

1/8

OSN

OS : Coordinates everything

↳ Middleman b/w Hardware and user

OS handles interactions with the disk and performs storage management.

2/8

```
1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <unistd.h>
4 int main(){
5     printf("This is parent. PID: %d\n", (int) getpid());
6     int rc = fork();
7     if(rc < 0){
8         printf("Fork failed\n");
9         exit(1);
10    }
11    else if(rc == 0){
12        printf("This is child. PID: %d\n", (int) getpid());
13    }
14    else{
15        printf("This is parent my child is %d\n", rc);
16    }
17    printf("Who am I? PID: %d\n", (int) getpid());
18    return 0;
19 }
```

```
1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <unistd.h>
4 #include <sys/wait.h>
5 int main(){
6     printf("This is parent. PID: %d\n", (int) getpid());
7     int rc = fork();
8     if(rc < 0){
9         printf("Fork failed\n");
10        exit(1);
11    }
12    else if(rc == 0){
13        printf("This is child. PID: %d\n", (int) getpid());
14    }
15    else{
16        int rc_wait = wait(&rc);
17        printf("This is parent my child is %d\n", rc);
18    }
19    printf("Who am I? PID: %d\n", (int) getpid());
20    return 0;
21 }
```

```
19 }
20 printf("Who am I? PID: %d\n", (int) getpid());
21 return 0;
22 }
23 // wait() makes the parent wait for any one of the child and returns the pid of child
24 // we can use wait() in a loop to wait for all children (loop will not return)
25 // waitpid(child_pid) waits for the specified child
26 // wait() only works for parent-child pair where parent waits for the child
27 }
```

5/8

OS → divides CPU work , makes it feel like any process has infinite access to CPU.

Process → Takes resources

• Memory :

- Static Memory → Variables [Stack]
- Dynamic Memory → malloc, calloc [Heap]

What Constitutes a Process?

- Unique Identifier (Process ID) getpid
- Memory Image
 - Code and data (static)
 - Stack and Heap (Dynamic)
- CPU Context: Registers
 - Program Counter
 - Current Operands
 - Stack Pointer
- File Descriptors
 - Pointers to open files and devices

Memory Image of Process

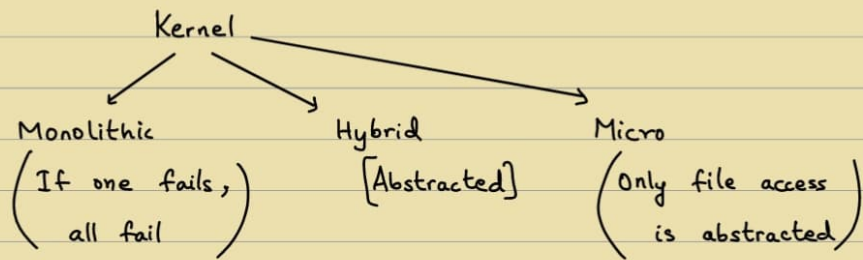
Code
Data
Stack
Heap

Create memory image for the process



Allocate PC, SP

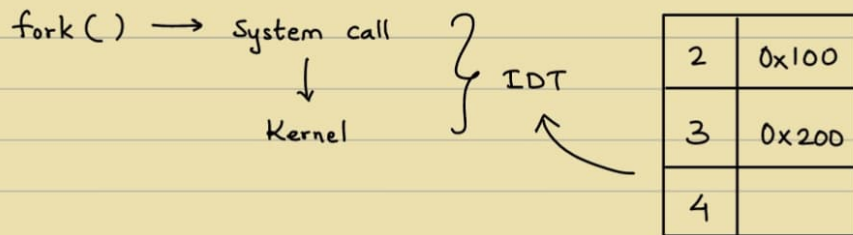
- Kernel: Handles direct instructions with hardware
 ↳ Part of the code



Kernel has stack
 ↳ Process

- API → System calls (fork, exec, wait)
 ↳ Resides in Kernel

- Limited Direct Execution (LDE) → CPU



System call ⇒ Mode change ⇒ User mode to Kernel Mode
 ↳ TRAP Instruction: Changes privilege

