Network & Operation Systems Essentials

Dr Nikos Ntarmos

Joao Almeida-Domingues[*]

University of Glasgow

September 24th, 2018 – December 4th, 2019

Contents

These lecture notes were collated by me from a mixture of sources , the two main sources being the lecture notes provided by the lecturer and the content presented in-lecture. All other referenced material (if used) can be found in the *Bibliography* and *References* sections.

The primary goal of these notes is to function as a succinct but comprehensive revision aid, hence if you came by them via a search engine , please note that they're not intended to be a reflection of the quality of the materials referenced or the content lectured.

Lastly, with regards to formatting, the pdf doc was typeset in LaTeX, using a modified version of Stefano Maggiolo's class

[*]2334590D@student.gla.ac.uk

1 Networks

1.1 *Introduction - Networked Systems*

1.1 definition. Networked System a collection of autonomous computing devices that exchange data to perform some goal

In this first part of the course we'll focus on 3 key aspects of these systems (1) how information is exchanged between the different devices involved ; (2) how we can build larger networks by linking devices ; (3) how systems communicate amongst themselves

1.2 definition. Signal a function which conveys information

1.3 definition. Communication Channel component of a a data transfer system responsible for carrying the signal

1.4 definition. Information Entrophy how much useful information a message is *expected* to contain

Claude Shannon the father of *Information Theory* showed that the amount of information that can be coded into a message could be quantified, and is known as *Information Enthropy*. Shannon stated that a data transfer system is composed of three parts: a source, a communication channel and a receiver. He identified the main problem within the system was to make sure that the information passed over the channel could be successuflly *recreated* by the source.

This encoding and decoding of messages can be done in several ways, some introduce more noise than others, but they all follow the same process of taking some form of physical signal (e.g. a wave) and converting it into some sort of simplied form of itself and then recreating it at the source end

Extra Information Enthropy Formal Definition

If we take X as the set of messages $\{x_1, \ldots, x_n\}$

1.5 definition. Analogue Signal a smooth continuum of values

1.6 definition. Digital Signal a discrete sequence of values

The simplest analogue signal is when information is encoded directly using amplitude (e.g. AM radio), however of particular interest to us is the process of converting analogue signals to digital, which can be done for any

analogue signal. (see Physical Layer)

1.7 remark. the the rate at which the signal must be sampled for accurate reconstruction is given by the sampling theorem

Switching

1.8 definition. Coding the act of mapping information to symbols

1.9 definition. Link the combination of a signal with a channel

1.10 definition. Hosts receivers and sources

1.11 definition. Network a collection of connected links

Within a networked system, information flows via channels forming links which connect hosts. The devices connecting the links are called *switches* or *routers* depending on the type of network. This *network switching* is responsible for determining how the information flows through the network and can be setup so that there are dedicated connections between hosts - *circuit switching* - or by splitting the messages into smaller packets before transmission allowing several hosts to share the same channell - *packet switching*

1.12 definition. Circuit Switching a dedicated circuit between hosts

1.13 definition. Packet Switching a shared link where messages are split into packets before transmission

The main trade-off here is between capacity and availability. For example, traditional phone networks are circuit switched (the very first ones had actual humans switching the channels and connecting hosts) which means that the two hosts requested a channel and they had guaranteed capacity over that channel while the connection was active, but it also meant that if some other hosts needed to use any part of the same link then their connection would be refused.

The internet on the other hand, is packet switched, by breaking the messages apart into smalled chunks hosts can share links the catch here being that though connectivity is guaranteed the capacity/speed is dependent on how many users are using the same channel.

1.2 *Protocols*

The different building blocks of a network presented above allow for the transportation of information, but is is the use of protocols which provide the se-

mantics. For a message to be decoded the parties involved must agree on some sort of well-defined syntax, so that noise can be separated from meaningul information, this is precisely the role of the various network protocols existing at all levels within a network.

1.14 definition. PDU stands for protocol data unit, and is the basic unit of information for any given protocol

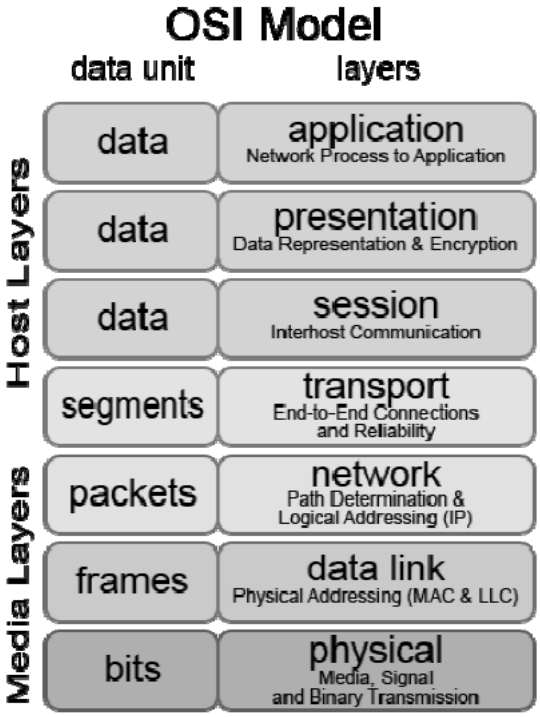
PDUs can be textual where rules of syntax and grammar are used to implement behaviour (e.g. HTTP), or binary where similar appropriate rules are used (e.g. TCP/IP). It is the role of PDUs to define what messages are legal to send, but is up to protocol semantics to define when to send them and what should be expected in response

Layers

Communication systems are tipically organised into layers, which reduces complexity at each layer's level. Peers on the same layer, use that layer's protocol to communicate using services provided by the well-defined interfaces of the lower layers

1.15 definition. OSI Model stands for *Open Systems Interconnection Model* and is a conceptual model that characterises and standardises the communication functions of a telecommunication or computing system without regard to its underlying internal structure and technology.

