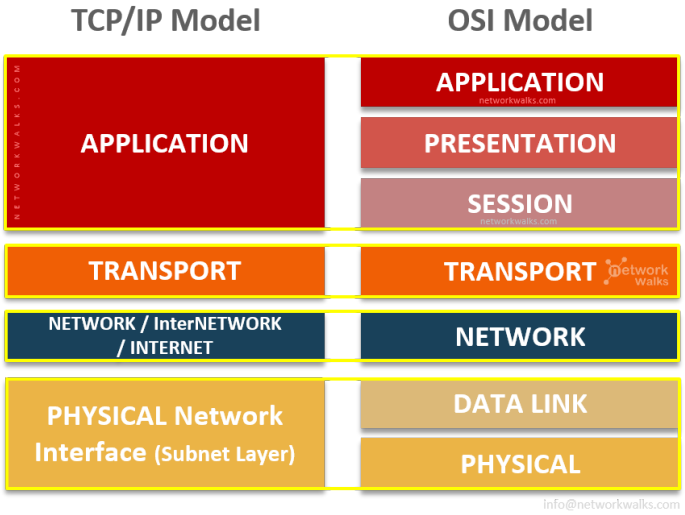
CS2005 Networks and Operating Systems

Lecture 1 - Introduction

The internet is a “network of networks” that are connected via transmission mediums in order to provide services for the user, e.g., E-Mail, File Sharing, etc.



Operating Systems (OSs) - Systems that provide an interface for the user so that the hardware may interact with the software.

Quick “Sum Up” of IP Stack: ATNLP

- **Application** - Supporting network applications (**FTP, SMTP, HTTP**)
- **Transport** - Ensures data packets are sent accurately and reliably across network (**TCP, UDP**)
- **Network** - Handles routing of data from source to destination (**IP, Routing Protocols**)
- **Link** - Data transfer between connected network elements (**Ethernet, WiFi, PPP**)
- **Physical** - Physical transfer between connected network elements (**Wired**)

Quick “Sum Up” of ISO/OSI Stack:

- **Presentation** - Allows applications to interpret meaning of data, e.g., encryption, compression
- **Session** - Synchronisation, checkpointing, recovery of data exchange. Handles opening/closing of “sessions”

Lecture 2 - Application Layer 1

The internet consists of billions of connected devices (**hosts**) that run network apps in order to communicate amongst each other. Communication is made possible via communication links, whether physical or wireless (f**iber, copper, radio**). The rate of transmission is known as **bandwidth**.

Packet Switching - Forwarding packets (data split into small parts) across a network independently.

Internet - Essentially a “network” composed of “networks”, or interconnected (Internet Service Providers) ISPs. Can also be described as an infrastructure that provides services to applications.

Internet Standards - Guidelines dictating the communication/sharing of data, therefore enabling the smooth functioning of the internet.

Protocols - Defines format, order of messages sent/received among network entities and actions taken on message transmissions:

- **Human Protocols** - Basic conversations, essentially. Specific messages sent. You can send/say anything like a human.
- **Network Protocols** - Machine conversations that are dictated by protocols.

Lecture 3 - Application Layer 2

HyperText Transfer Protocol (HTTP) - The web's **application layer** protocol. It is a **client/server** protocol:

- **Client** - (Usually a web browser), requests, receives and displays web objects.
- **Server** - Sends objects in response to the client's requests - HTTP is stateless (each response is treated new - NO history)

HTTP is an **application layer** protocol, but it uses **TCP** (a **transport layer protocol**) in order to ensure packets are delivered:

1) The **HTTP** client initiates a **TCP** connection with the server.
2) Once a connection is made, the browser and the server processes access **TCP** through their socket interfaces
3) The **client** sends **HTTP** request messages and receives response **HTTP** messages
4) The **server** sends request messages and receives response messages

