SE Linux (Security Enhanced) has extra security everytime a new object comes up.

Points

1. fork() :-

return value : -ve . creation of child process unsuccessful

0 : Returned to newly created child process

+ve : Returned to parent /caller . contains pid of newly created child process

Child process : uses same PC, Registers, open files of parent

different data and state for the two processes

eg
```
x = 1;
if( fork()==0 )
    print (++x); print (&x);
else print (--x); print (&x)
```
o/p : 2 0 or 0 2

prints location in main memory

both give same result as the child process has an exact copy of all memory segments of parent process. It has a separate address space

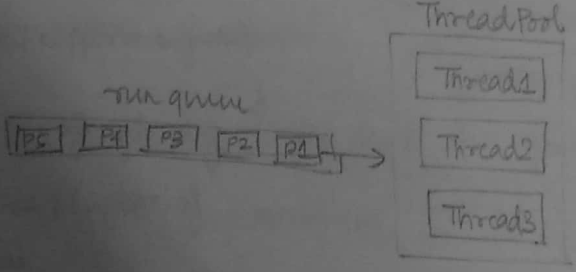
2. Clone()

3. Trashing : Process which creates page faults every few instructions

4. Thread Pool in Java : Create a fixed number (or on demand) of threads & use them as necessary. Addresses two problems :-

(i). The initialization phase of a thread is bunked.

(ii). If we have limited no. of threads, less system resources are consumed.



Syntax :

```
ExecutorService pool = Executor.newFixedThreadPool (<size>);
    Runnable r1 = new Task ("task 1");

             rn = new Task ("task n");
    pool.execute (r1);

      pool.execute (rn);
```

```
class Task implements runnable

void run (){

}
};
```

5. <u>Magic Number</u>

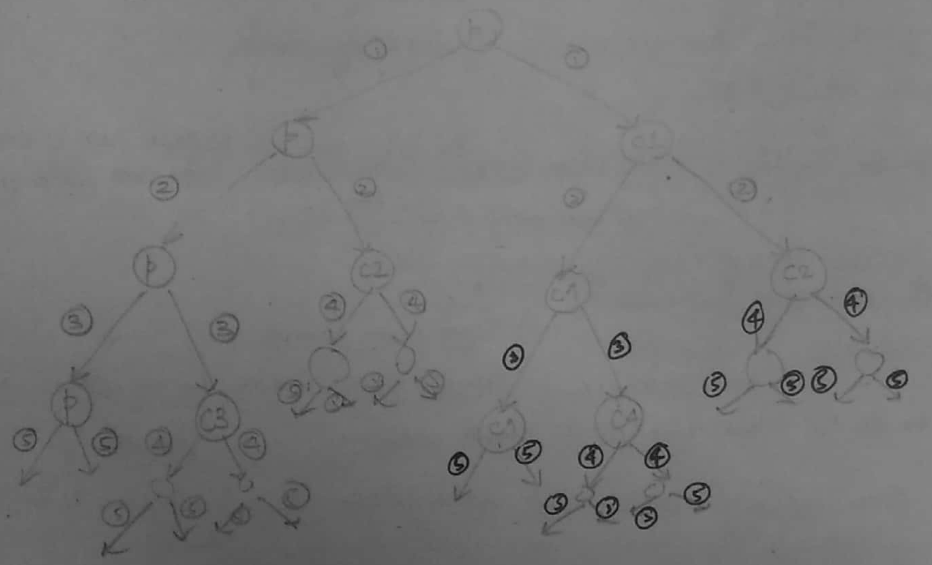
```
while (num > 10) {
    num = sum_of_digits(num);
}
if (num == 1)   original number was magic number.
```

<u>In Java</u>

The JVM & the class loader take special care to ensure that the class are loaded intact. For this, every class definition contains at the beginning a magic number, a sequence of 4 bytes that identifies it as class definition file. The magic number is "CAFEBABE".

6. Fork Example:-

```
fork(); →1
2(fork() && fork()3) || fork()4;
fork()5;
```



Total 20 processes

7. CFS achieves the following:-

(i). Every process gets a fair share of CPU: Since processes are ordered according to CPU times received, assurance of each process being attended to equal time is given

(ii) Priorities taken into consideration: For low priority jobs time runs quickly. Their vruntime increases faster than jobs of higher priority

(iii). Execution time dependent on I/O boundness of process: Since I/O is not taken into consideration while computing vruntime, total execution times varies.