A design tool used widely to model layered communication channels is the *OSI model*. It is merely a design tool, real implementations are more complex and usually the boundaries between layers are not so well defined.



1.3 *OSI - Physical Layer*

The physical layer is concernerd with the transmission of raw data bits. In order for this to be possbile, the information needs to be transformed and encoded and a decision on the best medium for the job (e.g. cables, fibre optic etc.) and their phyiscal properties needs to be taken.

Transmission Channels - Enconding & Modulation

1.16 definition. Wired Data Trasmission the signal is transmited over a cable and is *directly* enconded onto the channel, by varying the voltage/light intensity

1.17 definition. Wireless Data Transmssion the signal is transmitted without the aid of an electrical conductor, most commonly using radio waves and some kind of modulation

A signal can travel with or without the aid of an electrical condutor, if it is directly encoded into a cable, then one of several *enconding schemes* can be used in order to change the signal into discrete pieces of data (e.g. bits).

High $\approx [3,5]v$ *, and Low* $\approx [0,3)$
NRZ : Non-Return to Zero

- **NRZ :** 1 – High ; 0 – Low

- **NRZ Inverted :** 1 – Change ; 0 – Constant
- **Manchester :** 1 – High-Low ; 0 – Low-High

To do (??)

Alternatively one can encode information onto a channel by varying the properties of the carrier signal via a modulating signal, a process know as *modulation* which allows the same channel to be shared by different signals

To do (??)

Bandwith, Capacity & Noise

1.18 definition. Bandwith determines the frequency range it can transport

1.19 definition. Sampling Theorem states that to accurately digitise an analogue signal, $2H$ samples per second are needed, where H is the bandwith in Hz

1.20 definition. Signal-to-Noise Ratio the ration between signal power and noise floor, typically quoten in $\text{dB} = 10\log(\frac{S}{N})$

The bandwidth of a channel is determined by physicial limitations of the channel, and given the existence of noise in the real worl, the *Signal-to-Noise* ratio and the bandwidth represent the fundamental limits for the rate at which information can be transmitted

1.21 remark. The maximum transmission rate of a channel grows lograithmically to the SNR

Extra Theoretical Maximum Transmission Rate

$$R_{max} = 2H\log_2 V$$

where:

$R_{max} =$ max trasmission rate in bits/s

$H =$ bandwith in Hz

$V =$ # of discrete values per symbol

Extra Shannon's Theorem

$$R_{max} = H \log_2(1 + SNR)$$

Summary

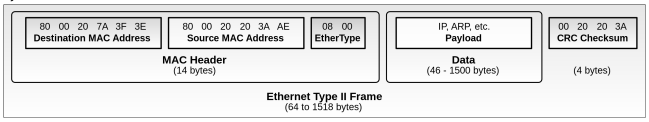
- **PDU :** bits
- **Function :** transmit a sequence of bits over an analogue channel
- Information can be encoded directly into the channel or the signal can be modulated
- Physicial limitations cap the transfer rate

1.4 *OSI - Data Link Layer*

The main purpose of the data link layer is to arbitrate access to the physical layer and turn the raw bit stream of data into a structured communications channel with the goal of transferring data between nodes attached to the same physical cable. There are several services provided by the DDL, the main ones being: *Media Access Control (MAC addressing) , Error detection and/or correction, Framing*

Framing

Frames are essentially fenced packets, which signal to the next layer the significant part of the data , i.e the *payload* and often whether that data was corrupted via some sort of error code or *checksum*. The DLL add the final tail and header to the payload and is therefore responsible for the last bit of data *encapsulation*



Error Detection & Correction

Though rare in wired systems, it is quite common for noise to cause bit errors in the data being transmitted. If the DLL implements error detection then it must add a *error detection code* as an header of the frame, the simplest one being the *parity code*. When the receiver gets the packet it uses the same

function/rule to recalculate the code, if it doesn't match the frame is either discarded or corrected

1.22 definition. Parity Code an error detection code which is able to detect single bit errors. It works by adding all the bits, and checking the parity of their sum

1.23 definition. Checksum works similarly to the parity code, but is able to handle some multiple bit errors by using a 16-bit one's complement checksum

1.24 definition. Cyclic Redundancy Code a more advanced algorithm, commonly used in the DDL which can check if the bits are out of order. It relies on the remainder of a polynomial division of the data being sent

MAC

MAC is needed to determine which machine gets access to the channel when multiple hosts try to access it simultaneously. Because if that happens then the signals are said to *collide* and will overlap, resulting in an unreadable/garbage message.

1.25 definition. Contention-Based MAC a system is contention-based if multiple hosts share a channel in a way that can lead to collisions

CB MAC deals with collision in one of two ways:

1. **ALOHA** is the simplest protocol developed at the University of Hawaii in 1970. It tries to transmit whenever data is available, and if a collision occurs the frames are destroyed and the node will wait for a random amount of time before retransmitting, repeating until successful.
2. **CSMA** listens to the channel before sending, if it hears no traffic then it starts transmitting. Note however, that if the message takes time to reach the node , i.e. if there's a high *propagation delay* then there is an increased probability of a collision occurring mid-transit, an improved algorithm is **CSMA/CD** where the sending node keeps listening even during sending, if a collision occurs then both stations cease transmission immediately. This is an improvement because even though the frames are still corrupted, it saves time and bandwidth by reducing the time the channel is blocked due to collision.