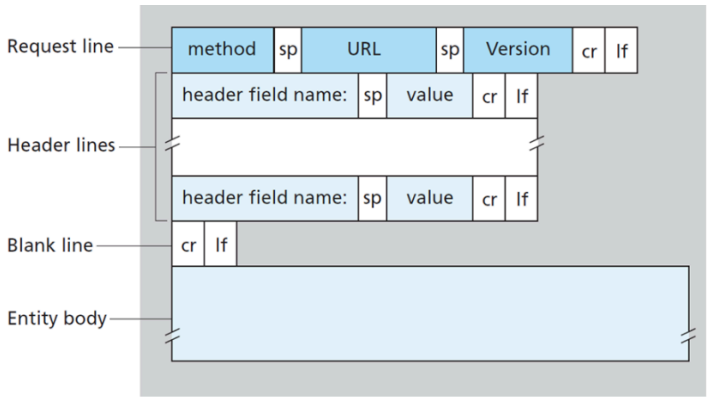
Two types of HTTP connections exist:

- **Non-Persistent** - At most, only ONE object is sent over a TCP connection and then the connection is closed. You will need to repeat a connection over and over if there's multiple objects at play.
- **Persistent** - The server leaves the connection open after receiving a response message, thus allowing for multiple objects to be sent over in a single. This is useful when loading web-pages with lots of assets.

Round-trip Time (RTT) - The time taken in order for a small packet from client to server, and then back to the client again. For example, the TCP connection/File Request and Response each take 1 RTT. Includes:

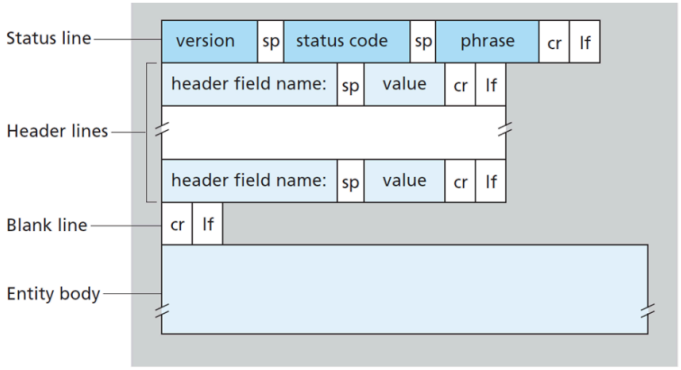
- **Delays**
- **Packet Propagation Delays**
- **Packet Queueing Delays**
- **Packet Processing Delays**

HTTP Request Message



GET /index.html HTTP/1.1\r\n - **Request Line (GET, POST, HEAD)**
Accept: text/html,application/json\r\n - **Header Lines**
Accept-Language: en-us,en;q=0.5\r\n
Connection: keep-alive\r\n
……
\r\n - **Carriage Return**

HTTP Response Message



HTTP/1.1 200 OK\r\n - Status Line - **(Protocol, Status Code/Phrase)**
Date: Sun, 26 Sep 2010 20:09:20 GMT\r\n
Content-Length: 2652\r\n - **Header Lines**
Content-Type: text/html\r\n
……
\r\n
data data data … - **Data, e.g., Request HTML file**

Cookies - Small pieces of data that keep session state information in order to identify/restrict users. Has four components:

- **Header line in request message.**
- **Header line in response message**
- **A file saved on the user's PC and managed by the web browser.**
- **A back-end database on the website.**

Web Caches - A stored copy of a website that's used to improve website loading times and reduce server load by re-using the same data. Stale data is avoided via conditional GET - I**f-Modified-Since** and **Last-Modified.**

E-Mails - Computer based applications that are used for exchanging messages across networks. Has three MAJOR components:

- **User Agents** - Compose, edit, read and save mail messages. (Outgoing, incoming emails stored on server).
- **Mail Servers** - Mailbox (incoming messages) and message queues (outgoing messages) and an SMTP implementation.
- **Simple Mail Transfer Protocol** - A protocol between mail servers to send emails. It uses **TCP** and **port 25.** Involves **handshaking (greeting)**, **transfer of messages** and **closure**. It's a **push protocol**, meaning you cannot obtain emails using SMTP. That requires **pull protocols**.

Mail Message Format - Header lines, then BLANK then the body (bulk of message)

Internet Message Access Protocol (IMAP) - An email protocol that allows for more advanced features, such as the management of stored messages on a server.

Post Office Protocol 3 (POP3) - An email protocol that deletes emails from a server once they have been saved onto a local device. Like taking your mail out from the box.

