

Memory:

Computer is **finite** → memory is finite → number representations are finite.

Memory leaks = when you forget to deallocate memory when you're not using it
128 characters can be represented with **ASCII**; Unicode is the new universal standard.

Unicode encodes the coding to facilitate to more characters.

CPU:

CPU reads instructions from memory; instructions are represented as bits and stored in a sequence of bytes. **This was the major breakthrough in computer development.**

Registers are storage in the CPU: the size of an integer, used for intermediate results.

Instructions like "load R1, 4" are a readable way of writing instructions: **assembly language**.

An **assembler** translates assembly language into computer instructions.

Index registers contain addresses used to fetch values; makes moving through strings easier.

Operating Systems:

The most widely used Operating System is **Linux**; a Unix clone invented by Linus Torvalds.

An **Operating System** is a privileged program (can access and control all of the hardware) that provides all of the services.

CPU has two modes of operation, system and user: **system call** switches from user to system.

Disk drives provide **persistent storage**: storage that doesn't disappear when machine is off.

File system: we structure raw storage into *files* and *directories*.

The shell is a program that interprets command-line instructions: command interpreter.

Shell Variables:

Assign **variable**: `$ NAME=bob`

Retrieve variable value: `$ echo $NAME`

Arithmetic: `$ A=1`

`$ expr $A + 1`

Backtick (```) means "replace the command with its output": `$ echo `echo hello``

Shell Scripts:

"myname" file	command	output
<code>#!/bin/bash</code> <code>echo \$1 is your name</code>	<code>\$ bash myname Bob</code>	Bob is your name

Pipes:

Redirect the output of a command into a file: `$ echo hello >myfile`

Take the output of one command and send it to the input of another: `$ cat myfile | wc`
1 1 6

wc	line/word/character count
cut	remove a portion of each line (eg cut out columns)
paste	paste several files together vertically
sed	stream editor
tr	transliterate one class of character to another
sort	sort a file on one or more fields
grep	search for matching lines
uniq	remove duplicate lines
fmt	format the text

`$ cat > list1` adds text from standard input (keyboard) to list1; `$ cat >> list1` appends.

`$ cat list1 list2 > biglist` directs contents of list1 and list2 into biglist.

`$ head -n2 list1` outputs first 2 lines from list1; `$ tail -n1 list1` outputs last line.

Processes and the Unix Kernel:

The **Unix kernel** is responsible for switching between programmes.

interrupt saves the running program's "state" (memory, cpu registers) & enters the kernel.

A **process** is a program with its code and data stored in the memory along with some metadata that shows who the owner is (UID), what open files it has (file descriptors), etc.

<Ctrl-Z> sends processes to sleep. **ps** lists running processes (**ps aux** prints per line); **bg** starts sleeping processes in background; **fg** starts them in foreground; **&** runs program in bg.

User IDs:

info1112:x:1003:1003:Bob Kummerfeld,,,:/home/info1112:/bin/bash

The first 1003 is the **UID**, and the second 1003 is the **Group Id**.

When an account is created, it gets a group:

adm:x:4:syslog,info1112 → "syslog" & "info" are members of the group "adm"

To find out the UID of the running shell: **id**

Superuser ID:

A process with **UID of 0** can access any file and run any program.

The superuser is given the user name "root".

File Access:

The **metadata** of a file include it's: name, owner, group, access permissions, position in the name space, location in mass storage.

A program can access a file using the **open system call**: it finds the file → sets up a data structure containing the metadata.

This data structure is referred to by a **file descriptor** (an integer).

When a program is started it is automatically given three open files and file descriptors:

0 standard input	→ redirect using: <
1 standard output	→ redirect using: >
2 standard error output	→ redirect using: 2>

Devices in the namespace:

In Unix, devices can be accessed the same ways that files can.

When a device file is opened, the OS looks at the metadata and passes it to the **device driver**.

The namespace can also specify **pseudo files** that return information about the system.

Regular expressions (grep):

grep is a command that searches for text patterns.

Flags: -i case-insensitivity; -v non-matches; -C counts only; -B lines before; -A lines after

- * (0 or more of the preceding character)
- . (any character),
- [abc] (any one of the included characters)
- [^abc] (any one character except those included)
- ^ (put at the start; string must begin with the match)
- \$ (put at the end; string must end with the match)

E.g.:

```
$ grep '^Z.us' /usr/share/dict/words
```

This searches for any line with a string starting with 'Z' followed by any character, followed by a 'u' character, and finished with an 's'.

Program:

Programs are stored on the persistent storage (e.g. disk).

Starting a program:

- operating system reads the file and generates a memory image containing code & data
- the image is read into some free memory and the metadata structures are created
- the operating system starts the process executing.

System Calls:

System calls are like a function call to the operating system; they manage processes.