The back-of interval between retransmissions is also random, but should increase with the number of collisions to reduce congestion

Summary

- **PDU :** series of bits - *frame*

- **Function :** Physical addressing , Framing, Error Detection
- Last encapsulation of data step ; adds an error detection code and some other meta data for framing in the trail of the packet
- Simpler error detection codes rely on summing up the bits of data and either checking their parity or sum
- Contention-Based MAC handle collisions by listening to the channel before sending the data and/or during

1.5 *OSI - Network Layer*

1.26 definition. End-to-End Principle is a network design principle whereby application-specific features reside in the end nodes, rather than being distributed across the network

The Network layer is responsible for end-to-end delivery of data and is therefore the first end-to-end layer in the OSI. This is important because networks can be of enourmous sizes, and since implementing a function always has a cost, if that function is implemented across the whole of the network that cost quickly multiplies. If instead the system is concerned with achieving reliability of communication above a certain threshold, it is more efficient to implement processes for checking for correctness and completeness at the end hosts rather than at the intermediary nodes, since end-to-end reliability is not perfectly aligned with the reliability of intermediary processes

1.27 definition. Internet set of interconnect networks

1.28 definition. Autonomous System a network within a larger network which is administered separately by a single organisation who is responsible for making independent policy and technology choices

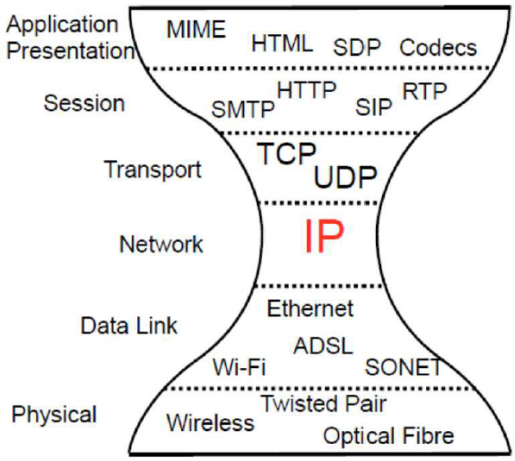
The basic components of an internet are:

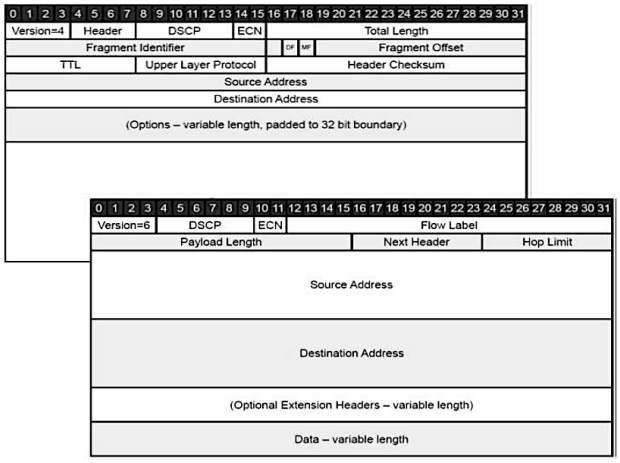
1. The existence of a common end-to-end protocol, which provides a single seamless service to the transport layer
2. A set of intermediate gateway devices, i.e routers, which implement the protocol and hide differences in link layer technologies by performing the usual services provide by that layer

1.6 *Network Layer - Internet Protocol*

The IP provides an abstraction layer, including a *simple, best effort, connectionless* packet delivery service. Which means, that it does not guarantee correctness or delivery, but it allows for addressing, routing, fragmentation and reassembly of packets.

The IP service model does not require a connection to be setup first, instead it just sends the available packets and anything which cannot be delivered is discarded. This means that it favours simplicity over assurance and correctness, leaving that for the layers above.





Addressing

IPv4 addresses are 32 bits (4×8), while IPv6 are 128 bits (16×8), one part of the address identifies the network while the other identifies the host (which part is which may vary across different networks). Each network interface must have an unique address, however virtual IP addresses are sometimes

used to give the illusion of privacy

1.29 remark. IPv4 addresses are deemed insufficient for the near future, so IPv6 was created. It has been slowly deployed since 2000 given the costs of replacing hardware and updating current software

1.30 definition. netmask describes the number of bits reserved for the network part of the address

1.31 definition. network address the host part of the address is set to 0

1.32 definition. broadcast address the host bits are set to 1

IP address: 130.209.241.197 $\Rightarrow$ 11010001 11110001 11000101
Netmask: 255.255.240.0 $\Rightarrow$ 11111111 1111111 11110000 00000000
Network: 130.209.240.0/20 $\Rightarrow$ 10000010 11010001 11110000 00000000
Broadcast: 130.209.255.255 $\Rightarrow$ 10000010 11010001 11111111 11111111

1.33 remark. a host with several network interfaces will have one IP address per interface (e.g Ethernet and WiFi interfaces in a single machine)

Fragmentation

1.34 definition. Fragmentation process which breaks packets into smaller fragments so that a layer with a smaller maximum transmission unit can receive them

The link layer has a MTU, hence before sending the packets to it, the IP breaks the packets apart into smaller frames. The process is reversed when receiving, and this can be achieved by including metadata with each frame - the *fragment identifier* , the *DF, MF flags* and the *fragment offset*. The DF flag is set if the packet is not to be fragmented, the MF flag lets the node know that it should expect more fragments part of the same packet.

1.35 remark. Note that both the last fragment of a packet as well as an unfragmented packet have MF set to true. These two cases are differentiated by setting the offset to a non-zero value in the first case

Loop Protection

In order to ward against loops, packets include a value which sets the maximum number of hops allowed in the network. With each hop this value de-

IPv4 - TTL ; IPv6 - Hop Limit

creases and if 0 is reached the packet is discarded

Transport Layer Protocol Identifier

IPv4 - Upper Layer Protocol ; IPv6 - Next Header

The TLPI is responsible for identifying the protocol used by the Transport layer above, in order to pass the data to the correct one.