Lecture 4 - Distributed Systems

Distributed System - A collection of loosely coupled nodes interconnected by a communication network - A collection of computers that work together to complete a unified goal:

- **Site** - Refers to the location of a machine/node in a DS layout
- **Node** - Varies in size and function / e.g., processors, machines, hosts or even sites - **Generally, one node at one site, the server, has a resource that another node at another site, the client (or user), would like to use**

Why do we need Distributed Systems:

- **Resource Sharing** - Allows a user at one site to use the resources available at another.
- **Computation Speedup** - Sub Computations can be distributed across various sites and run at the same time - If one site is overloaded, move a job to another.
- **Reliability -** If one site fails, the remaining sites can continue to operate, thus making the system more reliable. With enough redundancy, the system can continue operation even if some sites are down.

- **Communication -** Users at the various sites are able to exchange information. At lower levels, messages are passed. At higher levels, files, logins, mails and RPCS can be used.

Advantages and Disadvantages of Distributed Systems

Advantages:

- Reliable, high fault tolerance
- Scalability
- Flexibility
- Speedup
- Openness
- High Performance

Disadvantages:

- Difficult troubleshoot
- Less software support
- High network infrastructure costs
- Security Issues

Process Communicating - A program while executing is a "**process**". Processes communicate within the same host through the use of inter-process communication defined by the OS. The initiator process is known as the "**client**", whereas the process that waits to be contacted to is the "**server**"

Sockets - A client and server process communicate with each other by reading and writing to **sockets.** Think of them as doors - push a message out and rely on the transport infrastructure on the other side to deliver the message. NEEDS a destination address to send the data to:

- Includes destination host's IP address so that the routers can send the packet(s) through the internet and to the destination host.
- When a socket is created a **port number** (identifier) is assigned to it, which is included in the destination address.
- Also attaches the sender's **host IP** and **port number**.
- Typically, the source address is attached automatically by the OS.

User Datagram Protocol (UDP) - A **connectionless** and **unreliable** communication protocol used in computer networks. Primarily used for **SPEED** and **EFFICIENCY.** Occasional packet loss is tolerable. **No connection (handshakes) is established before transmitting data**. SOCK_DGRAM

Transmission Control Protocol (TCP) - A **reliable**, **connection-oriented** protocol that ensures data is sent in the correct order and without errors - works with IP (Internet Protocol). ByteStream oriented. **TCP SERVER HAS TWO SOCKETS,** SOCK STREAM

Remote Procedure Calls (RPCs) - Allows the program (client) to request a service from another program (the server) without needing to know the specific network details - hides the details.

Pipes - A channel that allows for two processes to communicate with each other:

- Ordinary - Standard, one-way, parent-child pipes. The consumer reads from the read end and writes to the write-end
- Named - Sophisticated, bi-directional pipes that several processes can use

Lecture 5 - Transport Layer 1

Transport Layer - Provides logical communication between application processes running on different hosts. - Implemented in the end systems, but NOT in network routers:

- **Sending** - Converts the application layer's messages into transport layer packets (segments) by (possibly) breaking the messages down into chunks and adding a transport layer header. The segment is passed to the network layer, in which it is encapsulated into a network layer packet
- **Receiving** - The network layer extracts the transport layer segment and then passes it up to the transport layer. It then processes that segment and makes the data available to the receiving application.

Difference between Transport Layer and Network Layer: TRANSPORT IS COMMUNICATION BETWEEN **PROCESSES**, NETWORK IS COMMUNICATION BETWEEN **HOSTS**. Transport layer relies on the network layer and enhances its services.

Process-Level Addressing - Used to differentiate between processes (applications). The network layer addresses the identification of hosts

Multiplexing - The gathering of data chunks from different sockets, encapsulation of each data chunk with a header AND THEN passing to the network layer (**Sending host**)

Demultiplexing - The job of delivering the received segments to the correct socket (**Destination host**)

Connectionless Transport - Aside from Multi/Demultiplexing and some **light-error checking**, UDP does NOT add anything to IP. It simply takes messages from the application processes, attaches SOURCE and DESTINATION PORT NUMBERS (for mux/demux) and sends it off to the network layer.