IDT ⇒ OS code gets executed
 ↳ Interrupt disrupt Table

Context → PC, SP → Saved

User stack X

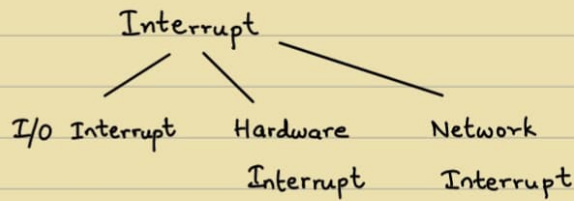
Kernel stack ✓

↳ Per process level

10

11 syscall ← OS

12

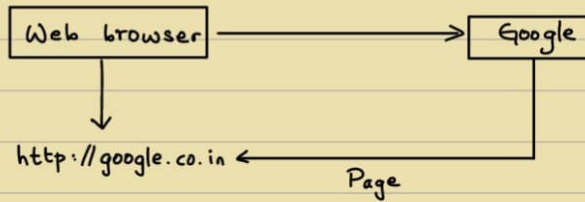


- Context Switch:

How to decide which process to run next → Scheduling

In reality, Jobs can arrive at any time.

26/8



Port → 80

Process id	Port

← Mapping

- Socket → Abstraction → Allows communication
↳ API

Stream → continuous (TCP)

↳ Mutual connection (strict) + Connection oriented

Datagram → No memory + No persistent connection (Stateless connection)

↳ UDP

↳ Unreliable (by default)



TCP	UDP
Connection Oriented	Not Connection Oriented
Reliability (order is maintained and retransmission)	No reliability at L4
Higher overhead - reliability, error checking, etc	Low overhead
Flow control (based on network)	No implicit flow control
Error detection - retransmit erroneous packets	Has some error checking - Erroneous packets are discarded without notification
Congestion Control	No Congestion Control
Use cases: HTTP/HTTPS, File transfer, Mail	Use cases: Streaming data, VoIP, DNS queries, ..

Flow control : Control speed of data transmission.

- UDP \rightarrow Unreliable at L4 \because Fire and Forget, i.e. responsible for giving data but not ensuring receiving.

Headers differentiate TCP and UDP.

\hookrightarrow 32 bits \hookrightarrow lightweight (8 bytes)

TCP \rightarrow 3-way Handshake (Agreement)

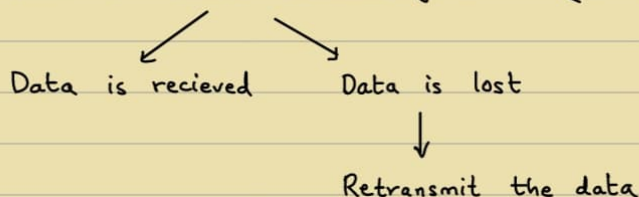
L4 ensures packets getting delivered in-order.

\hookrightarrow Transmission Rate

• TCP Header :

- ① Sequence number : Allows sequencing of data
- ② Acknowledgement number : Sent back from receiver, to let the sender know that data was received and is waiting for next data.

Data is sent \rightarrow Wait to receive acknowledgement (OS starts a timer)



OS does not want to waste resources

$\Rightarrow \therefore$ Starts a timer and if it elapses \Rightarrow Timed Out

Data can be lost even when acknowledgement no. $>$ sequence no.

Acknowledgement cache \rightarrow Prevents Deadlock

\downarrow
 Sender sends & waits for response (ACK)
 Receiver sends back & still on the way

“ IIIT Timer \rightarrow 1½ hour delay ”

- RTT : Round Trip Time

Sample RTT : How much time

Estimated RTT : Keep updating estimation on every sample

$$(1-\alpha) \text{ ERTT} + (\alpha) \text{ SRTT}$$

\hookrightarrow If very high, \therefore Works only on current & not past
 Default : 0.125

Deviation RTT : Worst case scenario , can happen but rarely

$$DRTT = (1-\beta) DRTT + \beta \times |SRTT - ERTT|$$

↳ Weighted moving range ($\beta : 0.75$)

$$\text{Timeout Interval} = ERTT + (4 \times DRTT)$$

↳ $\begin{cases} 3 \rightarrow 85\% \text{ confidence} \\ 2 \rightarrow 65\% \text{ confidence} \\ 4 \rightarrow 99\% \text{ confidence} \end{cases}$
[Based on experiments]

Q) Check acknowledgement constantly for every packet ?