2 OS - Intro

2.1 definition. Operating System a program which acts as an intermediary between the user and the hardware

The OS is the software which allows the user and higher level applications to communicate with the metal. It serves 3 main purposes:

- Usability
- Program Execution Scheduling and Control
- Efficiency Allocation of Resources

2.1 *Core Components*

2.2 definition. Kernel the program running at all times which is responsible for providing and API for the interaction between hardware and higher level programs; performing such tasks as reading and writing data to memory and determining how data received from and sent by *I/O* devices is interpreted.

2.3 definition. System Programs are programs which despite being associated with the OS do not form part of the kernel, they provide a platform for apps

2.4 definition. Application Programs are programs which do not interact with the OS directly

2.5 definition. BIOS the *Basic Input Output System* is a special piece of software which comes preinstaled in the system's board ROM and is responsible for checking that the hardware is operation and loading the OS

2.2 *Computer System Organization*

2.6 definition. Computer System can be defined as the interaction between four main components: *Hardware, OS, Application Programs, Users*

A modern computer system consists of 1+ CPUs and a number of device controllers which interact via a common bus that provides access between components and shared memory. It is the job of the device controller to transfer data between the devices and the local buffer storage which it maintains along with a set of special-purpose registers.

The CPU and the device controllers can execute in parallel, competing for memory cycles. The three key aspects of the computer-system - Interrupts, Storage Structure, I/O Structure - are deigned in such a way so as to allow for memory to be used as efficiently as possible

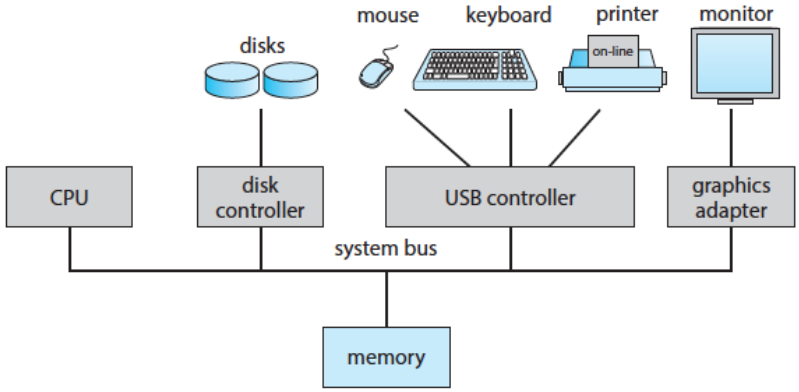


Figure 1.2 A typical PC computer system.

CPU Interrupts

A typical operation of a program performing I/O:

1. Device driver loads the registers in the controller
2. Controller inspects data in order to know what action to execute
3. Controller transfers data to local buffer
4. Controller informs driver that operation has finished
5. Driver gives back control to OS

The controller accomplishes 4 via an *interrupt*. When hardware triggers an interrupt, the CPU stops the execution of all programs and it transfers execution to a fixed location which will usually contain the starting address for the interrupted service so that execution can restart when the interrupt routine finishes.

Different architectures have different interrupt mechanisms, but they all share several essential functions, such as the transfer of control to the appropriate service routine. In order for this transfer to occur quickly a table of pointers to different interrupt routines - *interrupt vector* - is kept in low memory and is used instead of an extra routine whose sole purpose would be to inspect the interrupt information. In this way the routine is called indirectly via the table.

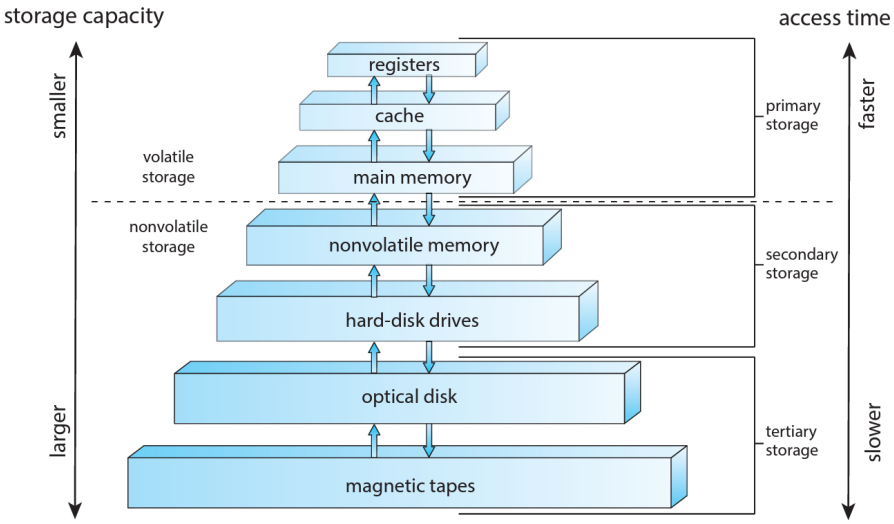
For implementation details see 1.2.1.2

vector number	description
0	divide error
1	debug exception
2	null interrupt
3	breakpoint
4	INTO-detected overflow
5	bound range exception
6	invalid opcode
7	device not available
8	double fault
9	coprocessor segment overrun (reserved)
10	invalid task state segment
11	segment not present
12	stack fault
13	general protection
14	page fault
15	(Intel reserved, do not use)
16	floating-point error
17	alignment check
18	machine check
19–31	(Intel reserved, do not use)
32–255	maskable interrupts

2.3 *Storage Structure*

When looking at a storage system one can pick out three main characteristics of importance, whose importance differs depending on their function and at which level of the system they are to be implemented, and which compete against each other. These are: *Speed, Cost, Persistence/Volatility*

Devices which need to process information immediately like the registers in the CPU are more costly,have a lower storage capacity and are in general volatile. In contrast, long-terms storage devices like the hard disk, or a flash drive are fairly inexpensive but their read/write speeds are significantly slower.