(iv) Context switching time is minimized: Implementation is done using RB tree. Time to select new node (job) = O(1). Inserting task back = O(log N) which is lesser than the linear priority queue.

8. Any solution to CS problem should ensure:-

(i) mutual exclusion

(ii) No process outside critical region can block another process

(iii). No process should have to wait forever to enter it's critical region.

9. XCHG

fn enter_region:

```
MOVE REG, #1
XCHG REG, LOCK
CMP REG, #0
JNE enter_region
RET
```

leave_region

```
MOVE LOCK, #0
RET
```

10. Finding out Java JVM bit version: java -version

Mi- 64 bit JVM

Theoretical limit of heap size $= 2^{64}$. (Not possible because RAM is limited) max ~ 32 GB for 64 bit JVM

on 32-bit ~~JVM~~ JVM $= 2^{32} = 4G$

Default max heap size $= \frac{\text{RAM SIZE}}{4}$

Start $=$ RAM SIZE / 64

Windows can allocate less Heap Space than Linux because it tries to allocate contiguous heap.

We can specify more heap size than RAM because of virtual memory management.

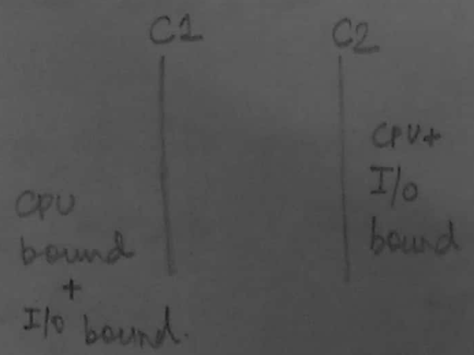
1 M → minimum heap size.

LKM :- Loadable Kernel Module. (Kernel Object ".ko" file).
Used to insert new drivers.
This requires to export all the kernel symbols needed by the outside modules.

To protect the OS from crashing, a shadow driver is present along with every driver. Nook's Architecture has this flexibility.

Three reasons why people didn't move into C++ kernel :-
(i) Few Open Source C++ developers.
(ii). What can be done in C++ can be done.
(iii). Slow (Not very correct because the C++ compiler does lot of optimization).

GPU Assisted Scheduler (GAS).



Hardware Performance Counters - balance.

Unix File System Design

Regular files

Device files → Character Device Files (no seek)
Device files → Block Device File (Seek possible)

Note: Sockets are also seen as files.
Pipes } IPC

The problem of considering everything as a file is that we do not have specific control over the management of that device. The device won't be able to support something unique (eg. interface) outside the scope of file interface.

Eg: All Network devices (like sockets) can be abstracted from files.

Unix File

Every U-File has "Index node" (i-node) which has the meta data for the file (like filename, permissions, file type, last access time, last modified time)

In case of file corruption, we need to do File check (FSCK) during boot-up. This forces the OS to repair the broken links in the file.

Database cannot allow data corruption but OS doesn't care.

atime
mtime } Jiffy's counters for access & modified.

log-file   used to maintain file consistency.

Access permissions :- s u g o

s → sticky
u: rwx
g: rwx
o: rwx

Data blocks

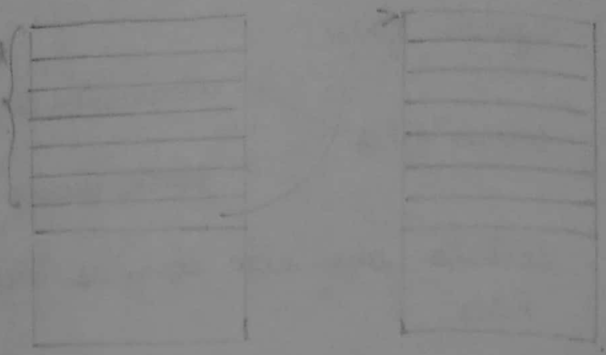
~~9~~ 12 direct addresses

1 indirect address block

2 indirect address block

3 indirect address block

9 direct



Most Linux Files size $< \cancel{9}\,12 \times$ block-size

Next Level File size $< \cancel{9}\,12 \times \text{block\_size} + n \times \text{block\_size}$

Next $< \cancel{9}\,12 \times \text{block\_size} + n \times \text{block\_size} + (n \times n) \times \text{block\_size}$

Next $< \cancel{9}\,12 \times bs + n \times bs + n \times n \times bs + (n \times n \times n) \times bs$

seek time :- Time to move head to appropriate position

Transfer time :- Time to transfer data.

1 block read from hard disk = seek time (major) + transfer time.