Main system calls are `fork`, `exec`, and `wait`.

fork creates a copy of the process which has called, except:

- the child process has a new process ID
- the child process has the parent process ID
- various values of resource metadata (e.g. cpu time used) are reset

The child fork returns a pid of 0 and the parent fork returns the process ID of the child.

wait suspends execution of the process that called the function until one of its child processes exits. It returns the process ID of the child that has exited.

`os.wait()` and `os.fork()`:

Python examples: (note: import `os`, `time`, `sys`)

parent process	child process
<pre>pid = os.fork() if pid == 0: print ("\tHi! I'm the child process.") print ("\tsleeping for 5 seconds...") time.sleep(5) print ("\t... exit with status 99.") sys.exit(99) elif pid == -1: print ("yikes! fork failed!") sys.exit(1) else: print ("I'm the parent process") print ("waiting for child with PID", pid) wval = os.wait() print ("wait over! process ID was ") print (wval[0], "exit status",wval[1]>>8)</pre>	<pre>pid = os.fork() if pid == 0: print ("\tHi! I'm the child process.") print ("\tsleeping for 5 seconds...") time.sleep(5) print ("\t... exit with status 99.") sys.exit(99) elif pid == -1: print ("yikes! fork failed!") sys.exit(1) else: print ("I'm the parent process") print ("waiting for child with PID", pid) wval = os.wait() print ("wait over! process ID was ") print (wval[0], "exit status",wval[1]>>8)</pre>

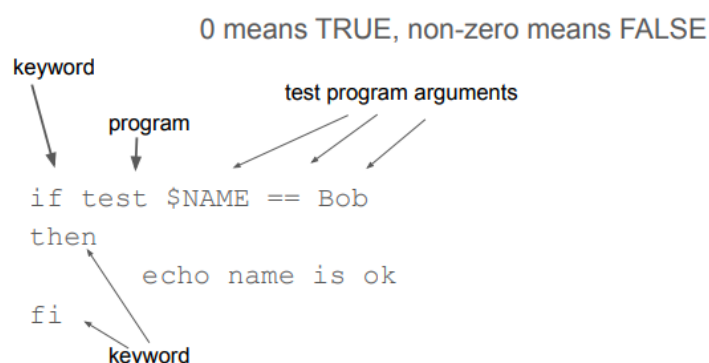
exec replaces the calling process by the new program.

Stopping a Process:

Processes can be terminated using the **kill** system call; it sends a signal (default: TERM).

Signals can be sent if the sender & receiver have the same UID or the sender is the superuser.

Shell **if** Statement:



Note: The character "[" is a synonym for "test". The character "]" is ignored by "test"

File systems:

All the Unix file systems provide the same interface.

Mount: grafts files into a particular place in your namespace.

The ability to implement your own file system (with a system module called “fuse”) allows users/developers to easily implement new services and put it into the namespace.

Programs in Memory:

The stack is for temporary variables; **the heap** is for data structures.

Processes begin at address 0; the heap builds upwards while the stack grows downwards.

If they clash, all the memory has been used.

Memory Mapping:

Since all processes can be referenced at 0, hardware in the CPU **remaps** the address spaces; the kernel of the operating system offsets all references to addresses.

There is a register that puts an upper limit on the addresses the process can use.

Virtual Memory:

There is a “**page table**” in the system that contains an entry for every remapped block of memory (called a “page”).

The table maps the **virtual address** used by the process to the physical address.

The kernel manages this table and moves pages to and from mass storage.

If a program attempts to use memory that’s mapped to mass storage, an interrupt will occur; the kernel will find the page on the disk (the “**swap area**”), read it into main memory, and point the page table to the new physical location.

If the kernel discovers there is no free physical memory available, it has to first write an existing block to the swap area to free up the memory. If this happens too often, the system will become very slow and inefficient; **mass storage is much slower than RAM**.

“shared memory” is when some processes have pages that are mapped to the same physical memory i.e. read-only data.

“copy on write” means that when a process forks, it shares physical memory pages between the processes until one or other wants to change a page.

Scheduling:

A CPU system executes one process at a time; the others wait for a form of I/O to call them.

The **scheduler** maintains an ordered queue of processes waiting to run.

Simplest scheduling algorithm: **“round robin”** gives each process an equal slice of CPU time.

Sophisticated schedulers take into account priority value.

Scheduler and Linux:

Processes in Linux have a priority value that can be seen on a `ps al` listing (under PRI).

Priorities range from negative integers (the highest priority) to positive (the lowest priority).

A normal user process starts with priority 20.

Set priority using `nice -[number]` command. Default (without number) adds 10.

Change priority of a running process using `renice -n -[number] process-id`.

Check priority: `ps -o nice process-id`.

STARTING THE SYSTEM

First Stage, Hardware:

RAM is volatile; content is lost when power is turned off.

Flash memory is persistent but can be changed by a program.