2.7 definition. Caching the copy of information into faster storage systems

2.4 *I/O Structure*

A large chunk of OS code is dedicated to I/O management. Given the large amounts of data being moved by some I/O devices, the interrupt-driven cycle can produce high overhead instead direct memory access is used.

2.8 definition. Direct Memory Access a resource-conserving and performance-improving operation for device controllers which allows devices to transfer large amounts of data directly to and from main memory, effectively bypassing the CPU and using instead a single purpose processor - *DMAC*.

3 Computer-System Architecture

The organization of a system can be categorized according to the number of general-purpose processors used.

3.1 definition. Core the component that executes instructions and registers for storing data locally

3.2 definition. CPU The hardware that executes instructions

3.3 definition. Processor A physical chip that contains one or more CPUs

Single-Processor

The SP architecture uses a single CPU with a single processing core along with other device-specific processors which can only run a limited set of instructions.

3.4 remark. This type of architecture is no longer present in modern computers

3.1 *Multiprocessor*

3.5 definition. Multicore a single CPU with more than one core who usually share some low-level memory

3.6 definition. Throughput the amount of information which passes through the system

Modern computers have 2+ processors each with a single core CPU. The processors share the computer bus, and may also share the clock, memory, and peripheral devices.The main advantage of this type of architecture is increased throughput, i.e. the higher number of processors allow for more data to be processed in less time

3.7 remark. Increasing the number of processors by N does not imply a N-fold increase in speed, since some of the processing power will be lost as overhead to tasks which make sure that the processors behave correctly when working together and sharing resources

There are two main types of multiprocessor systems - *Symmetric* and *Asymmetric*, the former being the most common. They differ in the way tasks are split amongst different processors

Symmetric

In SMP each CPU processor performs all tasks, each CPU containing its own set of registers and private cache but all processors share access to the main memory. This has the advantage that many processes can run simultaneously. The downside of this type of system is that inefficiencies can occur by overloading one CPU while the other sits idle.

N processes can be running in a system with N CPUs

Asymmetric

Was the only way to handle multiprocessor systems before the invention of SMP. It is characterized by the fact that CPUs are not treated equally, i.e dif-

ferent CPUs prioritize certain tasks , like say I/O while others are reserved for system operations.

3.2 *Clustered*

The rationale behind clustered systems is similar to that of the multiprocessor systems, but we expand vertically, i.e instead of looking at multiple CPUs within a single computer, we use several computers within the same network.

Clustered systems are very common nowadays and advancements in speed and computing power are linked to their pairing with SANs which are essentially local, dedicated data-storage networks. By having multiple machines connected to the same SAN if one machine goes offline another can quickly pick up the slack.

3.8 definition. SAN describes a combination of hardware and software which essentially work together to form a dedicated storage area network

3.9 definition. hot-standby mode one system runs parallel to another, in case the primary fails the backup quickly takes over

3.10 definition. Asymmetric Clustering one machine in hot-standby mode

3.11 definition. Symmetric Clustering multiple nodes running applications and monitoring each other

3.3 *Process Management*

3.12 definition. Process a program in execution

3.13 definition. Single-Thread single program counter which specifies the location of the next instruction to execute

3.14 definition. Multi-Thread one program counter per thread

In order to manage processes the system needs to be able to:

Create and delete user and system processes

Suspend and resume processes

Provide mechanisms for process synchronization, communication and deadlock handling

3.4 *Memory Management*

Memory management is responsible for determining what is in memory, this includes:

Tracking which parts of memory are in use and by whom

Deciding which processes and data to move into and out of memory

Allocate memory space as required

3.5 *Storage Management*

Storage management concerns itself with the management of the file-system, in particular how the OS handles free space and how that free space is split into files and directories with appropriate access permissions so that the user can easily access them.

4 System Calls

4.1 definition. System Call the software, typically written in a high-level language (e.g C), which sits between the hardware and the applications

4.2 definition. User mode restricted level where most programs run without direct access to memory or hardware

4.3 definition. Kernel mode unrestricted program execution

Programs running on user mode are provided an API by the OS which provides the application which mediated access to the hardware. A classical example is a basic task like writing to and reading from a file.

4.1 *Parameter Passing*

In order to pass parameters to the system call we can:

- use registers
- stored them in memory and address of the block passed in register
- push onto the stack by the program and pop by OS

	Windows	Unix
Process control	CreateProcess()	fork()
	ExitProcess()	exit()
	WaitForSingleObject()	wait()
File management	CreateFile()	open()
	ReadFile()	read()
	WriteFile()	write()
	CloseHandle()	close()
Device management	SetConsoleMode()	ioctl()
	ReadConsole()	read()
	WriteConsole()	write()
Information maintenance	GetCurrentProcessID()	getpid()
	SetTimer()	alarm()
	Sleep()	sleep()
Communications	CreatePipe()	pipe()
	CreateFileMapping()	shm_open()
	MapViewOfFile()	mmap()
Protection	SetFileSecurity()	chmod()
	InitlializeSecurityDescriptor()	umask()
	SetSecurityDescriptorGroup()	chown()

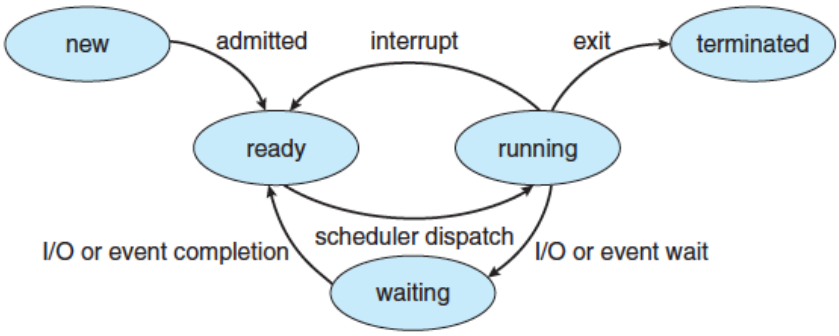
4.2 *Program Execution*

As we've seen above the difference between a program and process is that a process is a program being executed. Note than that one program can spawn several processes, i.e for the same file on disk (the written code) multiple requests to resources might happen (e.g multiple tabs on a browser)

Process States

4.4 definition. Ready Queue an array of active processes waiting to be run

Once execution of a program starts a request to the OS is made and the process is added to the *ready queue*. It will then be transitioned into the running state, it can then exit and terminate or can be put into waiting state and back into the ready queue until resources are available for it to finish execution.



