SCC.211 Operating Systems

Live Session 4

Dr Andrew Scott

a.scott@lancaster.ac.uk

Overview

- Topic 3: Memory Allocation (recap)
- Topic 4: Input and Output (introduction)
  - Device drivers
  - How can we create a device driver interface?
  - Disk storage

3. Memory Allocation

Recap

Videos

- Fixed and Variable length allocation
  - Internal vs. External fragmentation
- Buddy allocation
  - Allows allocation of contiguous memory blocks
    - Can be important for use by devices, sometimes in dedicated memory zone
  - Must track buddies; allocations cannot span buddies
- Slab allocation
  - Pre-allocate common sized memory areas, also avoids *allocate…free…allocate* cycle
  - Commonly used in kernel for frequently used structures

4. Input and Output (I/O)

Introduction

Includes diagrams from Silberschatz 10th ed.

Videos

- Overview of I/O mechanisms
  - How we attach devices and communicate with them
- Interrupt systems
  - How we work around unknown, and often long, duration of I/O operations
- Device drivers
  - Framework for device manufacturers to provide control software for devices

Device Drivers in Different OS Types

- Devices controlled by device drivers
  - In monolithic OSs
    - Tightly bound into OS *…failure will likely bring down whole system*
  - In modular OSs
    - Clean 'plug and play' interface *…more reliable and no need to reconfigure OS*
  - In microkernel OSs
    - Appear as unprivileged 'user' processes *…secure and easily restarted on failure*

An aside…

How can we create a device driver interface…

…or how to 'hook' arbitrary functions in C

C Function Pointers: *putting it together*

```
#include <stdio.h>

void input ( int * var ) {
        printf( "Enter starting value... " );
        scanf ( "%d", var );
        printf( "You input: %d\n", *var );
}

void addition ( int * var ) { (*var)++; }

void output ( int * var ) { printf( "After addition: %d\n", *var ); }

void ( * sequence[ ] )( int * ) = { input, addition, output };

int main ( int argc, char ** argv ) {
        int     variable;

        for ( int i = 0; i < sizeof( sequence ) / sizeof( void * ); i++ )
                ( * sequence[ i ] )( &variable );
}
```

```
Enter starting value... 100
You input: 100
After addition: 101
```

Storage: Disk systems

- Overview
  - Structure
  - Two approaches for improving
    - Performance
    - Resilience

Hard Disk Structure

- Significant access latency
  - Seek time to move head in/ out
  - Rotational latency
- Sector (native block) size
  - Traditionally 512 bytes
  - Now typically 4K

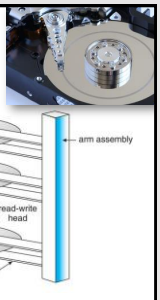

Figure 11.1 HDD moving-head disk mechanism.

Hard Disk Structure II

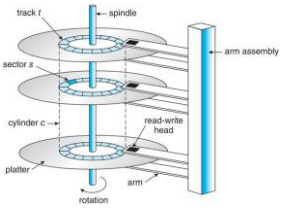

Figure 11.1 HDD moving-head disk mechanism.

Hard Disk Structure II

- Group tracks into zones of *n* tracks
  - As move from spindle out, each zone will have more sectors per track
- A run of sectors is known as a cluster

Cluster of *n* sectors

Disk Addressing

- Traditionally used simple { Cylinder, Head, Sector } or CHS tuple
- Now disk controllers typically offer Logical Block Address (LBA)
  - Hides what can now be a complex internal zone structure/ geometry
  - Also allows controller to hide bad sectors
    - Controller may maintain and remap spare sectors in case any become unusable

Redundant Array of Independent Disks (RAID)

- Aims
  - Improve throughput with parallelism
    - Block or bit striping data across multiple disks
  - Improve Mean Time Between Failure (MTBF), recover from failed disk(s)
    - Mirroring ('backup' / 'shadow' drives)
    - Error correcting codes

Common RAID Levels *…there are many*

- RAID 0: Block based **striping** across *n* disks to improve throughput
- RAID 1: **Mirror** each drive onto a shadow drive to improve resilience
- RAID 4: Block based **striping** across *n* disks + dedicated parity disk
- RAID 5: As 4, but cycles parity through drives so one (parity) disk not a bottleneck
- RAID 6: As 5, but adds another error correcting code to protect against multiple failures
- RAID 0+1 and 1+0: combination of levels 0 and 1
  - Stripe then mirror, or mirror then stripe

RAID 2 like 0 but bit based, RAID 3 like 4 but bit based (typically require complex drive synchronisation)

RAID Levels

- C: Mirror copy
- P: Parity
  - Recover from single failure
- Q: Second error code
  - Used in addition to parity enabling recovery from double failure

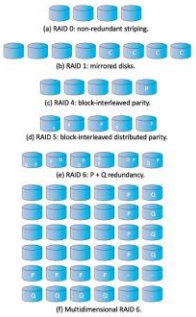

Figure 11.15 RAID levels.

Problems with RAID

- Mean Time To Repair can be long with some RAID levels
  - Significant time to rebuild lost drive
- Controller faults can cause whole array to fail
- Power failure can cause multiple (un-correctable) block writes to fail
  - A fast Solid-State Drive (SSD) based transaction cache can help