ROM is persistent but cannot be changed after manufacture; contains the initial instructions.

Second Stage:

The first instructions are executed in persistent memory from address 0.

Then there is a **jump instruction** that transfers control to program code for the next stage of the boot sequence— does more hardware checks & works out what devices are connected.

Third Stage:

The program stored in the persistent memory (the **"firmware"**) works out where to find the next stage of the program by looking at a fixed list of places stored in flash memory.

It loads it into memory and jumps to it, storing it in the **Master Boot Record**.

Master Boot Record:

The MBR is stored on the first block of a disk and less than 512 bytes.

It contains a program that reads in the next stage of boot program (in Linux it's called **GRUB**) as well as a table of device configuration.

Fourth Stage:

The boot program navigates the namespace of the file system on mass storage and finds the file containing the kernel. It will then load it into memory and start it running. There may be several versions of the kernel in the file system and GRUB works out which one to load.

Fifth Stage:

The kernel is now running. It performs final, high-level hardware checks and then loads device drivers and kernel modules as required. The kernel will then mount file systems.

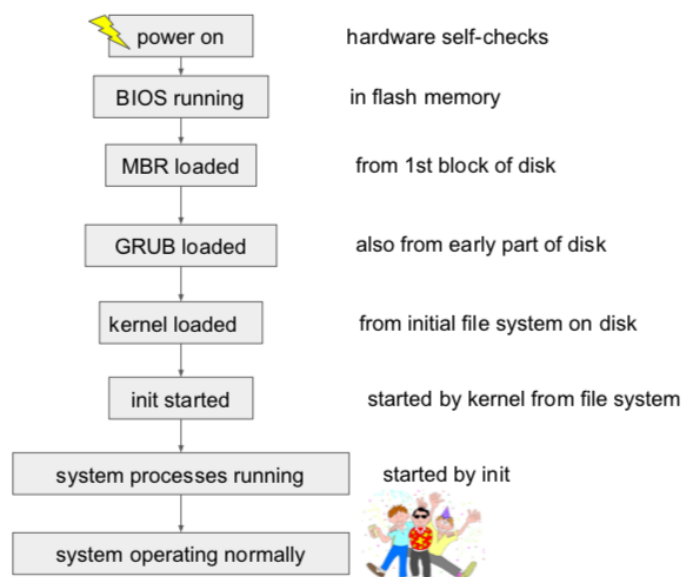
Sixth Stage:

The kernel is now ready to start real processes.

The first process is called "init". This is the only process started by the kernel.

From then on processes are only started by other processes.

Summary



Emulator Programs:

An **emulator** reads each instruction as a binary number and breaks it down into the key parts. It can do this using logical operations- 'shift', AND, OR.

Shift moves the bits left or right the specified amount; the bits at the end are lost

01100100 10001100 >> 4 = 0000 011001001000

Extract code using shift + AND.

Assume 01100100 10001100 To extract the first address:
 ↑ ↑ ↑
 0100 address 4 1000 address 8 1100 address 12
 0110 code for "add" instruction

Shift: 01100100 10001100 >> 8
 = 00000000 01100100

And: 00000000 01100100 & 0b0000000000001111
 == 00000000 00000100

This is because only 1 & 1 = 1. Also called **masking**.

Shell for statement:

keyword list of values variable list of values: all the files in the directory

```
for name in Alice Bob Carroll
do
    echo name is $name
done
```

```
for file in *
do
    echo file name is $file
done
```

General form is: **for** variable **in** list ; **do** statement list **done**

Emulators and Virtual Machines:

Emulators emulate instructions and

registers in a CPU.

We can write an emulator program to emulate any instruction set; but it will be very slow.

You may speed up the emulator code by writing in a low-level language.

Emulators are called **virtual machines** if they emulate all the system aspects (i.e. memory).

VMs are capable of running complete operating systems in the emulator.

This means you can run another OS (e.g. Linux) on Windows by running it in a VM.

Why use a Virtual Machine?

- It allows you to run a different operating system/different version of an OS
- Allows you to run older software
- Provides isolation (mostly) from bugs or malicious code
- Allows you to develop and test software in many different target environments
- Allows many small systems to simultaneously run on a single large machine

Containers:

Containers capture all system calls from an application and emulate them, tricking the application into thinking it was running in a certain environment or system.

Containers are lighter weight than full VMs and are widely supported on cloud systems.

They have many of the same benefits, e.g. isolation.

Docker is a popular container system; there is a free version.

Circuit Switching:

When computers were connected using telephone network technology; is rare now.

Send data as a continuous stream over a fixed circuit.

Advantage: each data connection has reserved bandwidth.

Disadvantages: Reserved bandwidth is wasted if no data is being transmitted;

An interruption cannot be recovered from without making a new call

Packet Switching:

Fixes the issues of circuit switching by giving up guaranteed bandwidth; is common now.