Disadvantage (if check) \Rightarrow High overhead & Latency

Solution \Rightarrow Delayed Acknowledgement

↳ Wait after sending multiple packets and receiver sends ACK

To prevent deadlocks :

Timer on the receiver end

③ Window : Flow Control

Receiver has a limit to receive data & then waiting time.

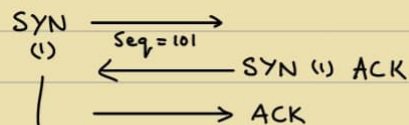
Send a 0 byte data to know whether receiver can receive data.

Receiver & Sender \rightarrow Both send data , but have separate values of ACK.

Sequence number \rightarrow Any random number within the bounds.

↳ Decided after agreement

3-way handshake \rightarrow "Can we start connecting?"



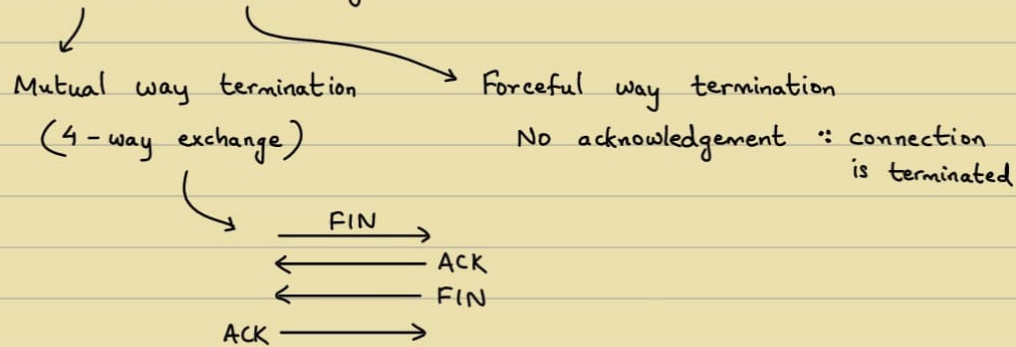
1 : Setting the SYN bit , i.e. sending the acknowledgement Agreement on Seq. no. from both sides

ACK \rightarrow 1 : Accepts the mutual seq. no.

SYN ACK : Mutual Agreement & that both receive data

SYN : 1 \Rightarrow Connection establishes and sender initiates sending

Closing connection : FIN and RST flags



Note : Size of TCP Header : 30 bits to 60 bits (Option Speed)

P bit : [Push bit]

Send data \Rightarrow Receiver gets and puts in buffer (Generally)

$P = 1 \Rightarrow$ Data is directly sent to process, No buffer used

U bit : [Urgent bit]

Some packets sent need to be processed urgently and rest can be processed later

- Memory :

Every Process requires memory.

Memory Virtualization \rightarrow Every process feels like its getting entire memory.

Some of the RAM is used up by OS, code and data. Rest of the memory is usable RAM, i.e. our code and data.

Multiple Processes running at the same time \Rightarrow Fast

Ways to achieve process virtualisation :

(1) Partitioning OS into slots

Disadvantages : - Other slots data can be accessed.

- Each process may not be fixed amount of data.

{ Saturday : 8:30 Tutorial \rightarrow Class }

Flow control : Just b/w sender and receiver (L4)

Congestion control : C-bit, In the network layer

• Memory virtualization :

e.g. Supermarket vs e-shopping

Every process \leftarrow Address space
 \searrow
 Virtual Address Space, mapped to Physical memory (VA)

* $p = 3 \Rightarrow 8p \leftarrow$ Virtual memory

Note : Only 1 Physical Memory \leftarrow Physical Address (PA)

Processes might not always stay in Physical memory, it can be in disk as well from overflowing. (Note : Swap Space)

* Translation : Memory Management Unit (MMU)

\rightarrow Fast

\rightarrow Conversion + Fetch

- Goals of Virtualization :

(1) Transparency \leftarrow Each process gets an illusion of infinite space

(2) Efficiency \leftarrow Use hardware

(3) Protection

- For Process Virtualization :

Mechanism : Limited Directed Execution (LDE)

Policies : Scheduling Algorithms

- For Memory Virtualization :

Assumptions :

(1) User address space is contiguous in Physical Memory.