4.5 definition. PCB is a data structure used by the OS to store all the information associated with a process

For every process the OS keeps a PCB containing all information associated with it, such as state, numbers, counter etc.

4.3 *Process Creation*

4.6 definition. Process Tree a hierarchical representation of processes

4.7 definition. Boot Loader is the program stored in the ROM responsible for loading data and programs into RAM

4.8 definition. Zombie Process a child process which is not *reaped* by its parent upon termination, hence it still exists in the process table

4.9 definition. Orphan Process a child process which is inherited by `init` because it is still running despite its parent having terminated

Processes are spawned by other processes, creating what is know as a *process tree*. When a machine starts there is a single process , the *boot loader* which is responsible for loading the OS into the main memory, this is usually done in multiple stages where simpler programs load increasingly more complex ones, a process know as *chain loading*.

The system calls responsible for the creation of processes are:

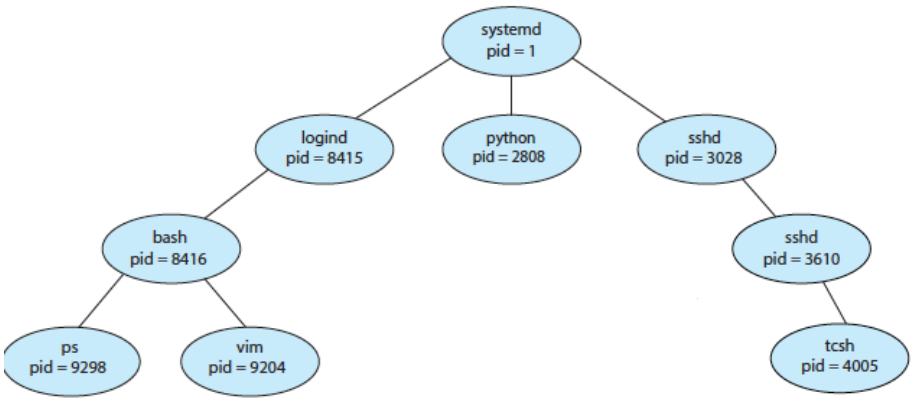
`fork(2)` - takes a process and creates a copy of it

`exec(3)` - family of functions which takes one of the processes and replaces its code with another program's code

`wait(2)` - stops the running process and waits for one of its children to terminate

By default parent and children processes run concurrently, and the parent waits for the children to terminate. There are multiple level of resource sharing, between sharing all to none.

4.10 remark. It could happen that a parent process may fail to *reap* its children state after execution, in which case a *zombie* process is created, which keeps using resources in the background. If the opposite happens, i.e if the parent finishes first then the child process becomes *orphan*



5 Inter-Process Communication

Processes can be independent running in isolation or cooperative sharing data. Cooperating processes can be affected by other processes, which can lead to a chain failure. However, cooperating processes can often be more efficient, since they share information, are more modular and more convenient to run.

5.1 *Communication Models*

It follows that if two processes are cooperating then they need to be able to communicate with each other. There are two possible modes of communication, *message passing* and *shared memory*. In the former processes communicate by sending calls to each other via a message queue in the latter a space of shared memory is allocated where both processes can read and write.

Both processes are often implemented within the same system. The shared memory model is particularly useful when large amounts of data need to be shared; it is also faster since it only uses system calls to establish the region in memory. The message passing model on the other hand is easier to implement

in distributed systems, hence it is usually applied when small amounts of data need to be shared.

Message Passing

A message passing model has to provide at least two operations `send`, `receive`. There are however several factors to consider when implementing these:

- Directly : If the communication is implemented *directly*, then processes much name each other, if both the receiver and sender are named then the communication is said to be *symmetric*. On the other hand, if only the recipient is explicitly named then one calls it *asymmetric*.
- Indirectly : If the communication is implemented *indirectly* the messages are sent and received via *mailboxes*. Unlike before, links are not necessarily $1-1$ since many processes can share a mailbox. Once a mailbox is created by an initial process, other processes access to the mailbox can be mediated via system calls.
- Synchronization : if when performing a `send, receive` the actor is blocked, then we say that the process is *synchronous*, if the opposite is true then it is *asynchronous*
- Buffering : The size of the queue can be either o, in which case sender must wait for receiver; may be bounded by n, in which case the sender must wait if queue is full , or may be unbounded where the sender can keep adding to the queue without restrictions.