UDP Structure:

- **Source Port#** - Port number of the sending application
- **Dest Port# -** 16-bit number that identifies the receiving app/service at the destination PC
- **Length** - Specifies the number of bytes in the UDP segment (including header). Explicit length value is NEEDED
- **Checksum -** Checks if errors have been introduced into the segment - data integrity

TCP Structure:

- **Source Port#** - Port number of the sending application
- **Dest Port# -** 16-bit number that identifies the receiving app/service at the destination PC
- **Sequence Number -** 32-bit values used to count each byte of data transmitted
- **Acknowledgement Number** - The sequence number of the next byte the receiver expects to get
-

Lecture 6 - Transport Layer II

TCP Light Facts:

- It's a "full-duplex service", meaning data can be transmitted both ways
- Always point to point
- A handshake NEEDS to happen before any data is sent

- Chunks of data are combined with TCP headers, forming TCP **Segments**

TCP Reliable Data Transfer - Ensures that the data steam has no gaps, isn't duplicated and in sequence. If a timeout is triggered, the ACK may still arrive, causing confusion

TCP Flow Control - A speed matching service that can prevent data spam. It matches the rate at which the sender is sending against the receiving applications reading rate.

TCP Connection Establishment:

- **Client asks to connect (SYN):** The client sends a special message (a TCP segment with the SYN flag turned on) to the server. This message includes a starting sequence number chosen by the client. Think of it as: "Hi Server, I want to connect! My first message number is X."
- **Server acknowledges and replies (SYN-ACK):** The server receives the request, sets aside resources for the connection, and sends back its own message (a segment with both SYN and ACK flags on).

Why would we use UDP?

UDP	TCP
• Application-level control over what data is sent, and when. Under UDP, as soon as an application process passes data to UDP, UDP will package the data inside a UDP segment and immediately pass the segment to the network layer.	• TCP has a congestion control mechanism that throttles the transport layer TCP sender when one or more links between the source and destination hosts become excessively congested.
• Real-time applications often require a minimum sending rate, do not want to overly delay segment transmission, and can tolerate some data loss.	• TCP will continue to resend a segment until the receipt of the segment has been acknowledged by the destination, regardless of how long reliable delivery takes.
• No connection establishment. UDP just blasts away without any formal preliminaries. Thus UDP does not introduce any delay to establish a connection.	• TCP uses a three-way handshake before it starts to transfer data. For example, TCP connection-establishment delay in HTTP is an important contributor to the delays associated with downloading Web documents.
• No connection state. UDP does not maintain connection state and does not track any parameter. For this reason, a server devoted to a particular application can typically support many more active clients when the application runs over UDP rather than TCP.	• TCP maintains connection state in the end systems. This connection state includes receive and send buffers, congestion-control parameters, and sequence and acknowledgment number parameters, information needed to implement TCP's reliable data transfer service and to provide congestion control.
• Small packet header overhead. The UDP segment has only 8 bytes of overhead.	• The TCP segment has 20 bytes of header overhead in every segment.

Lecture 6 - Network Layer

Now, I want you to remember:

Difference between Transport Layer and Network Layer: TRANSPORT IS COMMUNICATION BETWEEN **PROCESSES**, NETWORK IS COMMUNICATION BETWEEN **HOSTS**. Transport layer relies on the network layer and enhances its services.

Okay, good?

Network Layer - The layer/protocol that handles communication between **HOSTS (*A host is a computer or other device that communicates with other hosts on a network.*).** It can be split into two interacting parts/planes:

- **Data (Forwarding) Plane**: Determines how a datagram (**unit of data transmitted over a network**) is **forwarded** from one system to another, e.g., **WiFi networks, Ethernet. IT'S THE ONE THAT MOVES HANDLES THE FORWARDING OF THE DATAGRAMS**
- **Control (Routing) Plane**: Determines how the data should be **routed** and **processed** across a network through the use of **routing protocols**. **IT'S THE ONE THAT MANAGES HOW THE DATAGRAMS ARE ROUTED**

Software Defined Networking (**SDN**) - The **separation** of the data and control plane by making the control plane functions a separate service - e.g., in a remote "controller"

Network Layer Protocols run in every host/router!:

- **Sending** - The Network layer takes segments from the transport layer, encapsulates them (**ADDS IP HEADER**)
- **Receiving** - Receives the datagrams from the nearby router, extracts the **Transport Layer** segments and then delivers them to the **Transport Layer**.