It breaks the stream up into small chunks, sends them independently through the network, and reassembles them in the correct order at the destination.

Advantages: Packets can follow different routes so they can be re-routed around congestion;

The data from different connections can share a channel.

LANs and WANs:

Categories of connections between computers: **local area networks** and **wide area networks**.

LAN can be your home, campus, etc.; WAN is global and consists of interconnected LANs.

Both use packet switching.

Local Area Networks:

The most common form of LAN uses **ethernet** technology for the interconnection.

In wireless and wired networks, the data packet is carried in an ethernet frame (a packet).

Ethernet frame: a packet (data) inside another packet (ethernet).

Ethernet frames/packets are carried from a device to the access point or switch.

Data packets hop along the route until eventually reaching their destination.

Layering: with each hop data packets are carried by a lower level network message.

Internet Packet – IP:

The data stream is divided into chunks called "internet packets" or **IP**.

IP can be up to 645536 bytes (but are typically smaller i.e. 1500 bytes).

The packet consists of a 20-byte long header (containing length, dest. address, source add...) followed by the data.

Internet Packet Addressing:

The source and destination address are 32bits/4bytes each.

An **internet address** is written as 4 numbers in the range 0-255 separated by dots; 129.78.8.1.

An **IP address** is used to identify each computer and to determine the route to a computer.

Internet Protocol:

A **network protocol** is the set of rules for controlling the packets exchanged between comps.

Network protocols have to be implemented by all the computers on the network.

The Internet has a set of these protocols that govern all the layers of the network.

When a packet is lost or corrupted, the packets may fail to arrive or arrive out of order.

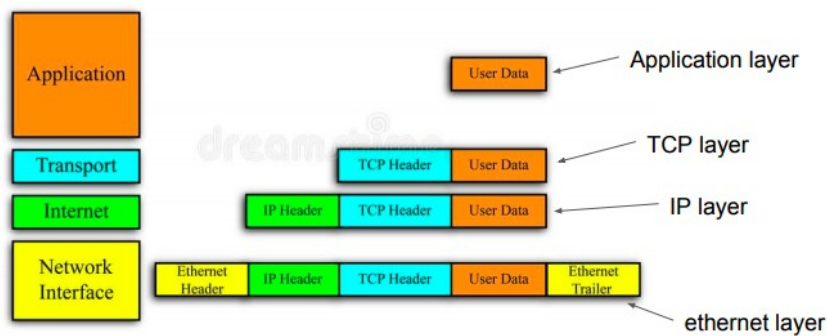
We can detect most data errors using **checksum**.

Transmission Control Protocol – TCP:

The solution is to create a protocol layered on top of Internet Packets;

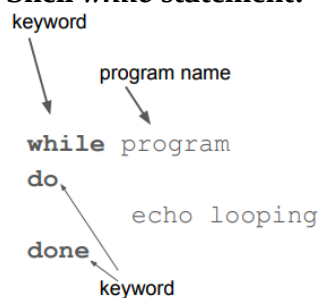
TCP: maintains a sequence number in each packet; requests retransmission of packets with errors; acknowledges packets received correctly.

TCP/IP Network Model Encapsulation



"User Data" is data for an application; these applications have their own protocols.

Shell *while* statement:



General form is: `while program; do statement list done`

Start-up Shell Scripts:

Shell programs (scripts) can be stored in a file and run like a command.

When bash starts up, it first executes commands from `/etc/profile`, then from `~/.bash_profile`, `~/.bash_login` and `~/.profile` in that order.

→ ('.' hides these files from `ls`)

When starting scripts, the file `~/.bashrc` is read first.

IP Packet Level Protocols:

The Internet packet carries higher level protocols: i.e. TCP and UDP.

User Datagram Protocol (UDP) is like TCP but offers no guarantees of delivery unless you embed a higher-level protocol.

There is also a protocol for exchanging control messages about the Internet itself: **ICMP**.

Internet Control Message Protocol (ICMP):

Provides messages that indicate the status of a connection or error.

ICMP has an echo request where the destination sends the packet back to the source.

The **ping** command verifies that a computer can communicate with another device.

An ICMP message has a **time-to-live** (no. of hops) which, once exceeded, traces the route.

LAN Routing:

A LAN is a "broadcast" network on a shared medium, e.g.

Wi-Fi: every node on the network can hear packets sent by other nodes.

Nodes decode packets sent to either its own ethernet address or a special **broadcast address**.

An ethernet broadcast address consists of all 1's.

Address Resolution Protocol (ARP):

ARP is used by a node to discover the ethernet address of other nodes.

A node sends an ARP message to the broadcast address declaring the destination IP address; the target node will hear it and reply directly with its ethernet address.

Outside the LAN:

All nodes inside a LAN have an IP address chosen from a contiguous range; this block starts with the right-hand bits set to zero; the first node has address 1 in the block.