Interleave factor :

Disk Scheduling Algorithms

H/w Raid Disk vs Normal Disk

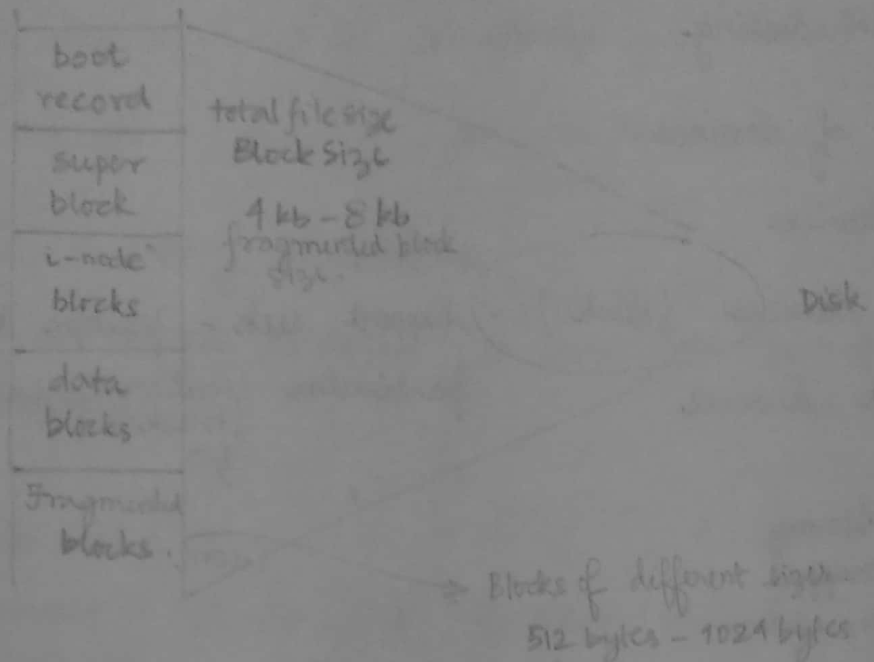
For Windows File System, metadata is stored in FAT.

In Linux when we do mkdir, an i-node is created storing metadata & a data block is also created to store data.

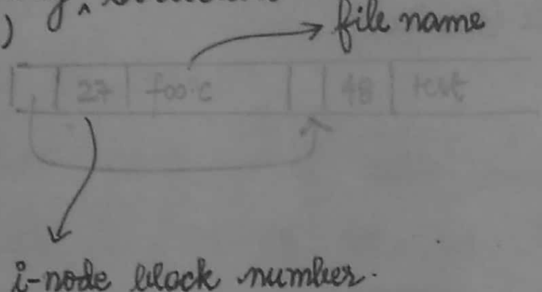
H/w 3 sec gap b/w file creation on one side & other
1 program creates a file & writes to it & other program reads

Buffer stores the newly created i-node & ~~metadata~~ data block which is flushed periodically

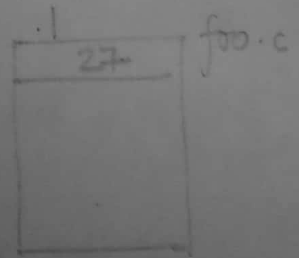
Unix File System doesn't guarentee consistency. For consistency we need locks on the files.



Directory i-node structure
(is a file)



Linked list structure



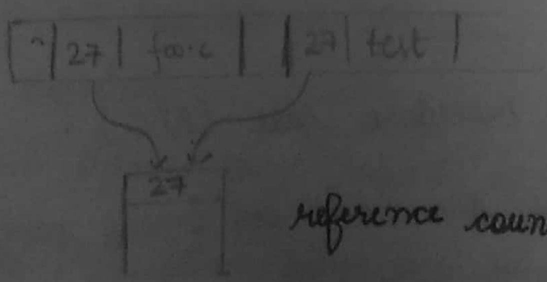
i-node of file doesn't store name of file.

Note: Directory doesn't have data nodes. So it is essentially only i-node

4/10 Read i-node information of directory & files: ls -l ... lists i-node numbers.

i-node also maintains reference count of a file - No. of files pointing to it.

Eg:



reference count = 2 stored in file inode.

When reference count = 0, it is safe to delete the file.