Routing - The **NETWORK-WIDE** process of determining the best path for that packet to get to its destination! It takes a **while,** so it's often implemented in **software.**

Forwarding - The **ROUTER-LOCAL** action of transferring a packet from a **router's interface** to **another interface,** based on the packet's **IP Address.** It's **quick**, so it's typically implemented in **hardware.**

Forwarding Table - A data structure used by a router to determine **how to forward incoming packets** based on their **destination IP addresses**. There are two main **approaches** forwarding tables use:

- **Traditional** - The router builds its own routing table and makes its own forwarding table from it. The **control** and **data** plane (see above) are combined into it.
- **Software Defined Networking -** A **centralized (see below)** SDN controller computes the network paths and **directly programs** forwarding tables on all switches and routers. As with SDN, the control and data planes are **separated**

Router Architecture - A router consists of **four** router components:

- **Input** - Handles the **PHYSICAL LAYER**'s functions (**Receiving packets from incoming links**, **error checking, removing ethernet headers, etc.**)
- **Switching Fabric** - Connects the router's input ports to its output ports - completely contained in router
- **Output Ports** - Stores packets received from the switching fabric and sends them to the outgoing link, via the Link/Physical-Link Layer
- **Routing Processor** - Performs control plane functions

Routing Algorithm - Determines the best possible route for senders/receivers to go through via the network in the router. They can be defined as:

- **Centralised** - Every router has the complete topology of a network
- **Decentralised** - The router knows only the physical connected neighbouring routers
- **Static** - Routes change very slowly over time
- **Dynamic** - Routing paths change, depending on network traffic/topology change
- **Load Sensitive** - Link costs varies, depending on the level of congestion.
- **Load Insensitive** - Link cost does not explicitly reflect the current level of congestion

Lecture 7 - Network Layer II

A reminder that the Network Layer:

- Provides logical communication between **HOSTS** (Not processes, that's the **Transport Layer)**
- The **Data Plane** controls the actual **forwarding** of a datagram to a router's output link

- The **Control Plane** controls the logic that dictates how that datagram is **routed** across a network, through the use of **routing algorithms.**
- **Sending** - The Network layer takes segments from the transport layer, encapsulates them (**ADDS IP HEADER**)
- **Receiving** - Receives the datagrams from the nearby router, extracts the **Transport Layer** segments and then delivers them to the **Transport Layer**.

IPv4 Datagram Format - Widely deployed version of the Internet Protocol. It's still widely used - Very useful for the identification and location addressing for devices on a network - common in NAT-Based networks.

IPv6 Datagram Format - A successor to IPv4 that was developed to solve 4's limitations. More next-gen, as it supports modern devices and has **unlimited** address space.

Alright, let's go over the structure of **IPv4/IPv6!:**

- **Version Number -** A **4 bit field** that indicates the version of the IP used - **If 4, then IPv4. If 6, then IPv6**
- **Header Length** - Indicates the length of the IP header, **(min 20 bytes, max 60)**
- **Type of Service** - An **8-bit** number that tells the user the type of **IP datagrams** and **request services**
- **Datagram Length** - The total length of the IP Datagram, (**including the header**)
- **Identifier, Flags and Fragmentation Offset** - These keep information needed for IP fragmentation
- **Time to Live** - Ensures datagrams **don't circulate forever**. Each datagram is given a field - **or time limit. If it reaches 0, then the datagram is dropped.**

- **Upper Layer Protocol** - Indicates the specific transport-layer protocol to which the data portion of this IP datagram should be passed. **For example, TCP is 6, UDP is 1.**
- **Header Checksum -** Aids a router in **detecting bit errors** in a received datagram.
- **Source and Destination IP Addresses** - Tells the receiver where the packet came from/where it's headed.
- **Options** - Allows an IP Header to be **extended**. Rarely used, according to the lecture slides.
- **Data (Payload) -** Mostly carries the **Transport Layer segment (TCP or UDP) to be delivered to the destination**, but it can also carry other forms of data like ICMP messages.