Shared Memory

5.1 remark. See future lecture on memory management

6 Process Scheduling

6.1 definition. I/O Bound a process which spends most of its time doing I/O instead of computations

6.2 definition. CPU Bound a process which spends most of its time doing computations

6.3 definition. Multiprogramming when theres a process running at all times

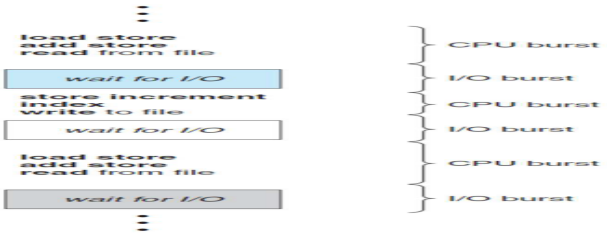
6.4 definition. Time Sharing switch among several different processes quickly enough so as to give the impression of parallel execution

6.5 definition. Long-Term Scheduler is responsible for maintaining a healthy balance between IO and CPU bound processes in memory

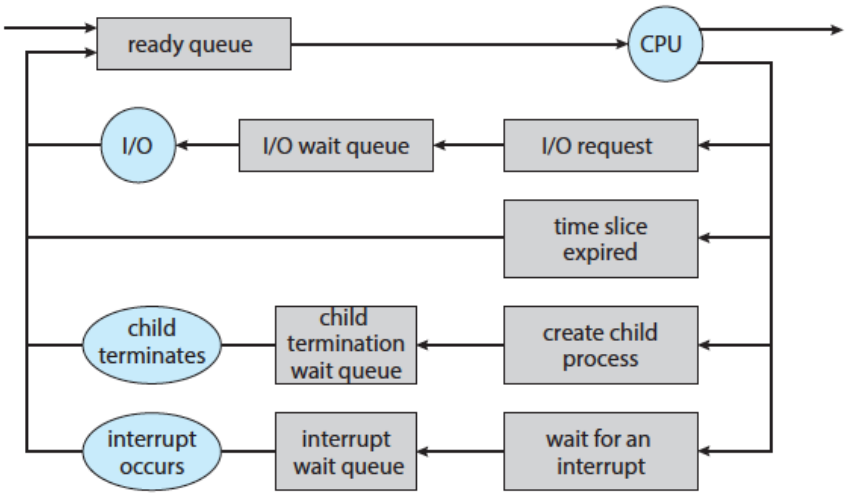
6.6 definition. Medium-Term Scheduler swaps processes from active to passive and vice-versa

6.7 definition. Short-Term Scheduler selects which process to execute next

Even with multiple cores, if there are more processes then cores then some processes will need to wait before they're able to run. Hence, there is the need for some sort of scheduling of processes. The main goal of the scheduler is to maximize CPU utilization at all times, in order to achieve this there exist several types of schedulers and several queues need to be maintained in order to keep track of each process state.



The *job scheduler* concerns itself with loading programs into memory, the *medium-term* one performs possibly necessary swaps between active/holding states and the *CPU scheduler* decides which process to execute next. These 3 schedulers can move processes among the various queues : all processes exist first in the *job queue*, once they are in RAM and ready to execute they are put into the *ready queue*, if the process is waiting for a particular device then it is put into that device's queue.



6.1 *CPU Scheduler*

6.8 definition. Non-Preemptive Scheduling let the process run its course without interruption

6.9 definition. Preemptive Scheduling the process decides when the process should yield the CPU

As discussed a process may switch several times between one of the queues and the CPU, though CPU bound processes will spend more time executing it is unlikely that the scheduler will grant long chunks of uninterrupted time to these processes. Instead, processes in the core are constantly interrupted and switched.

6.10 remark. It is common for the CPU scheduler to execute much more frequently than once every 100ms

When to decide when to schedule a process to run in the core may take place at any of 4 stages of a life of a process:

Running $\rightarrow$ Waiting (e.g I/O request)

Running $\rightarrow$ Ready (e.g interrupt)

Waiting $\rightarrow$ Ready (e.g I/O terminates)

Termination

6.2 *Dispatcher*

6.11 definition. Dispatch Latency is the time it takes the dispatcher to switch processes

It is the job of the dispatcher to give control of the CPU to the process selected by the scheduler. In order to do this the scheduler must be able to *switch context*, *switch to user mode* and *jump back to the proper instruction in the user program to resume execution*

6.12 remark. Given that the dispatcher is called every time a process is swapped and the time it takes to do so is pure overhead, it is important that it be as fast possible.

6.3 *Scheduling Criteria*

The best algorithm for a particular process may not be the best for another set of processes. It is important that one is aware of this fact when choosing an algorithm. In order to compare them, the following criteria can be used:

CPU Utilization : maximize , in practice somewhere between $40\% - 90\%$

Throughput : maximize , for any given time unit, one aims to complete as many processes as possible

Waiting Time : minimize , waiting time is wasted time

Turnaround Time : minimize , the aim is faster total completion times

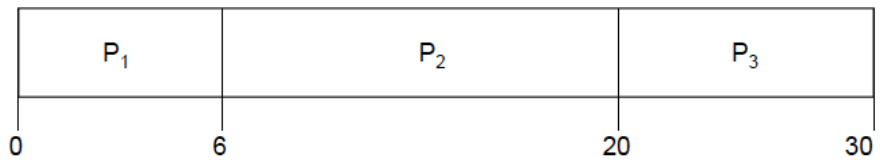
Response Time : minimize , in an interactive system this criteria can be better than turnaround time, we measure when the first response is issued

6.13 example. Find below the application of each algorithm for the following processes:

Process ID (PID)	Arrival Time	CPU Burst Time
P_1	0	6
P_2	4	14
P_3	5	10

6.4 *FIFO*

The *First-In First-Out* algorithm, is a non-preemptive algorithm which executes the processes in order of arrival. It is very simple to implement but can lead to large waiting times

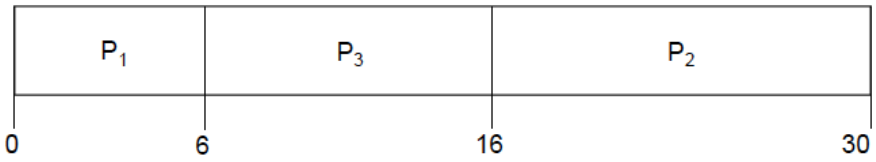


- Execution times:
  - P1: 0 - 6
  - P2: 6 - 20
  - P3: 20 - 30
- Waiting times
  (= execution - arrival):
  - P1: 0 - 0 = 0
  - P2: 6 - 4 = 2
  - P3: 20 - 5 = 15
- Average waiting time:
  - (0 + 2 + 15)/3 = 5.66…