If a destination address is outside the LAN, the node sends the packet to a **gateway address**.

LAN Addressing:

A typical block would be 256 addresses long, e.g. 129.78.8.0 to 129.78.8.255

Test if a node address is inside this LAN by using a mask that is 8 1s; if 129.78.8.0 then yes.

We can generalise this with a "routing table":

Kernel IP routing table

Destination	Gateway	Genmask	Flags	Metric	Ref	Use	Iface
default	gw-11.cs.usyd.e	0.0.0.0	UG	0	0	0	em1
129.78.8.0	*	255.255.252.0	U	0	0	0	em1

Interface to send the packet.

This is displayed with the linux "route" command.

We can add more entries to the table.

WAN Routing:

Routing outside the LAN is done with routing tables maintained by **edge systems** talking to each other using a routing protocol, e.g. Border Gateway Protocol.

The IP address range is divided into blocks called **Autonomous Systems (AS)** and information about each AS is exchanged between routers running BGP.

Internet Assigned Numbers Authority (IANA):

Addresses in the internet are managed by the **IANA**; it is responsible for top level allocation but delegates lower level allocation to regional organisations.

Running out of addresses:

IPv4 uses 32 bit addresses; we've almost run out of addresses.

Some areas have small blocks unassigned and are recycling addresses.

A common solution is to reuse addresses from address ranges that are "private".

Network Address Translation (NAT):

If an address is in private range, it needs translation to a non-private address at your router.

The router remembers which private address is used and manages this translation.

The solution, IPv6:

The real solution to address exhaustion is a new version of the Internet Protocol: version 6. IPv6 has 128 bit addresses and a better approach to routing.

Writing Shells:

We can declare any program a "shell" and write scripts that are interpreted by the program.

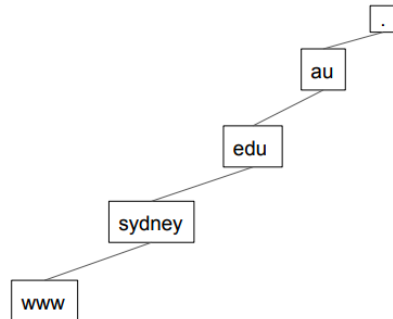
To tell the system what program to use to interpret your script: `#!/home/bob/myshell`

Languages that are not compiled are run this way: `#!/usr/bin/python`

Python: `os.walk` traverses a given directory. Returns the root, directories, and files.

Domain Name System (DNS):

DNS is the internet's equivalent of a phone book; it translates domain names to IP addresses. E.g. www.sydney.edu.au is like a file path name, with dots instead of slashes and reversed.



DNS – nameservers:

At each level in there are computers that run **nameservers** for that level and maintain information about the level below.

DNS – name lookup:

When your system wants to lookup a domain name, it asks the nameserver using the DNS protocol. The DNS protocol sends messages using **UDP**.

Dynamic Host Configuration Protocol (DHCP):

When a system is started it uses DHCP to find its IP address and the nameserver to use.

- System sends out a broadcast packet that requests config info from any DHCP server.
- The server responds with the IP address and IP address of the DNS server to use.

DNS – records:

Each record contains

Name	full domain name of the node in the tree
Type	Type of the record, ie what information it carries
Class	Class, really only IN meaning Internet
TTL	Time to live in seconds before this record expires
Data Length	Length of the data field
Data	the value of the record

DNS – record types:

Type	Description	Example
A	IPv4 address	129.78.8.1
AAAA	IPv6 address	fe80::3617:ebff:fea1:e43f
CNAME	alias	e14794.dscj.akamaiedge.net.
MX	Mail Exchanger	au-smtp-inbound-1.mimecast.com.

Advantages of DNS:

The DNS is an example of a **distributed database**.

The advantage of this is that each zone is responsible for managing names in that zone.

E.g. USYD manages `.sydney.edu.au` zone, updating DNS records when machines are added.

Application Protocols:

The upper layers of protocol in the internet where application programs communicate.

- **HTTP** (HyperText Transfer Protocol) used to fetch web pages
- **SMTP** (Simple Mail Transfer Protocol) used to send email

These protocols are lines of text that are sent to and from the relevant server.

TCP Ports:

TCP connections have an extra level of addressing called a **port**.

A port identifies an application running within a system + listens for connections to that port.

A **port number** is 16 bits (64k possible numbers).

Some are reserved for certain protocols, e.g. HTTP has port 80, SMTP has port 25.