Below is a table with comparisons to both IP versions:

Field	IPv4	IPv6
Version number	4-bit field, value = 4	4-bit field, value = 6
Header length	Variable (min 20 bytes, max 60) – specified in header	Not present – IPv6 has a fixed header length of 40 bytes
Type of service (ToS)	8 bits; used for priority and QoS	Replaced by **Traffic Class** (8 bits)
Datagram length	16 bits; total length = header + data	**Payload Length** field (excludes header, which is fixed)
Identifier, flags, frag. offset	Used for fragmentation of large packets	Removed from base header; handled by a **Fragment extension header**

Time-to-live (TTL)	Decremented each hop; when 0, packet discarded	Renamed to **Hop Limit**, same behavior
Source IP Address	32-bit field	128-bit field
Destination IP Address	32-bit field	128-bit field

Fragmentation - The process of breaking down an IP packet into smaller pieces in order to account for a network link's **Maximum Transmission Units** (**MTU):**

- The IP splits the data (payload) into smaller datagrams (**Fragments**)
- Those smaller datagrams are then **encapsulated** into **seperate link layer frames** and sent over the outgoing link.
- When a datagram is created, an **identification number** is created.
- When a router needs to fragment a datagram, each resulting datagram (**fragment**) is stamped with the **source address**, **destination addres**s, and **identification number** of the original datagram.
- These identification numbers **help determine which of the datagrams are fragments of the same larger datagram.**

Reassembly - As established, fragments from the same sender and **identification number** belong to the same **datagram**:

- In case any of the fragments get lost, the last fragment always has the **flag bit set to 0.**
- The **offset field** (see above) is used to specify where the fragment fits within the original IP datagram.

IPv4 Addressing - The process of assigning 32-bit addresses to devices, in order to help said devices identify themselves and communicate over a network:

- Written in **dotted-decimal notation**, (**193.32.216.9 ← This format**)
- Note that its in decimal format, meaning each dot separates the **32 bit address into 4 part**s, (**8 bits separated by the dots**)
- They **cannot** be changed.
- Some IP addresses have a slash after them, (e.g. 223.1.1.0/**24),** which indicates a **subnet** (see below)

Classless Inter-Domain Routing (**CIDR**) - A method for allocating IP addresses:

- The **prefix length** indicates how many bits of the IP address are used for the **network portion**.
- The remaining bits are used for the **host portion**.
- **192.168.1.0/24** → The first 24 bits (**192.168.1**) are the **network**, and the **last 8 bits are for hosts**.
- **10.0.0.0/8** → Very large network, **8 bits for network, 24 bits for host**

Subnets - A smaller network, split from a larger one:

- Acts as a **separate network**, albeit **still part of the same overall IP address space**

Subnet Mask	CIDR Notation	Usable Hosts
255.255.255.0	/24	254

255.255.255.128	/25	126
255.255.255.192	/26	62

Lecture 8 - Operating Systems Structures/Processes

Finally… Networks is over. Now onto more hell.

Operating System - Provides an environment to the user wherein they can **execute (run)** programs. Below are the services an OS offers:

- **User Interface (UI)** - The space where the user interacts with the machine. There are different types - **Command Line, Graphical, etc.**
- **Program Execution** - The system must be able to load a program into memory AND run it - **must also be able to end it!**
- **I/O Operations** - Operations involving reading or writing data from and to external sources - essentially input and output
- **File-System Manipulation** - The operating system must allow for programs to **read and write files to directories, search for files and list information about them**. Also permission management, e.g., deny access to files.
- **Communications** - The act of one process exchanging information with another process - May be implemented via **shared memory** (two processes read/write to a shared section of memory), or **memory passing** (packets of information are moved between processes by the OS.
- **Resource Allocation** - When multiple jobs/users are active, resources must be allocated to each of them

- **Accounting** - Keeping track of which users use how much and what kinds of computer resources
- **Protection -** Ensures that all access to system resources is controlled.
- **Security** - Securing the system from outsiders - may require outside users to authenticate themselves.

CLI/Command Interpreter - Allows for direct command entry - sometimes implemented in kernel, sometimes by system's program. Primarily fetches a command from the user and executes it. (**e.g., cmd.exe, C shell, BASH**)

Graphical User Interface (GUI) - User interface with a user-friendly interface - icons represent files, programs, actions, etc. Note that many OSs now include both CLI and GUI interfaces, such as Windows, Linux and Mac.

Program Execution - The system must be able to load a program into memory and to run that program, end execution, either normally or abnormally (indicating error) - typically written in a high level language - **Mostly accessed by programs with a high-level API (Application User Interface), instead of direct system call use**

System Call Interface - The OS normally associates each system call with a number. These numbers are maintained into a table by the system call interface:

- **Process Control -** create/terminate process, end abort, load, execute, etc.
- **File Management** - create file, delete file, open, close file, etc.
- **Device Management** - request/release device, read, write, etc.
- **Information Maintenance** - get/set time or date, get/set system data, etc.

