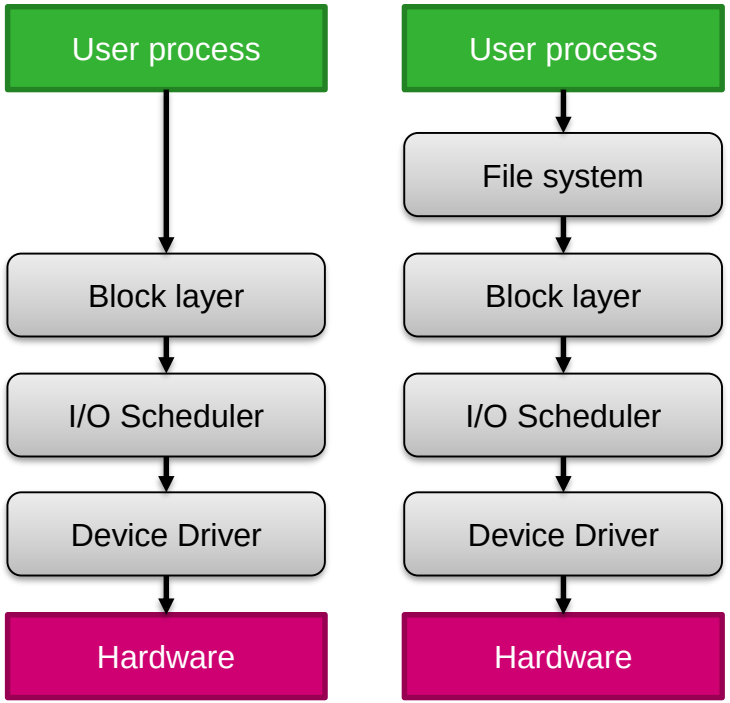
DIRECT MEMORY ACCESS (DMA)

- CPU sends commands to controller indicating:
  - what to do and
  - where in memory to get/put the data
- Controller runs in parallel with CPU
  - Problem : Contention for system bus cycles
  - Solution : I/O processor or I/O bus



DESIGN ISSUES

- Efficiency
  - Buffering (how?)
  - Scheduling
- Generality
  - Layering
  - Uniform access – Unix!



FILE SYSTEMS

The most important job of UNIX is to provide a file system. …

The system calls to do I/O are designed to eliminate the differences between the various devices and styles of access. There is no distinction between "random" and sequential I/O, nor is any logical record size imposed by the system. The size of an ordinary file is determined by the highest byte written on it; no predetermination of the size of a file is necessary or possible.

LINUX IS UNIFIED

- Block devices for fast peripherals
- Character devices for slow peripherals
- Devices are handled as special files:
  - df -l
  - ls –l /dev/tty /dev/sda
  - cat /dev/tty >junk

SPARSE FILES

```
#define M 1024*1024 /* One MByte */

int main(int argc, char *argv[]) {
    int out = open(argv[1],
        O_RDWR|O_CREAT|O_TRUNC, 0666);
    int n = atoi(argv[2]);
    lseek(out, n*M-1, SEEK_SET);
    write(out, "\0", 1);
    close(out);
    return 0;
}
```

```
gcc Sparse.c
for d
in    .    /tmp
do
    for n
    in 1 2 4 8 16 32 64 128 256 512 1024
    do
        time    ./a.out    $d/$n.dat    $n
        time    cat    $d/$n.dat >  $d/junk
    done
done
ls -l    /tmp/*dat    /tmp.junk
```

HARD DISKS

- Disk with rotating platter and moving arm (alternative?)
- Timing
  - Access time depends on the seek time + rotational delay
  - Transfer time depends on the rotational speed and the channel bandwidth
  - Example:

