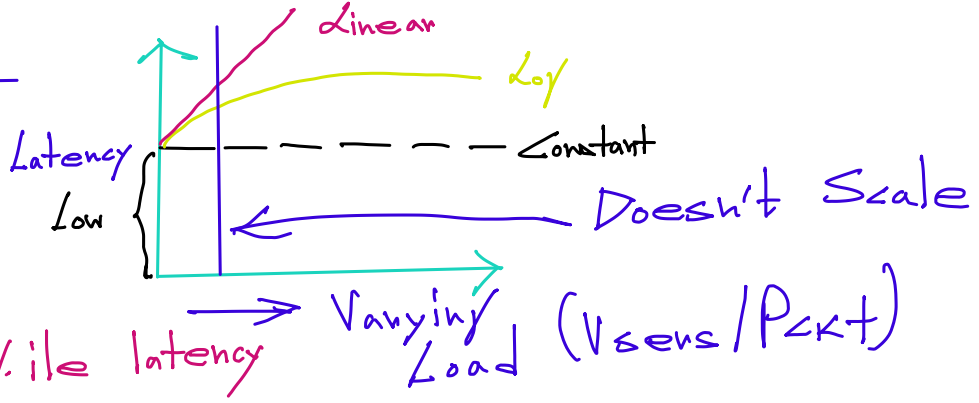
Basic Metrices

1) Throughput:-

2) Scalability:-

3) Latency:-

Median Latency, x%ile latency



4) Utilization:-

a) Time Based

$\frac{B}{T}$ → Resource Was Busy

T → Time Period T

Not good (Process doing busy Waiting on CPU during I/O, But it shows high CPU utilization)

Q) Why Semaphore is not used in Kernel? (spin-lock)

⇒ Context Switch

b) Capacity Based

5) Saturation → Reaching the breakpoint? Bottlenecks

Instrumentation:- Code to measure characteristics of the system

Static — Printf()

Dynamic — Tracepoints
VSDT

Breakpoint:-

Adding New Instructions In a Virtual environment

For Dynamic Instrumention of Kernel, Virtualization can not be used.

→ Counters:- Performance Counters

128 bit ⇒ Aligned by Bus Size

Memory Mapped Registers

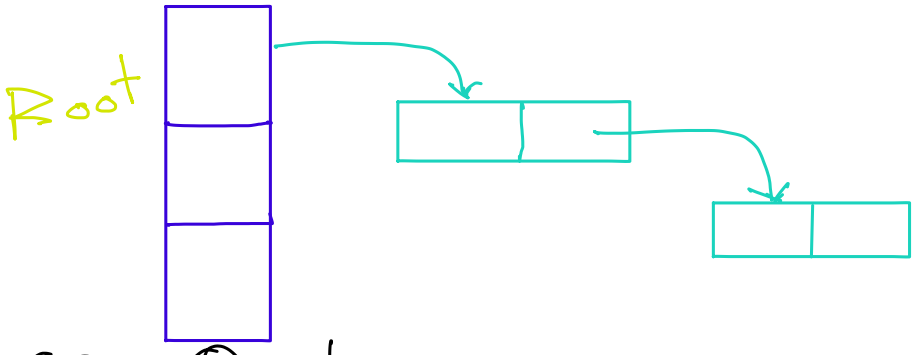
asm

Garbage Collection

⟹ The variables in stack are closely bound to the scope

⟹ Scope bound memory creation

1) Fast
2) Small memory footprint
3) Should not hamper the original working of the code



① Reg ② Stack
③ Live Object

⟹ Which part to focus more on?

⟹ Newer ones are likely to be used few times and become stale

gc.get_threshold()

⟹ Kref functions have to be atomic

⟹ Older generation fills ⟹ Move to younger generation

⟹ Bytecode → Platform independent

SIMD

Operations should be vectorizable

⟹ Multicore Programming

Embarrisingly Parallel

Branch prediction ⟹ Security Vulnerability

Downclocking SIMD will run the CPU in less speed

⟹ Non GPU → SIMD Bad

Multicore

RAM → Shared

Pipeline → Not Shared

What to share?

What to Keep Unique?