- Turnaround times
  (=completion - arrival):
  - P1: 6 - 0 = 6
  - P2: 20 - 4 = 16
  - P3 : 30 - 5 = 25

6.5 *SJF*

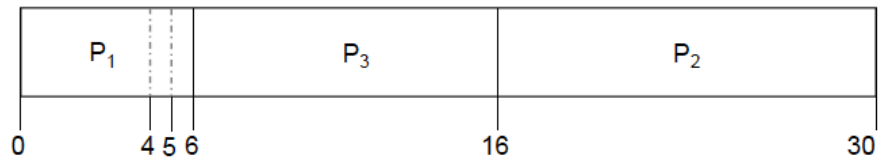
The *Shortest Job First* algorithm chooses the processes with the lowest CPU burst time to be executed first, if the processes have the same CPU burst time then it falls back to FIFO. This is also simple to implement, but an obvious problem is that processes with large CPU burst times may take much more to implement



- Execution times:
  - P1: 0 - 6
  - P2: 16 - 30
  - P3: 6 - 16
- Waiting times (= execution - arrival):
  - P1: 0 - 0 = 0
  - P2: 16 - 4 = 12
  - P3: 6 - 5 = 1
- Average waiting time:
  - (0 + 12 + 1)/3 = 4.33…
- Turnaround times (=completion - arrival):
  - P1: 6 - 0 = 6
  - P2: 30 - 4 = 26
  - P3 : 16 - 5 = 12

6.6 *SRTF*

The *Shortest Remaining Time First* is a preemptive version of SJF, i.e. it looks at the list of processes in the ready queue with each new arrival and compares the total remaining completion times, if the total time of a waiting process is lower than the one currently in execution then it will interrupt execution and swap them.

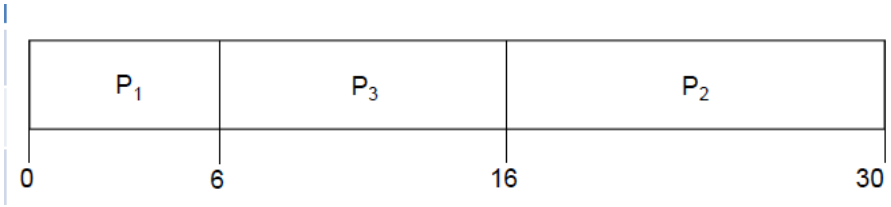


- Execution times:
  - P1: 0 - 6
  - P2: 16 - 30
  - P3: 6 - 16
- Waiting times (= execution - arrival):
  - P1: 0 - 0 = 0
  - P2: 16 - 4 = 12
  - P3: 6 - 5 = 1
- Average waiting time:
  - (0 + 12 + 1)/3 = 4.33…

- Turnaround times (=completion - arrival):
  - P1: 6 - 0 = 6
  - P2: 30 - 4 = 26
  - P3 : 16 - 5 = 12

6.7 *NPP*

The *Non-Preemptive Priority Scheduling* adds processes to a queue according to their priority values; if processes share the same priority level, then it reverts back to FIFO. It is also fairly simple to implement, but similarly to SJF it might lead to *starvation*, i.e processes with low priority may take ages or never run

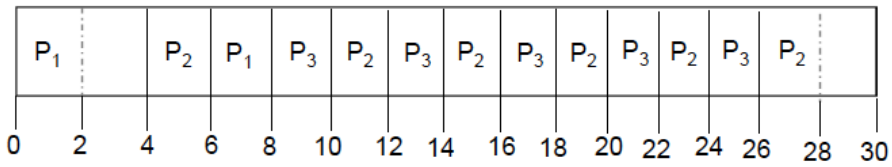


- Execution times:
  - P1: 0 - 6
  - P2: 16 - 30
  - P3: 6 - 16
- Waiting times (= execution - arrival):
  - P1: 0 - 0 = 0
  - P2: 16 - 4 = 12
  - P3: 6 - 5 = 1
- Average waiting time:
  - (0 + 12 + 1)/3 = 4.33…

- Turnaround times (=completion - arrival):
  - P1: 6 - 0 = 6
  - P2: 30 - 4 = 26
  - P3 : 16 - 5 = 12

6.8 *RR*

The *Round-Robin Scheduling* is the preemptive version of FIFO, where processes are executed according to time of arrival but only for a certain time period - *quantum*. If the process finished early then the dispatcher moves down the queue, if the opposite happens the process is added to the back of the queue.



- Execution times:
  - P1: 0 - 4, 6 - 8
  - P2: 4-6, 10-12, 14-16, 18-20, 22-24, 26-30
  - P3: 8-10, 12-14, 16-18, 20-22, 24-26
- Waiting times (= execution - arrival):
  - P1: (0-0) + (6-4) = 2
  - P2: (4-4) + (10-6) + (14-12) + 18-16) + (22-20) + (26-24) = 12
  - P3: (8-5) + (12 - 10) + (16-14) + (20-18) + (24-22) = 11
- Average waiting time:
  - (2 + 12 + 11)/3 = 11.66…
- Turnaround times (=completion - arrival):
  - P1: 8 - 0 = 8
  - P2: 30 - 4 = 26
  - P3 : 26 - 5 = 21

6.9 *MQ*

The *Multilevel Queue Scheduling* separates processes according to their priority values into separate queues. This has the advantage of reducing the time taken to search through all processes since it needs only to look at the highest-priority queue. It is then often combined with RR to decide which process within a queue should be executed next

To do...

- ☐ 1 (p. **??**): Insert image from anki card
- ☐ 2 (p. **??**): Insert image from anki card