Process \rightarrow Block

(2) Size of Address space is too big, less than size of Physical memory

Process \rightarrow No overflow

(3) Every Address space is of equal size.

Memory Virtualisation :

Q) How do I divide Physical Address space?

Q) How do I map Virtual Address to Physical Address?

Reference the variable \leftarrow Stack

Note : Base and Bound approach :

Values are stored in MMU

Then, Add Base Address

VA \rightarrow Process starts at 0

But PA : Process = 0 + Base value

$$= VA + \text{Base}$$

Accessing after bound \rightarrow Fault : Address out-of-bound error

Allocation : Dynamic \Rightarrow Dynamic Relocation

VA to PA : Address Translation

MMU not OS

* MMU \rightarrow @ Context-Switch

Has only 1 pair of Base and Bound

* Segmentation : (Memory Management System)

Code, Stack & Heap - Generalised Base and Bounds

Vsegment : Has a base and bound, Not Fixed

Dividing PA space

Segment Registers in MMU.

First \rightarrow Identify which segment

* Address Translation :

Stack \rightarrow goes up \therefore Subtract

$$\text{Real Address (PA)} = VA + \text{base address}$$

Code : No offset

Heap : Offset

Calculation

6/9

VA \rightarrow Every Process

Map : VA \rightarrow PA : Base and Bounds \rightarrow Disadvantages :

\swarrow
0 to Max

(1) Fragmentation

(2) Unused Memory

- Identify segment :

(1) Implicit :

Stack

(2) Explicit : (VA \rightarrow 14 bits)

First 2 bits \rightarrow Segment (12-13)

Rest \rightarrow offset (0-11)

00 : Code

01 : Heap

11 : Stack

\therefore Bits are unused \Rightarrow Code and Heap have same bit in some OS.

- Simple Address Translation :

Many OS : Segmentation \times

Paging \checkmark

PA : Base + Offset



Data - Max stack data

-

Segmentation

Fine-grained

| Small size segments

| High overhead

Course-grained

| Large size segments

When different sized segments , Disadvantage :

External Fragmentation \leftarrow Extra memory is available

But not contiguous , i.e. has holes

Defragmentation : High overhead

Note : Free space management algorithm , Closest Fit , First fit are algorithms

Fixed size segments \Rightarrow Code, Heap & Stack \times

Disadvantage: Internal Fragmentation

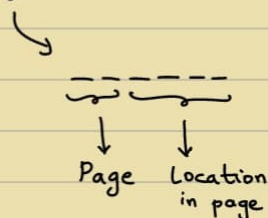
\rightarrow Better than external fragmentation

Page: Fixed size segment in a process

Paging: VA and PA are both divided

\therefore If 4 pages $\rightarrow 2^2 : 2$ bits

64 bytes VA $\rightarrow 2^6 : 6$ bits



VPN : Virtual Page Number
PFN : Page Frame Number } \leftarrow Every page has a number

In PA : Page Frame

In VA : Page

Advantage :

(1) Flexibility : OS finds a free page

(2) Simplicity : \forall Process, \exists Page table

\rightarrow Mapping b/w VA and PA

Array can be used, e.g. $VPN[i] = PFN_j$

Note: Multiple VPNs can be mapped to a PFN (not same process)

If $Mem_{process} > Mem_{RAM} \Rightarrow$ Swap in & Swap out from Hard disk

- Process of Translation :

VPN Offset
 $VA_5 \quad VA_4 \quad VA_3 \quad VA_2 \quad VA_1 \quad VA_0$

e.g. `mov 21 %eax`

$(21)_{10} = (10101)_2 = (010101)_2$

01 : VA space }
 0101 : Offset } & if $VPN_1 = PFN_7$ (from page table)

Then $\underbrace{010101}_{VA} \Rightarrow \underbrace{1110101}_{PA}$
 $(1110101)_2 = (117)_{10}$

Page Table Base Register \rightarrow store base of the address of the page table.

e.g. 32 bit address space with 4 KB pages
 no. of bits for offset ?