Some registers are shared, Reading or Writing them is expensive

Locks:- Test_and_set → Boolean Values

Compare_and_Sweep → Memory Regions

Spin_lock:- Battle_neck ⟹ Write(Set) is shared among cores

Context Switch > Page fault > Cache Coherence

Word alignment ⟹ Each Cache belongs to one Core

Lock Free Programming

Not atomic ⟹ Race Condition → Inconsistent state

Mutex ⟹ Only the owner can release the lock

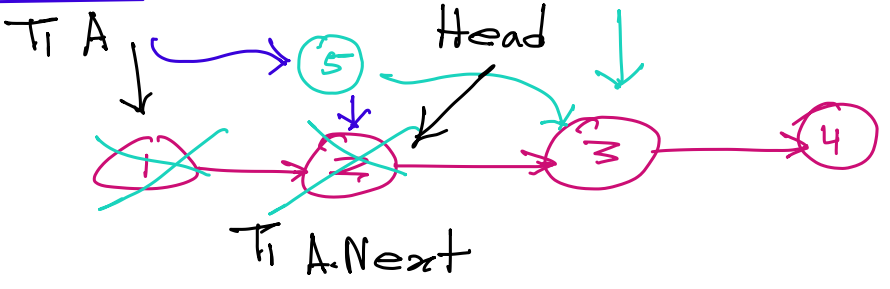
Semaphore ⟹ Anyone can release the lock.

futex ⟹ Pthread mutex implementation

⟹ How to make a thread more priority?

⟹ Lamport's non-blocking buffer

ABA Problem:- More Writers, What is the issue?



Lock-Free Primitives

Test and Set is implemented using LAS

Compare and Swap

⇒ Read and Write happens in size of word (Separately reading boolean is a waste)

Hardware:- CMPXCHG ⟹ Only for single core
⟹ Any threads can do any change in memory

Q) What if all 4 cores execute LOCK CMPXCHG together with same argument

⟹ Other cores except one has to stop

Wrappers ⇒ For different architectures

Q) User space process disables interrupt, printf("—"). Suppose you take away terminal display? ☺

RCU

141-152

smp_store_release

↳ send a signal to g pointer indicating it is updated

Q) Can we run (RCU) in user space?

Can not dereference gptr just after allocating. Need to publish first to all the cores.

Lightweight Threading

⇒ Don't overlap indiscriminately

⇒ Don't give CPU to A just because B is blocked. Give it when A needs it.

I/O Bound process moves from CPU to I/O? Advantages?

1) Multiple I/O Devices

2) I/O Device has Read/Write Buffers

3) Bus Size of Devices, Parallel read/writes

Why threads are taking so many cycles?

⟹ Threads are managed by Kernel Threads. So, you will need to visit Kernel space.

Not as bad as process since Process does a lot of other SYSCALLS

asyncio ⟹ Important for exam

No join for thread ⟹ While (1)

Thread ⟹ In Kernel, interrupt handler ...

⟹ Scheduler, take multiple threads from the user space and maps them to Kworker thread.

SYSCALL — functions in Kernel

Two level of interrupt handlers

Give ISR to process. Kernel Stack is used of the program. (Process) ⟹ Use the PCB of same process

Kworker thread for lower interrupt handlers

⟹ Using Threadpool ⟹ Scheduler is the bottleneck (GO)

Solution: Multiple Schedulers

Problem: Synchronization (Load Balancing)

Work Steal

(Assign from heavy loaded queue to lightly loaded queue)

Stack state :- Pointer to start and end where the stack is stored.

byte* entry ⇒ The function to be callbacked

Threads → What will happen? Sometimes program will run out of memory and it will crash

struct P → Virtual Processors

⇒ yield in python

⇒ OS threads are collaborating

Q) Why go uses heap?

⇒ Copy Since Go Threads will not use much stack most of the time

Q) What is the cost of this go routine?

⇒ Memory Read and Writes

Kernel Bypass

DMA ⇒ No cpu involvment in I/O ⇒ Grandfather of Kernel Bypass

RDMA ⇒ Remote DMA ⇒ Father of Kernel Bypass

NIC :-

Buffer is circular :- No need for two pointer

Speed :-

N/W < HDD < RAM < Cache < CPU

Q) Why DMA the packet?

⇒ If a Kernel thread tries to do it, it will have to do time-sharing with other threads

⇒ Buffer will be mapped to registers in the chip set

⇒ Shared between NIC driver and NIC card

Bottlenecks :-

1) Reading/Writing Data from buffer

2) Locks for accessing/shared data structures

⟹ Can not give userspace processes access to Interrupts.

Database and Optimizing Storage

Transactional

Analytical

Is SELECT a database transaction?

⟹ No!

Databases can be really really slow (opposite to our intution)

⟹ Why database instead of flat files?

⟹ System logs (Generally never read)

⟹ Database is used when computation is to be done. Store older data in logical way

Q) Replace flat files with database. What's the cost?

⟹ B-Tree / log(n) Time Complexity

Shared file System

Network Overhead

Database Schema

How? ⟹ Workload

Components of a Database

A single database is single file or multible file?

⟹ One database one file is a good model.