HTTP: client → [request from] webserver (port 80) → [fetch from] web pages

SMTP: client determines mail server dest. (using DNS MX record) → makes a connection

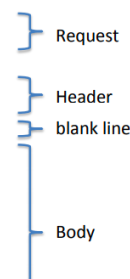
SMTP example (S: server, C: client)

```
S: 220 smtp.example.com ESMTP Postfix
C: HELO relay.example.com
S: 250 smtp.example.com, I am glad to meet you
C: MAIL FROM:<bob@example.com>
S: 250 Ok
C: RCPT TO:<alice@example.com>
S: 250 Ok
C: RCPT TO:<theboss@example.com>
S: 250 Ok
C: DATA
S: 354 End data with <CR><LF>.<CR><LF>
C: From: "Bob Example" <bob@example.com>
C: To: Alice Example <alice@example.com>
C: Cc: theboss@example.com
C: Date: Tue, 15 January 2008 16:02:43 -0500
C: Subject: Test message
C:
C: Hello Alice.
C: This is a test message with 5 header fields and 4 lines in the message body.
C: Your friend,
C: Bob
C: .
S: 250 Ok: queued as 12345
C: QUIT
S: 221 Bye
```

HTTP Example

```
Client:
  GET /index.html HTTP/1.1
Server:
  HTTP/1.1 200 OK
  Content-Type: text/html; charset=UTF-8

  <html>
  <head>
  <title>An Example Page</title>
  </head>
  <body>
    Hello World, this is a very
    simple HTML document.
  </body>
  </html>
```



Email format:

- Resembles HTTP.
- Header has lines starting with "keyword:" followed by blank line.

For network connections we wish to provide:

- Confidentiality
 - Protecting information from interception
 - Ensure information remains confidential if intercepted
- Integrity
 - Detecting whether information has been tampered with
- Authenticity
 - Ensuring we know who the sender is
 - Non-repudiation

Security when browsing the web:

HTTP web protocol has a secure version called **HTTPS**.

It is the same protocol but the data on the transport layer connection (TCP/IP) is encrypted.

Layering provides security for existing protocol. i.e. HTTP.

HTTP
Encrypted data stream
TCP/IP

In the **encryption layer** we scramble the data so that it can't be understood by an attacker, but the intended recipient can un-scramble it.

Cryptography is used to guard against interception of communication.

Encoding:

Encoding is done by an algorithm with a parameter k , called a **key** (or password).

- Keys are easier to keep secret.
- Keys can easily be changed.

To protect web transactions, we encrypt data stream with a key that the user and server know. Cipher text is $E(m, k)$; Decoder is $D(c, k)$ where c is ciphertext.

Message Integrity:

To check if a message has changed you can calculate a type of checksum called "**hash**" or "**message digest**".

This is a short value (eg 128 bits) calculated from a long message.

Different messages should have different digests.

Message digests are used to check the integrity of a message.

Cryptography:

Traditional ciphers depend on the key being known to both parties using it.

- They are called **symmetric** or **shared-secret** crypto.
- It is hard to arrange safe sharing between parties who are not in physical contact

Another approach uses secrets which are known to only one party; **Public Key cryptography**

- Also known as **asymmetric** encryption
- Much slower than symmetric encryption algorithms
- Not practical for most communication
- *E.g. RSA system*

Authenticity - Digital Signature:

If D and E can operate in either order (*i.e. RSA*) then they can be used to "sign" a message. Combine signature and encryption to protect against both interception and forgery.

Key Establishment:

For secure communication using conventional crypto:

Alice can choose a key value and send it to Bob, encrypted with Bob's public key and signed with Alice's secret key.

- the key value is a secret they share: they both know the value, but interceptor doesn't.

Public Key Infrastructure:

How can we protect the information about Bob's public key from attack, and get it securely to Alice?

- Modification attack: if attacker can change the website where Bob's public key is stored, and put a different value there
 - o Attacker could then read messages intended for Bob
 - o Attacker could prevent Bob reading messages intended for him

Bob can pass his public key to Alice through trusted intermediaries

Digital Certificates:

A data structure that certifies the ownership of a public key by a named subject

- This allows others to rely upon signatures or assertions made by the private key that corresponds to the certified public key.

Certificates are managed by "**certificate authorities**" that are trusted organisations. Certificates are signed by the secret key of the certificate authority and can be checked using the public key.

A certificate is issued by a higher level authority (to protect the public key of the authority).

→ HTTPS example:

When you request a web page using a url starting with HTTPS, the web server sends a public key to the browser in the form of a signed certificate.

This key is checked using the public key of the certificate authority.

SSH:

- SSH or Secure Shell is a program available on all Unix systems for secure communication.
- SSH has its own communications layer above TCP/IP that provides encrypted login sessions and other services.
- The simplest use of SSH is to provide a confidential session between unix systems.

- o The authentication is handled in the usual way with login program asking for a password.
- SSH can also be configured to use public key cryptography if a user stores their public key on the remote system and then uses their private key to encrypt a message.

SCP:

- SCP or Secure CoPy is a feature of SSH that allows a user to securely copy a file from one machine to another.
- The user can use password authentication or public key authentication and SSH will send the file to the SSH server on the remote machine.
 - o A user can also copy a file from a remote machine securely.

