Lecture 25 — Reliability: Fail-Soft Operation

Jeff Zarnett
jzarnett@uwaterloo.ca

Department of Electrical and Computer Engineering
University of Waterloo

March 30, 2023

Except when we talked about hard drives and the Byzantine Generals Problem, we usually think that things will work as they should.
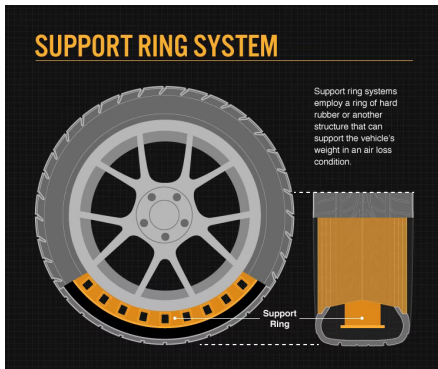

JUST A FLESH WOUND.

Downtime is okay for some systems: e.g., my laptop!

Reliability is important for real-time systems...
    Downtime may be intolerable if it's life- or safety-critical!

Or maybe it just costs money if you have some SLA.

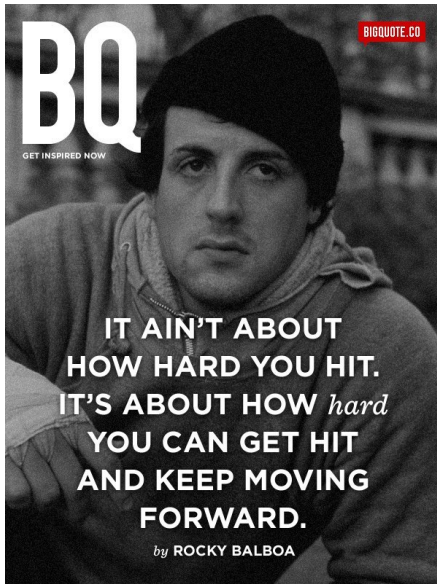Normally, if I get a flat tire, I can't drive until I change the tire.

Run-flat tires have limitations, but the car is functional at reduced capacity.

Two distinct goals:

1. Resiliency – carry on in the event of a failure.

2. Fail-soft – preserve as much capability as possible or terminate gracefully.

Where did RAID fall in this spectrum?

First question is how much resiliency you really need?

Is this "we lose money" or "people might die"?

Anybody not wearing two-million sunblock is gonna have a real bad day. Get it?

But no need to plan for nuclear armageddon (usually).

This is an engineering design tradeoff: how much extra capacity should we have?

Too little capacity: every problem becomes global...
Too much and we wasted money!

Best option for resiliency is to fix it!

Maybe not possible if this is an unrecoverable hardware failure.

Maybe deadlock detection and recovery?

Continue as best we can at reduced capacity.

Example: 4 CPU cores, all running at 50% capacity, but one dies?

Maybe running at reduced capacity until a repair comes…

What if we can't meet all deadlines?

The system is considered stable if it will always meet the deadlines of its most critical tasks.

Even if lower priority tasks may not be completed at all!

Painful choices may need to be made.

Perhaps it's sensible to do an orderly shutdown.

Prevent data corruption, but cannot carry on.

Cease all operation or execution immediately.

This may cause some damage, but may be the least bad choice.

Until now, vague words about "something going wrong".

What is a fault and failure?

Failure: When the response (outcome) deviates from the specification as a result of an error.

Error: A manifestation of a fault.

Fault: An erroneous hardware or software state of some variety.

Permanent: dead hard drive, software bug.

Intermittent: fault hardware chip.

Transient: cosmic radiation?

Instead of jumping right to fault tolerance, what about prevention?

What strategies would you use?

Following appropriate design, implementation, and testing processes will only make faults less likely.

It is also important to resist the pressure to drop or degrade these processes when user time pressure.

Still need to think about tolerance...

We already know a few things about fault tolerance from earlier topics.

Can you think of some?

- Process Isolation
- Dual-Mode Operation
- Preemptive, Priority-Based Scheduling
- Checkpoints, Transactions, Rollback
- RAID
- Checksums, Parity Bits, ECC...

Information redundancy: checksums, parity bits, ECC...

Physical redundancy: two CPUs instead of one...

Temporal redundancy: TCP communication, resend...

The Space Shuttle had both physical and temporal redundancy.

Important thing: no single point of failure!

We covered this a little bit in the Byzantine Generals Problem.

We'll consider one in particular: clock synchronization.

It is not trivial to get two independent systems to agree on what time it is.

Inevitably, all clocks except the universal reference clock are off by some amount; it's just a question of how much!

It's easy to imagine scenarios where independent systems who don't agree on what time it is will misbehave.

Ever been in a video call with lag?

Time zones also matter!

Clocks frequently use quartz for synchronization, but there is always drift and measurement error.

A quartz clock will typically vary by about half a second per day, so the idea of systems being off by a full second is quite reasonable.

Just imagine System A is fast by 0.5s and System B is slow by 0.5s.

In a non-real time operating system, it's often okay to just change the clock to the correct time and just jump there.

But for a real-time system, breaking the expectation of linear time can cause events to run again, so typically we do not wish to do that.

The solutions are effectively a graduated slowdown or speedup... but to when?

A possible solution is something like the Network Time Protocol.

Effectively, it's difficult or impossible to get more than one system to agree on what time it is.

Better: build your system to account for these things.

All of this is just a very simple overview of some of the issues that might arise when we have multiple systems for redundancy.

This is a complicated subject and is a whole 4th year ECE technical elective that you could take!