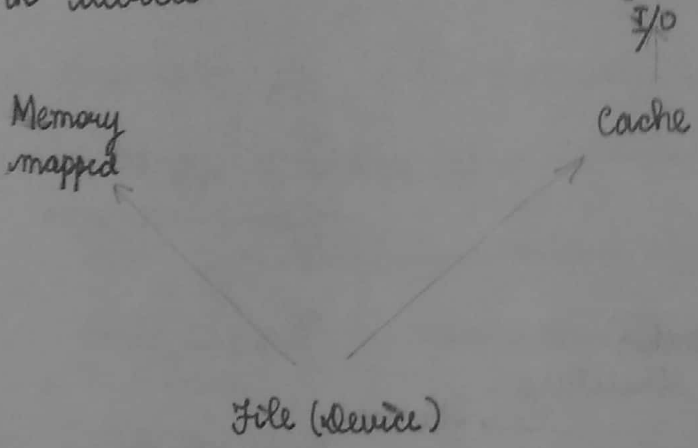
I/O Scheduling

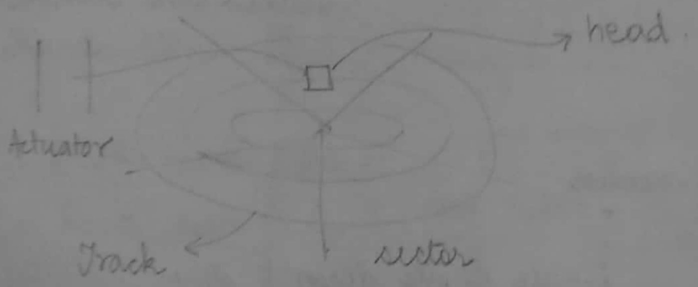
3 types of devices

(i) char devices

(ii) block devices (disk) - Support seek :- position head on a particular location & read. (track)

(iii) network devices





Note: Seek is the slowest component.

I/O calls on files

(i) fd = ~~fs~~ open-file ("<file-name>")

fd contains block start, buffer created

(ii) read (fd, ...) → bring i-node of data blocks to local cache

(iii) seek (fd, ...)

(iv) write (fd, ...)

(v). close (fd)

All these are Remote Procedure call (RPC).

In NFS, the changes are made on i-node cache. This is destroyed every 3s & made consistent with the disk.

The data cache is refreshed every 30 ms.

Each I/O request becomes a seek operation.

Disk Scheduler : Job is to optimize seek time.

I/O requests : Series of tracks to be read.

Disk scheduling algorithms

Disk the slowest component. So more effort to optimize it -

Interleaving

Disk Failure

SmartLog :- Records disk performance. Learning algorithm to predict failure. Written every few seconds.

Time becomes a main factor to predict failure (Real Time Prediction)

1. First Come First Serve

(i) Inefficient int terms of disk access.

Eg: 68 22 74 86 $\underbrace{11 \quad 21}$ (Tracks requested).

Seek time = 10.

Total Seek = Σ Seek time.

2. Shortest Seek First

(i). From current head go to nearest track.

(ii). Can lead to starvation.

3. SCAN / Elevator

(i) Service all requests in one direction. Turn and service in other direction.

~~(ii)~~ Unfair because some request which come in later might be serviced before what has come earlier.

4. C-SCAN

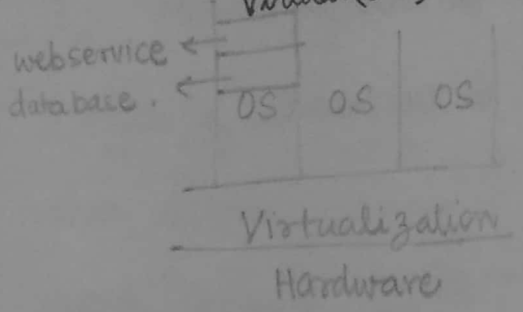
(i) Service all in one direction. Return back to start & re-scan.

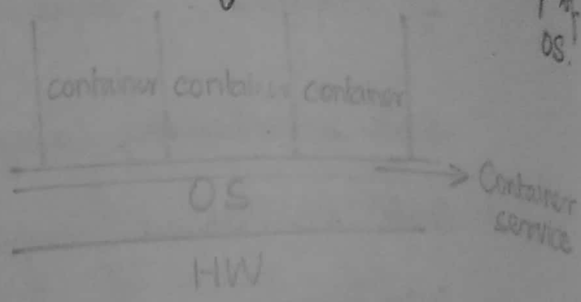
(ii). More fairer.

24-10-17
Tuesday

Virtual(Box) Instance vs Containers

Virtualization of applications on top of OS.





application development tied with OS.

