Discussion 14: I/O, Dependability

Hamming ECC

Recall the basic structure of a Hamming code. Given bits 1, . . . , m, the bit at position 2n is parity for all the bits with a 1 in position n. For example, the first bit is chosen such that the sum of all odd-numbered bits is even.

i. How many bits do we need to add to 0011_2 to allow single error correction?
   Parity Bits: 3

ii. Which locations in 0011_2 would parity bits be included?
   Using P for parity bits: $PP0P011_2$

iii. Which bits does each parity bit cover in 0011_2?
   Parity bit #1: 1, 3, 5, 7
   Parity bit #2: 2, 3, 6, 7
   Parity bit #3: 4, 5, 6, 7

iv. Write the completed coded representation for 0011_2 to enable single error correction.
   1000011_2

v. How can we enable an additional double error detection on top of this?
   Add an additional parity bit over the entire sequence.

vi. Find the original bits given the following SEC Hamming Code: 0110111_2
   Parity group 1: error
   Parity group 2: okay
   Parity group 4: error
   Incorrect bit: 1 + 4 = 5, change bit 5 from 1 to 0: 0110011_2
   $0110011_2 \rightarrow 1011_2$

vii. Find the original bits given the following SEC Hamming Code: 1001000_2
   Parity group 1: error
   Parity group 2: okay
   Parity group 4: error
   Incorrect bit: 1 + 4 = 5, change bit 5 from 1 to 0: 1001100_2
   $1001100_2 \rightarrow 0100_2$

RAID

Fill out the following table:

	Configuration	Pro / Good for…	Con / Bad for…
RAID 0	Data disks without check information	No overhead Fast read / write	Reliability
RAID 1	Mirrored Disks: Extra copy of disks	Fast read / write Fast recovery	High overhead → Expensive
RAID 4	Transfer units = a sector within a single disk. Errors are detected within a single transfer unit Can handle independent reads/writes per disks	Higher throughput of small reads	Still slow small writes (A single check disk is a bottleneck)
RAID 5	Check information is distributed across all disks in a group.	Higher throughput of small writes	

Small accesses = an access to a single disk in a group

I/O

Fill out the following table of polling and interrupts

Operation	Definition	Pro/Good for	Con
Polling	Forces the hardware to wait on ready bit (alternatively, if timing of device is known – the ready bit can be polled at the frequency of the device). It basically means manually checking the ready bit regularly.	PRO: -easy to write -poll handler does not have excessively high overhead -deterministic -doesn't require additional hardware Good for: -Mouse, keyboard	Infeasible on hardware with fast transfer rates that is actually rarely ready (e.g. Ethernet card receiver)
Interrupts	Hardware fires an "exception" when it becomes ready. CPU changes $PC to execute code in the interrupt handler when this occurs.	PRO: -Necessary for fast devices that are rarely ready. Good for: Fast devices - Hard drives, Network cards	-nondeterministic when interrupt occurs -interrupt handler has some overhead (saves all registers), meaning polling can actually be faster for slow, often ready devices such as mice

Availability

1) In this example, a WSC has 55,000 servers, and each server has four disks whose annual failure rate is 4%. How many disks will fail per hour?

(55,000 x 4 x 0.04) / (365 x 24) = 1.00 → MTTF = 1 hour

2) What is the availability of the system if it does not tolerate the failure? Assume that the time to repair a disk is 30 minutes.

MTTF = 1, MTTR = 0.5 → Availability = 1 / (1 + 0.5) = 2/3 = 66.6%