SSH Services:

- SSH is also used to provide a number of services where it is acting as a secure carrier for other protocols.
- SSH can also be used for **tunnelling** lower level protocols.
 - o This is where SSH effectively acts as a pipe for any protocol and so can carry TCP/IP. This is useful for Virtual Private Network connections.

A simple encryption scheme is called ROT13. ROT13 simply stands for Rotate-by-13.

Symmetric Encryption:

With OpenSSL we are able to encrypt and decrypt files using the openssl command.

openssl <encryption method> -in <input file> -out <output file>.

You can decrypt the file using the -d flag and changing the -in and -out flags.

GPG, Asymmetric Encryption:

The **gpg** command can be used to generate a public-private key pair.

This generates a public-private key pair: `gpg --gen-key`.

Export public key: `gpg --output public_key.gpg --export <your key id here>`

Encrypt and send message: `gpg --encrypt-files --recipient-file <your friend's public key>.gpg <file>`

Decrypt file sent to you: `gpg <file>`

Generate hash: sha256 or sha256sum. **Extract hash:** sha256 or sha2

Windows:

The first and widely used window system for Unix was "X Windows".

Command line interface: shell reads from keyboard and writes to the display via OP kernel.

Multi-window system: shell communicates with a "**window server**" that manages the display, keyboard, mouse. The window server can be broken down into parts, i.e. a window manager.

Android:

The base of Android is Linux.

Android uses an extra software layer (like the window server) for applications to access the screen and touch input.

Android is written in Java.

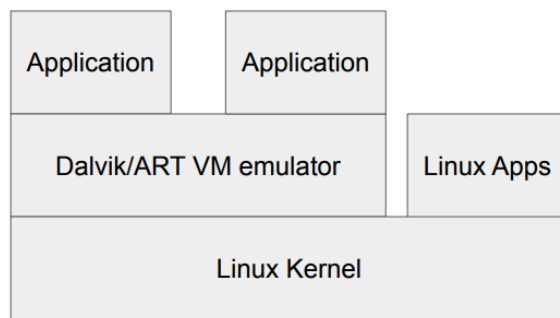
Java is translated to a virtual machine code, which is then emulated by a program.

In normal Java (non-Android), the virtual machine is called the "Java Virtual Machine"/JVM.

However, JVM is owned by Oracle and requires commercial products to license.

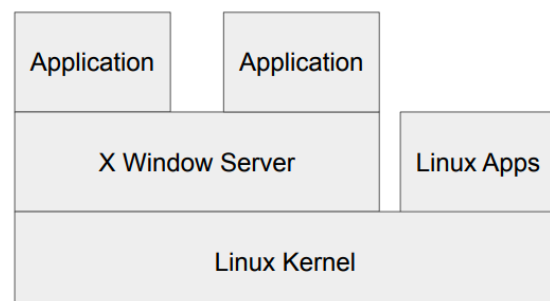
Android initially used a different architecture/virtual machine to avoid this, called Dalvik.

Original Android Architecture



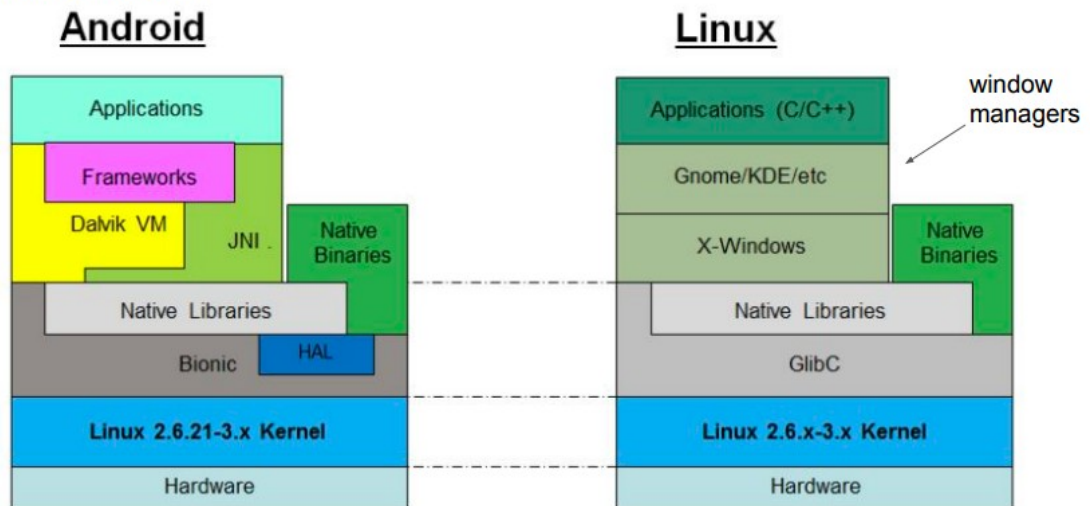
vs.