$4 \text{ KB} = 2^{12} \Rightarrow 12 \text{ bits to represent}$

$32 - 12 = 20 \text{ bits for mapping, i.e. VPN}$

$\therefore 2^{20} \text{ mappings per process}$

Each mapping $\rightarrow 4 \text{ bytes}$

$\therefore 2^{20} \times 4 = 4 \text{ MB per process per page}$

If 100 processes $\Rightarrow 400 \text{ MB}$ for address translation

\rightarrow Page Table

\therefore Use pages to store page tables

- Page Table:

Page Table Entry (PTE)

VPN, PFN, V, P, Present, Dirty, Ref

Valid Bit

Protection Bit

If page is recently used
 If page is modified (from memory)

If page in Physical memory or secondary

\therefore Disadvantage: Fetch twice for every translation
 (Extract memory)

- Efficient Translation: Caching

Translation Lookaside Buffer (TLB) - Register

\hookrightarrow Address Translation cache

\hookrightarrow In MMU

\rightarrow PTBR

\downarrow
 Page Table

\downarrow
 PTE

\downarrow
 Find

TLB \rightarrow Hit rate of cache \uparrow

Cache : VPN \leftrightarrow PFN

Size of cache $\uparrow \Rightarrow$ Hit rate \uparrow

Size of page $\uparrow \Rightarrow$ Hit rate \uparrow

- Spatial locality : Space dimension , i.e. same space accessed
- Temporal locality : Time dimension , i.e. recently accessed.
- Hardware TLB handling - Hardware goes through the page in $O(1)$.
Software TLB handling - Hardware raises exception (TRAP)
Context Switch from Hardware to Software

@ TLB miss handlers \rightarrow stored in OS Kernel , avoiding translation
(Physical memory)

TLB \rightarrow Fully Associative cache
(No direct mapping , random)

• TLB :

- Valid bit :

Translation is valid or not \rightarrow Page Table Entry case

Process A to B \Rightarrow B should not use A's mapping \rightarrow TLB case
 \therefore A \rightarrow invalid

- ASID :

Every process has a Address Space Identifier .

Mapping : Process id \leftrightarrow ASID
 \searrow \swarrow Small (\therefore Cache)

Disadvantage : Large amount of bits (32 to 64)

- (i) Segmentation + Paging $\} \rightarrow$ Reduce Page Table size
- (ii) Encoding

Page size $\uparrow \Rightarrow$ Mapping \downarrow but Internal Fragmentation ,
lot of empty sizes will be left

(i) Segmentation + Paging \rightarrow Getting best of both ?
 \searrow \swarrow External Internal $\} \text{Issues}$
Fragmentation Fragmentation

Page Table \rightarrow stored in RAM
 \rightarrow divided into pages
 \rightarrow Stored using Pages

Page Table contains mappings, entries.

Each mapping is not important, only valid entries are

Invalid entries \rightarrow don't store, store only when it becomes valid.

Valid entries \rightarrow store

\therefore Multi-Page Table (Tree-like)

i.e. Meta-index, multiple pointers hierarchy.

* $\left[\begin{array}{l} \text{Explore 2-level Page Table} \\ \text{Study 3-level Page Table} \end{array} \right]$ *

Page Directory \rightarrow Data structure for multi-level page tables.

PDBR \rightarrow less space (\because Only valid PTEs are stored)

Disadvantage: Time overhead

12/9

- Inverted Page Table: 1 page table per process
disadvantage: Linear scan is expensive

- OS creates Memory Hierarchy.