Eg: C program written on Linux willn't work on Windows because system calls like malloc, fork, etc. won't work.

Containers can run on top of any OS because it provides container service.

The VM Image is few GB (~20 GB) : OS (few GB) + application (few mb)

containers : few mb. Spinning instance is very fast.

Docker : popular container service.

Container Scheduling : For the containers on top of OS scheduling.

Advantage of virtual instance : hardware sharing (Different users can work)

Containers : OS Sharing.

Support for container by OS - flexible policies by kernel that can be init-process

Single Hierarchy

using

Resource Groups

CPU, Memory, Network, Devices, etc.

Limiting resource bandwidth to different processes not possible in single hierarchy.

Eg: (i) Running web service - some amount of network bandwidth
(ii). Shell
(iii). Word Processor - some amount of memory.

Limiting bandwidth :- (At low level)

CPU - resource group :- set CPU usage limit G1 ← PId_1
CPUset :- which cores to use.
Memory - resource group :- memory limits G2

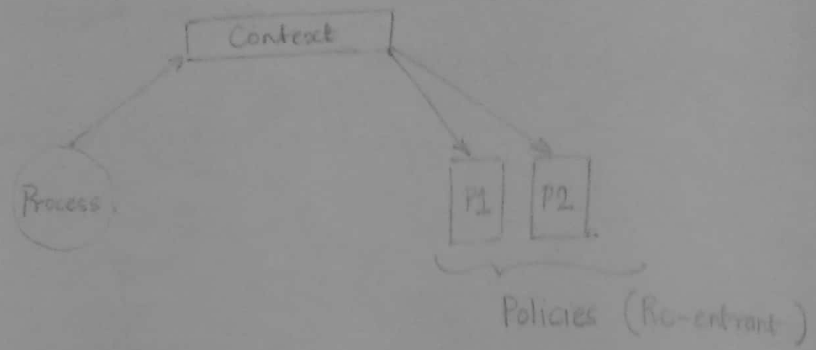
Network - resource group G3

Lab Create (memory) control group, put memory limit. Stressing program to check if the limits are enforced. (Opt. Create NameSpace & attach processes).

Namespace (At High Level) :- Group various control groups together to enforce various restrictions.
It is used to address the problem of creating context of a process.

We can attach processes to namespace. Eg. Network namespace, File System namespace.

Even android can have namespaces. But currently in Dalvik VM, we can only specify the heap size.



currently policies are hard-coded. So we cannot change policies for any kernel.

To be able to attach dynamically changing policies, requires objects.

26-10-17
Thursday

Within the kernel
Containers are built using
(i) CG Groups - resource provisioning & metering
(ii) namespaces - what is visible for a process
(iii). copy on-write - for optimization when adding containers.
With these integrated into the kernel, the kernel is atleast one container.

Memory limits & enforcing it on the process. Two types :-

(i) Hard Limit :- process killed if limit is exceeded.

(ii). Soft Limit :- process granted more resources. But in case kernel needs resources, it will take from this (Process which exceeded will be targeted).

Process within a container can be frozen - all processes are sent to wait state.

Block devices can be controlled by CGroups.

Note: If CGroups is not there, one process can adversely affect all the other processes.

How C Groups are created :-
A pseudofile system is created. CGroup is just a directory within that. The config is present within that.
Attach a process by copying pid of process into file in dir.

Eg : Cgroup / mygroup.

Namespaces

pId Namespace :- process within that can only see other processes within that.

Cloning :- Files are present in ~~separate~~ same namespace.

file Namespace :- specify when file system is mounted which files are visible to the processes

Eg : uid /usr/n1 /usr/n2.

Namespace :- what is visible

Network Namespace :- specify routing tables, ~~IP~~ tables.
Different namespace for private & public network.

1-11-17 Wednesday

clone - creates child process - set namespace (Separate context for child process).

File namespace - separate namespace is created. Parent file structure is copied.

We can then mount a new file system visible only to this process.

/tmp/abc - private for child (Namespace)

/tmp - parent namespace (cannot see abc).

8-11-17
Wednesday

OS Security

Malware

Wannacry

$$\text{Wannacry} \rightarrow \boxed{\text{Enter}} \rightarrow \boxed{\text{Encrypted files}}$$

Enter → Vulnerability

Encrypted files → payload

Vulnerability
(i) bug in program
(ii). humans

Subvert Execution :- change the expected execution of a program.

Steps :-

(i). Write malicious code in machine code.
Write first in assembly & do objdump.