Query execution engine will tell which place in storage

OLTP

OLAP ⟹ Analytics Workload (No need of acid property)

OLTP:- ACID ⟹ Strong guarantee

Q) Relational ⟺ ACID? No

HIVE

⟶ We don't care about Key-Value

Mongo

DynamoDB

How are these database stored in Disk?

Q) Why page size is 4, 8, 16 KB? Not 16 MB

⟹ You will have to do more calls

THP :-

Q) 1 GB Page Size ⇒ Issue?

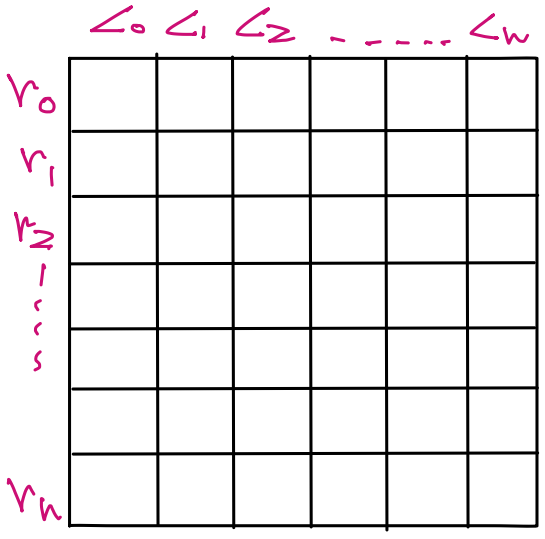
⇒ Fragmentation, Caching (Need to even Know how Much of page in cache)

Need pointers within page

Bookkeeping → New Linux

⇒ New Linux distro ⇓ Dedicated Server

Database Internals



Q) Which one to choose?

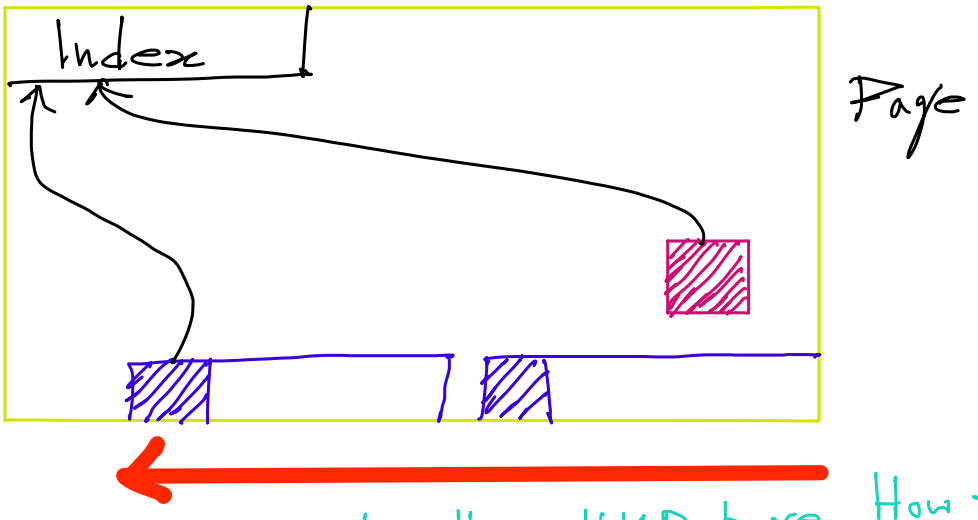
1) If more cols or more rows

2) Do you work on tuples? ⇒ Row

or do you work on column values? ⇒ Column

Row-Major :-

Page ⇒ Filesystem page (Hardware or Filesystem or MM?)



Q) Postgres only allows 4KB page. How to deal with it?

⇒ Pointers to some of the column entries

Column → TOAST Table → Data

⇒ lookup for primary key ⇒ Very Bad Idea

Column-major:- If Row j asked ⇒ Row j+1 likely need not be asked
Column j asked ⇒ Column j+1 likely to be asked

Will not fit in single page

Don't know which entry maps to which Row

PAX storage Model:-

→ Cluster the Rows ⇒ Then store in Column major format

Storage Optimizations:-

THP:- Linux asks not to use it, why? ⇒ Queries will take more time ⇒ All Queries will take time as SELECT * (Reading Whole Pages)

Benchmark

Numeric Representation ⇒ Treat as string
⇒ Fixed Precision

OLAP:- Statistical Query

Immutability ⇒ Append only

Sequential Scans

Merge Tree

Zone Maps:- When number of aggregate queries > 99.99% /Search

Bitmaps:- Both YES & NO to tackle null values

Bit weaving:- Paper

NOSQL:- Horizontal Scaling