HARD DISK SCHEDULERS

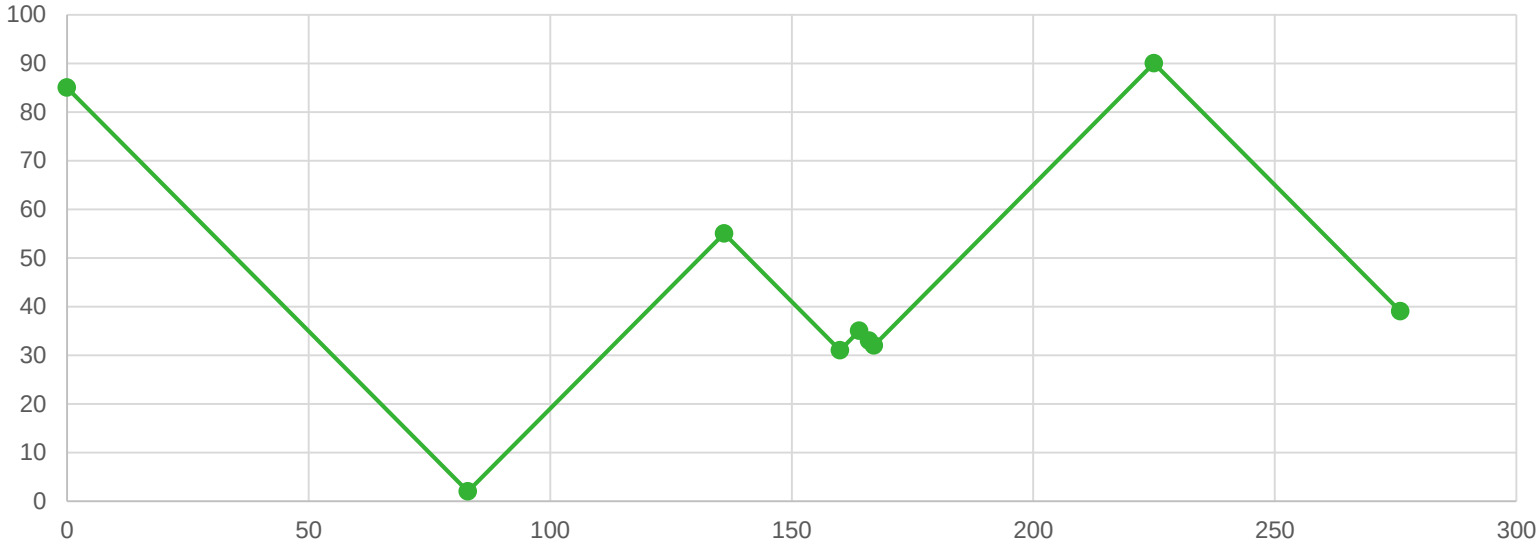
- FIFO
  - Fist In First Out
- SSTF
  - Shortest Seek Time First
- SCAN
  - As SSTF, but avoid changes to the direction as long as possible