(ii). Find buffer overflow in application

Defences :-

(i). Safer programming languages.
(ii) Make it difficult for malware to subvert execution
(iii). Detect malware's execution at runtime.
(iv). Sandbox system :- restrict malware's action
Track Information flow :- Keep a tag keeping track of sensitive information. Block when reaches network port.

<u>Canaries</u>

insert canary here
buffer 2
buffer 1

Insert canary & before returning from fn check if value is same.

-fstack-protector.

Scanned by CamScanner

Non-Executable Stack (W^X)

A segment can be only W (Written) or EX (Execute). An extra bit in page table will ensure this.

Counter :- Jump to code segment which is executable.

Typically libc code is exploited → system().

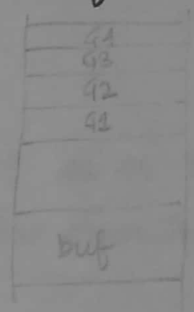
But this attack can only exploit fns present in libc.

Payload is not defined

ROP Attack

Gadgets :- Small pieces of useful instructions followed by return address.

Stitch gadgets to form payload.



Identify gadgets :-

Identify return instruction (0xC3)

Interpret instructions differently.

Target - x86 because CISC based. 1000's of instructions & variable length instructions.

Address Space Layout Randomization

Randomize the location of libraries & code segments.

Linux : /proc/sys/kernel/randomize_va_space.

Counters :-

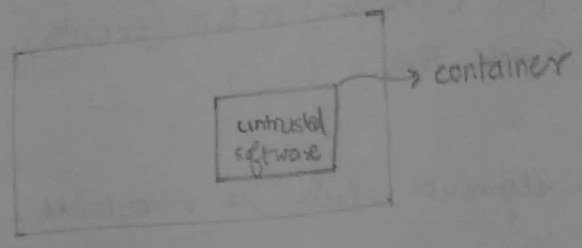
(i) Return to PLT.

(ii). Overriding GOT.

(iii). Brute force.

(iv). Timing attack.

Confinement



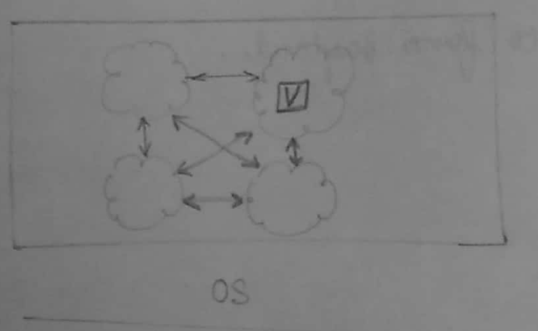
Isolation can be done by :- (different granularity)

(i) Air gap.

(ii) Virtual Machine.

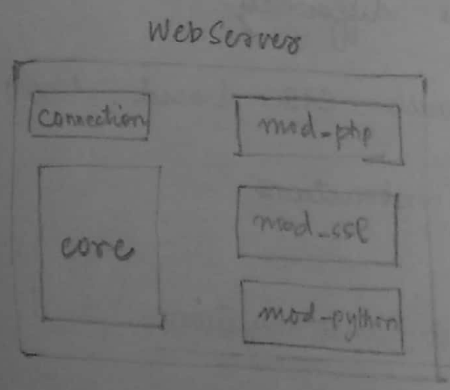
(iii) Run on separate containers.

Confinement within a process



Separate address space.
Problem :- context switch is expensive.

Partition a single application into separate modules.



Multiple users.
If one bad user is able to exploit one module, he can affect entire system.

Solution : Principle of Least Priviledges.

OKWS: Tradeoff b/w security / performance.

Achieving confinement in Unix :-

(i) chroot :- define file system process can see.

(ii) setuid :- set uid of process to confine what it can do.

(iii) Passing file descriptors :- priviledged parent can open file & pass descriptor to unpriviledged child.

Eg: $ password ⟶ change passwords. Passwords are written in /etc/shadow.

↓

'S' bit is set ⟶ setuid(0); ⟶ root priviledges.

Eg: chroot /home/user6/ — becomes new root. make it not access shadow file.

In OKWS, at least one node should be root so as to be able to read ports below 1024 (80: http) & pass descriptors to others.

Eg:
```
fork()
  if (child){
      setuid();
      chroot();
      exec();
  }
```

<u>Web Browser confinement</u>

C/C++ code not safe.

(i). Windows gave users the decision to run.

(ii) Chrome sandboxed C/C++ executables.

Isolate code, data. If request to outside → exception