Physical Memory \rightarrow Faster access

Less capacity

Disk \rightarrow Slower access

More capacity

On-demand Pages

- Swap Space:

\rightarrow Allocated in disk

\rightarrow divided into blocks \approx Page size

- Present bit:

1: page is in the page table

0: page is in the swap space

- Valid bit = 0 : Page is not in Physical memory \Rightarrow In disk

AKA Page Fault \rightarrow Handler

Process moved to
block state

What to swap, when & from where
AKA Page Replacement

- AMAT
- Optimal Replacement Policy (Theoretical Optimal)
Belady Replacement Policy (But Not Practical)
 \hookrightarrow Possible future access

Reduce any policy to Belady for comparison

(i) FIFO \rightarrow First in First Out

Cache size $\uparrow \Rightarrow$ no. of hits \uparrow (Belady's Anomaly)
 \hookrightarrow Not always, depends on ordering of stream of data.

(ii) LRU \rightarrow Least Recently used

- Valid bit = 1

- Access bit = 1

\downarrow Recently modified

Access bit = 0 \leftarrow Evict a page

- Thrashing \rightarrow Excessive Page fault Handlers

$\left\{ \begin{array}{l} \text{Sat} \rightarrow \text{Tut (IMP)} \quad [\text{Bonus Quiz}] \\ \text{Tues} \rightarrow \text{Tut (Attendance)} \quad [\text{MP2}] \\ \text{Fri} \rightarrow \text{None} \end{array} \right\}$

SMTP uses TCP

(Spotify - Web series)

Install Apache / nginx webserver

→ Peer 2 Peer Network:

Spotify: Midserver + Peer2Peer
(Client server)

→ Application level Protocol:

HTTP: 80

HTTPS: 443 (HTTP over secure network)

✓ connection → 3 way Handshake (∵ HTTP runs on TCP)

↘ Stateless

- 1.0 HTTP ← For fetching every connection, connection is established.
- 1.1 HTTP ← Maintain connection to fetch all objects and then terminates.
- 2.0 HTTP & 3.0 HTTP

→ Cookies:

∵ HTTP is stateless, Data stored in browser cache.

Metadata is stored in server.

Web cache:

Browser cache

ISP cache

Regional cache

Main server

} If recently changed, use cache

- Traditionally,

Client ↔ Server: Takes time
(Delay)

∴ Add cache at server.

Content distribution Networks (CDN):

↪ Servers distributed among diff. locations.

(Logics: Potential viewership + location)

Enter Deep : Build smaller clusters in more sites

Bring Home : Build large cluster in lesser sites

Cache does not contain originals

↳ Conditional GET : Data from server would only be header, no body

HTTP 2.0 → Current

HTTP 3.0 → Future, but has support now.

→ Domain Name System (DNS) :

Translation b/w name and I/P address. Allows aliasing.

Uses UDP.

- UDP call to get I/P address & HTTP call happens, i.e. then client uses HTTP to send request to server.
- DNS record
- BIND → Turns machine to DNS server → Port 53
i.e. machine acts like a directory, returns IP address
- Any request (HTTP, SMTP, etc) sent translates the hostname to IP address using DNS

- 1. Root DNS

Report to IANA

12 different Organizations are responsible for DNS creation, etc

2. Top-level Domain (TLD) : .com, .org, .ai domains

3. Authoritative Domain

- How it works : (Recursive Process), When you send a request to a host,

1. Request is sent through Port 53

2. Local DNS (ISP Provides) ← Cache kind-of

If mapping exists, fetch. Else goes to next level (Recursion)

3. Root DNS (if it has, it will give back, else send to next level)

4. TLD DNS (if it has, it will give back, else send to next level)

5. Authoritative DNS (Then, get I/P address and return to client)

6. Then, you send a HTTP request to the server.

- DNS servers stores Resource Records (RR) : (name, value, type, ttl)

tuple

Each DNS record

ttl : time after which it needs to be updated from the root DNS.

↳ Time To Live

(i) A type DNS record : (Authoritative)

Name : Hostname

Value : I/P address

$\left(\begin{array}{l} \text{AAA} \rightarrow \text{IPV6} \\ \text{A} \rightarrow \text{IPV4} \end{array} \right)$

(ii) CNAME type \rightarrow Mapping one domain to another domain

\hookrightarrow Canonical name record (alias)

(iii) NS \rightarrow Named Server

Main server itself (Authoritative)

(iv) MX type \rightarrow Mail

(mail.outlook.com)