X Windows Architecture:



Today (2019) Android has moved away from the Dalvik VM and towards native binaries. This is to improve performance.

Original Android architecture



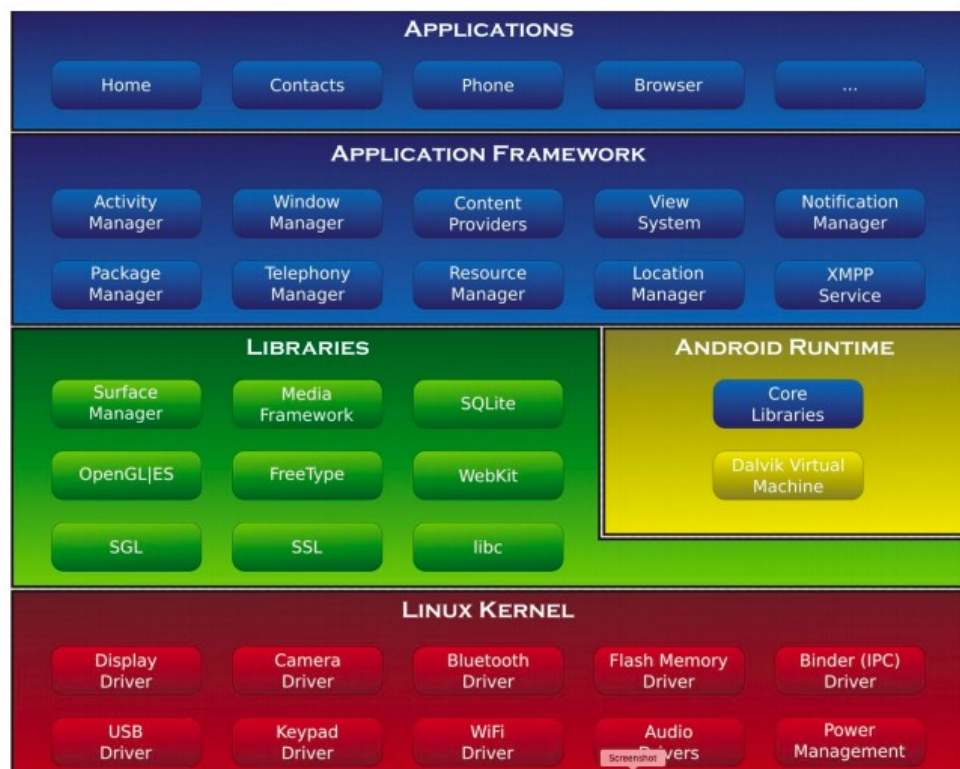
JNI is the **Java Native Interface**.

It enables Java code running in a Dalvik to call and be called by programs specific to a Linux platform and libraries written in other languages.

BIONIC is an equivalent copy of the standard C library glibc, made due to licencing restrictions.

HAL stands for **Hardware Abstraction Layer** and give a standard interface to screens and touch input. This is necessary because different phones may have different touch hardware.

Current (2019)
Android
Architecture



Terminal Commands

Basics

cd	Change directory
mkdir	Make directories
ls	List directory contents
rmdir	Remove directory
rm	Remove
mv	Move (also used to rename files)
cp	Copy
pwd	Print Working Directory

..	Parent Directory	cd ..
.	Current Directory	./program
-	Previous directory	cd -
~	Home directory	cd
/	Root directory	cd /

Kill commands:

kill -9 -1 kills everything
<Ctrl C> kill signal “-SIGINT”

Key: PID = Process ID

Shortcuts

- CTRL + A, moves the cursor to the start of the command string
- CTRL + E, moves the cursor to the end of the command string
- CTRL + K, deletes from the cursor to the end of the command string
- CTRL + U, deletes from the cursor to the start of the command string
- CTRL + LEFT_ARROW , moves the cursor to the beginning of a word
- CTRL + RIGHT_ARROW , moves the cursor to the end of a word
- CTRL + W, cuts from the cursor to the start of a word
- CTRL + Y, pastes from the cursor to the end of a word
- UP_ARROW, retrieves previous command that has been executed
- DOWN_ARROW, retrieves next command that has been executed

Access rights

- **chmod**

u	Owner
g	Group
o	Other (world)
+	Add permissions
-	Remove permissions
=	Set permissions
r	Read access
w	Write and delete access
x	Execute and access directories

E.g. -rwxr-xr-x 6 hadil staff 400 20 Aug 17:37 myscript.sh

User rights: rwxr; Group rights: xr; Other rights: x

File: myscript.sh

chmod g = myscript.sh removes group rights

chmod u = r myscript.sh sets user rights to read

chmod a+w myscript.sh adds write rights to all