Implementation lecture 2

IO requests

Installing devices

Dealing with disks - Ch 12

More on device tables

Some systems provide multilevel device tables.

e.g. All printers do similar things, therefore specific printer drivers can be called from a shared printer interface.

Other examples:

- displays
- keyboards
- modems
- disk drives
- sound systems

This way less code needs to be provided by the manufacturers of the device.

In some cases none at all.

IO requests

A link has been made between the process's file information block and the corresponding entry in the device table.

When we get an IO request (we will refer to this as a **doio** call)

it includes what sort of access (data transfer, control or status)

after checking (can this device do that? are the parameters ok?)

may need to connect it to the list of requests for that device

(we will refer to these as IORB - IO request blocks)

may also check access rights of the caller

could have been done at Open time.

Not necessarily performed in FIFO order.

eventually the request is handled

this uses the *upper half* of the device handling routines

initiate IO

may block waiting for the results

or may return to the caller

errors or results are passed back

What about the *lower half*

The bit started when the device has done its job and the output buffer can be used again or the input buffer is now full.

Interrupt driven

Interrupt handler restarts the *top half*.

I am now ready for the next character.

Your data is in the buffer.

Also has to have error handling code.

What sort of things can go wrong?

- no paper
- CRC error
- No such sector
- bad phone connection
- device not responding

Sometimes the error has to go to the user.

Sometimes the device should try again

started by the interrupt handler

needs to keep track of where it is up to in error recovery

finite number of retries

what about an error in error recovery?

e.g. A write operation in UNIX

```
write(fd, buffer, count)
get file table entry from fd
check accessibility
lock inode
if a block doesn't exist for the current position
    allocate one - updates the inode
while not all written
    if not writing a complete block
        read the block in
    put the data in the block's buffer
    delay write the block
    update file offset, amount written
update file size
unlock the inode
```

But the device doesn't get called from this routine.

Delay write the block

buffers shared by system

write doesn't occur until another process is to use the buffer for a different block (LRU replacement)

kernel informs the disk driver that it has a buffer (size of a disk block) to write

starts the disk write but does not wait

disk controller interrupts on completion and releases the buffer, it goes on the free list

advantage?

if a process wants to access this information it is already in memory

e.g. process writes some more and it fits in the same block

disadvantage?

information is not written immediately

sync forces buffers to write

delayed writes are different from but similar to asynchronous writes

<u>Buffer pool or cache</u>

Type “free” and see how memory is being used on your UNIX system.

```
             total       used       free     shared    buffers     cached
Mem:  31176    6336      24840       6008        548       2756
-/+ buffers:          3032      28144
Swap: 24188          0      24188
```

```
             total       used       free     shared    buffers     cached
Mem:  31112   30612        500      21388        196      10716
-/+ buffers:         19700     11412
Swap: 24188       2896     21292
```

These buffers are used for all block device IO.

- super blocks - info about what is on the disk
- inodes - info about each file
- index blocks - where is the file
- programs being read in
- file data

Installing devices

Plug and play

- don't want to have to recompile OS
- don't want to have to restart the OS
- don't want to have to turn the computer off

So

- device must identify itself to the system
- type, requirements - interrupts, memory access, DMA channel
- system attempts to meet the resource requirements
    - this may entail changing the allocations to other devices
- loads device drivers automatically
- initializes device automatically
- removes device drivers when device is switched off or no longer present

More on disks

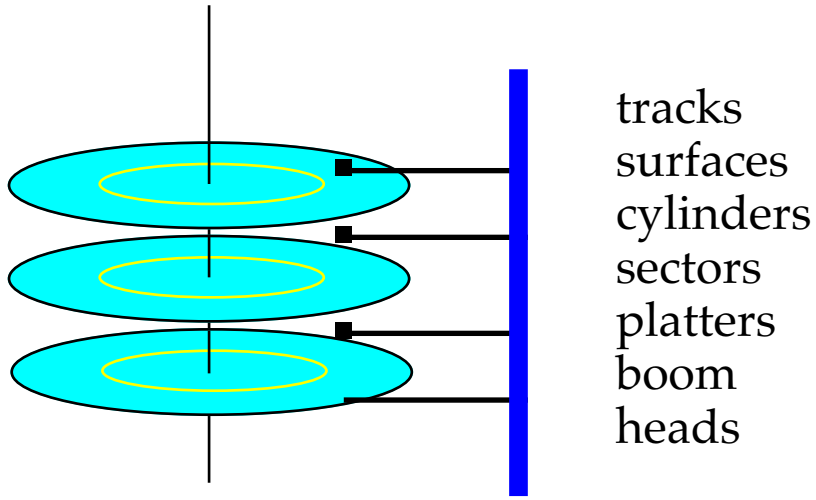
Still essential

> even though we now have a much main memory as disk space < 10 years ago

Disks devices are shared

Lots of requests

Want to maximise the throughput, minimise the average response and the variance.



Seek and rotational latency

Disk scheduling algorithms

FCFS - first come first served

> fair - not efficient

SSTF - shortest seek time first

> not fair (starvation)
>
> more efficient

SCAN - like SSTF but only move in until reach the centre and then out

outside tracks are discriminated against

N-step SCAN

only services requests which were present at the start of the sweep

stops new requests slowing down older ones

C-SCAN - circular SCAN

always goes one way and then zooms back to the beginning

Choosing one - depends on the load on the system.

Also depends on file layout - e.g. directories in the centre.

Some disk devices do the scheduling for you.