EXAMPLE

Request queue:
85 2 55 31 35 33 32 90 39

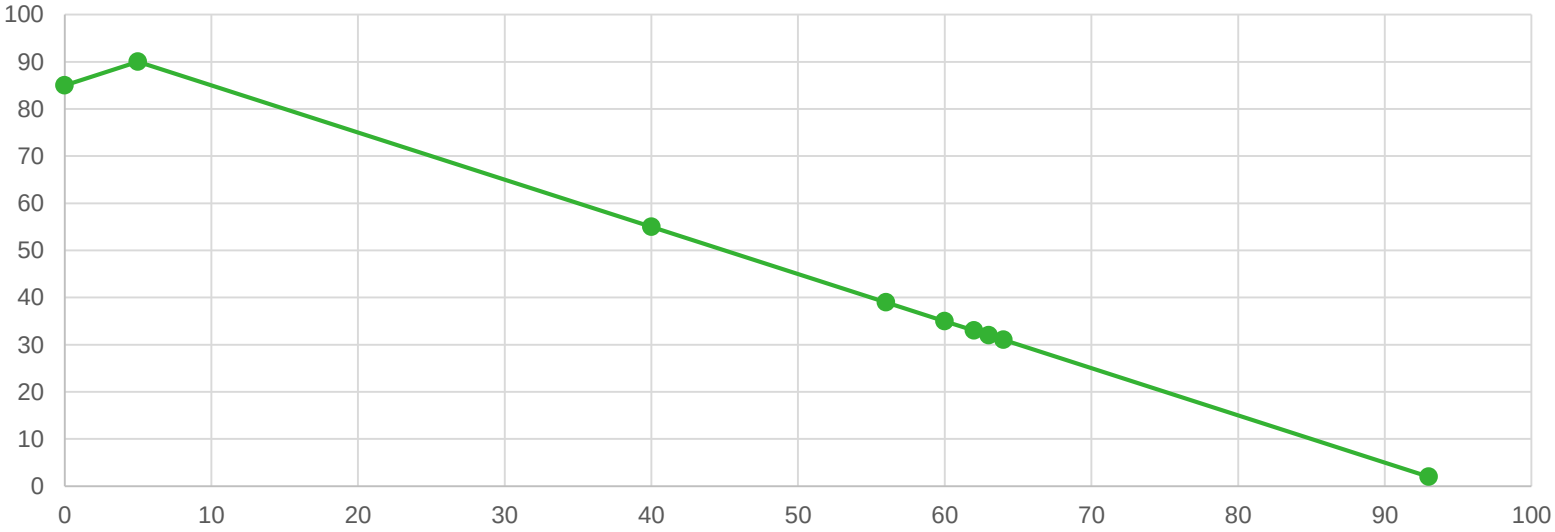
FIFO

REQUEST QUEUE: 85 2 55 31 35 33 32 90 39



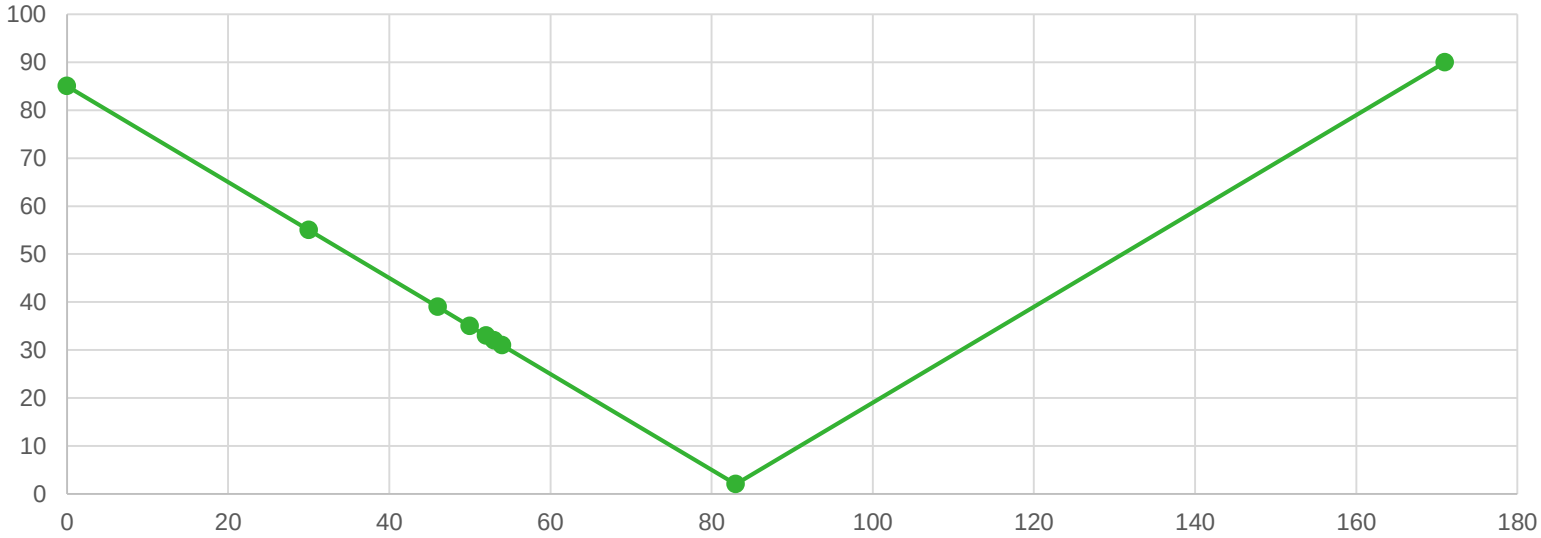
SSTF

REQUEST QUEUE: 85 2 55 31 35 33 32 90 39



SCAN

REQUEST QUEUE: 85 2 55 31 35 33 32 90 39



SSD

- The future
- No rotation anymore
- Wear levelling
- Variable block sizes
- Transfers in large blocks preferred

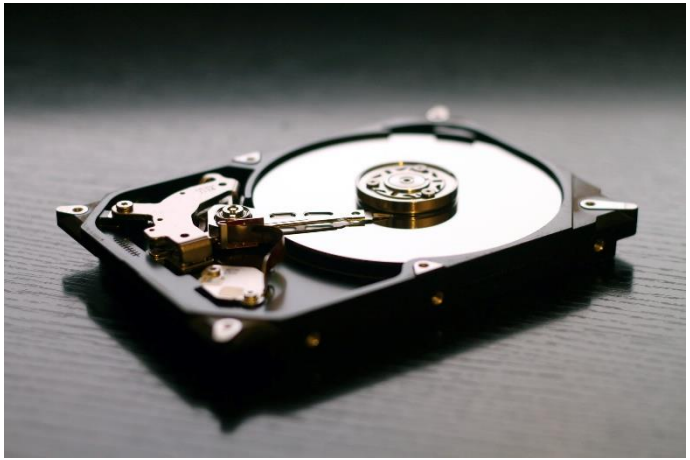
SHINGLED MAGNETIC RECORDING (SMR)

- The future of hard disks
- More capacity
- Read behaviour mostly the same
- Writes only work in large chunks
- Writes are also small
- Traditional schedulers do not work

HYBRID DRIVES

- Shingled Magnetic Recording is slow
  - Most drives have a cache/fast section using traditional technology
- SSDs are expensive and hard disks are slow
  - Hybrid drives exist
  - Traditional large hard disk
  - Very small SSD that is used as a cache

CURRENT DEVELOPMENT OF SCHEDULERS

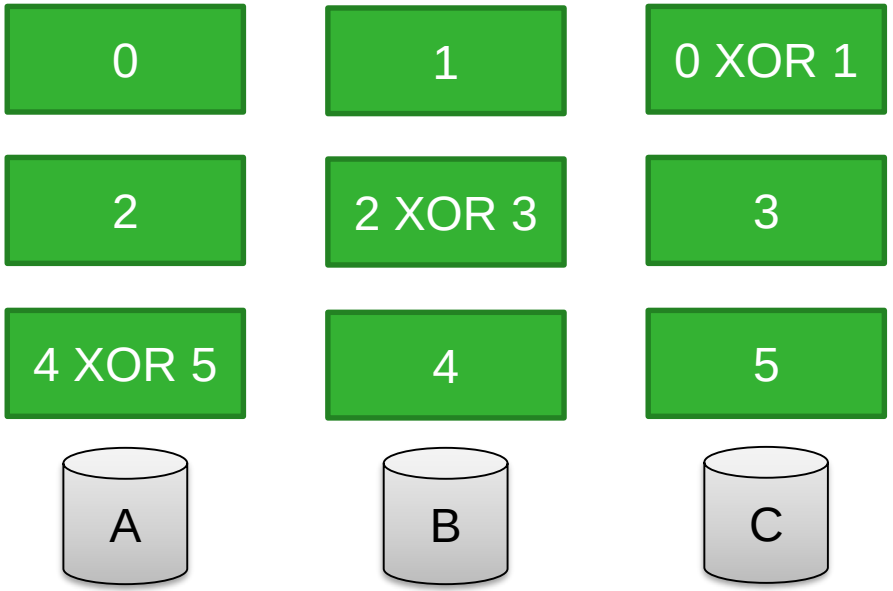


WHAT LINUX IS CURRENTLY USING

- Disks support Native Command Queuing (limited depth)
- Linux has multiple I/O schedulers to take requests from processes (request queue) and compile them into requests for the device driver (dispatch queue)
- For harddisks, CFQ is useful
  - Tries to give each process a fair share of the I/O bandwidth
- For SSDs, NOOP seems to be faster
  - Just focuses on merging I/O requests to larger requests

RAID

- Several physical drives seen as one logical drive
- Data is striped across the disks (why?)
- Redundancy (why?)



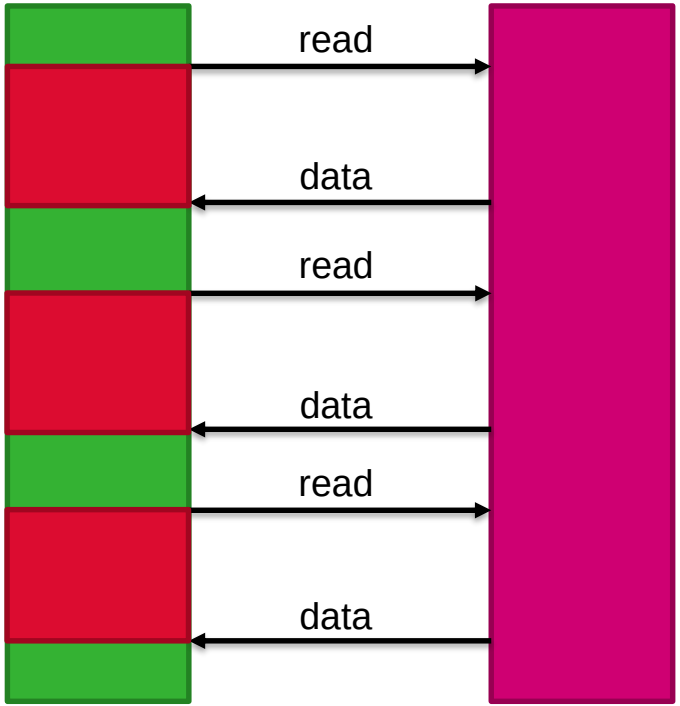
DISK CACHE

- man free
- gcc Fadvise.c

Command	Free	Time
free -m	793	
Fadvise Foo		34s
free –m	284	
cat Foo > /dev/null		0.34s
free –m	284	
Fadvise Bar x		42s
free –m	692	
cat Bar > /dev/null		11s
cat Bar > /dev/null		2s
Free –m	206	

```
#define M 1024*256
#define N 1000 /* number of buffers written */
int main(int argc, char *argv[]) {
    int out = open(argv[1],
        O_RDWR|O_CREAT|O_TRUNC, 0666);
    int i,k;
    int buf[M]; /* One Mbyte */
    for(i=0;i<N;i++) {
        for(k=0;k<M;k++) {
            buf[k] = i*N+k;
        }
        write(out,buf,sizeof(buf));
    }
    if(argc>=3) {
        fdatasync(out);
        posix_fadvise(out,0,0,
            POSIX_FADV_DONTNEED);
    }
    close(out);
    return 0;
}
```

SYNC VS. ASYNC



SUMMARY

- Without I/O no communication with the outside world
- Caching crucial for performance
- Disks are slower than the CPU hence more scope for scheduling
- RAID for more dependable storage