- **Communications** - create/delete communication connection, send & receive messages

System Programs - Provides a convenient environment for program development and execution - basically the bridge from UI to system calls:

- **File Management -** Programs that allow for the manipulation of files and directories
- **Status Information** - Programs that ask the system for the date, time or amount of available memory, disk space, etc.
- **File Modification** - Text editors and file search, **e.g., Notepad**
- **Programming-Language Support -** Compilers, assemblers, debuggers and interpreters for common programming languages.
- **Programing Loading and Execution** - Once a program is assembled/combiled, it's loaded into the memory to be executed
- **Communication** - Programs that allow for creating virtual connections across processes and computer systems. **Allows for users to browse web pages, send emails, log into websites or even transfer files.**
- **Background Services** - Constantly running system programs, such as **schedulers, system monitors, printer servers, etc.**

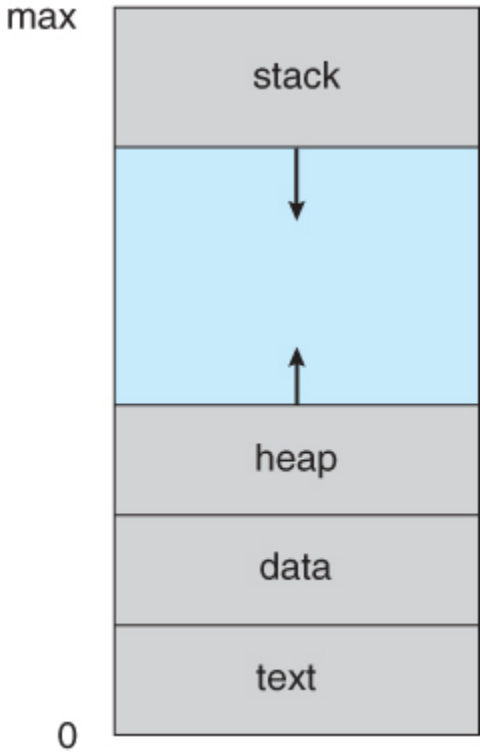
Process - A program ***while*** being executed. Requires certain resources to accomplish its task, **such as CPU time, memory, files, I/O devices**. These resources are allocated to the process either when it's created or while it's executing. Think of it as the unit of work in systems. The **OS** handles:

- The creation/deletion of user/system processes
- The scheduling of processes
- Deadlocking handling, synchronisation and communication

Deadlocking - When two or more processes are blocked indefinitely because each is waiting for the other to release a resource it needs.

Process Structure - A process is made up of:

- **Text** - Program code + current activity status, program counters, etc.
- **Stack** - Temporary data, **such as function parameters, local variables and return addresses**
- **Data** - Global variables
- **Heap** - Memory dynamically allocated during process run time.



Process State - As a process executes, it changes states:

- **New** - The process is being created
- **Running** - Instructions are being executed
- **Waiting** - The process is waiting for some event to occur
- **Ready** - The process is waiting to be assigned to a processor
- **Terminated** - The process has finished execution

It's important to note that **only one process can be running at any time**, whereas **many processes can be ready and waiting**.

Process Control Block (PCB) - Each process is represented in the OS by a **PCB:**

- **Process State** - The current state of the process, (see above)
- **Program Counter** - A counter indicating the address of the next instruction to be executed for this process.
- **CPU Registers** - High-speed memory contained in the CPU, that include, accumulators, stack pointers and general purpose registers
- **CPU-Scheduling Information** - Info that includes process process priority, scheduling parameters, etc.
- **Memory-management information** - Info that includes such items such as the value of base/limit registers, page tables, etc.
- **Accounting Information** - Info that includes the amount of CPU and real time used, time limits, process numbers, etc.
- **I/O status information** - Info that includes a list of the I/O devices allocated to the process

Process Scheduling - Responsible for choosing and running processes. Consists of queues - generally stored as linked lists. Also has a structure:

- **Job Queue** - Consists of all processes in the system
- **Ready Queue** - Processes that are residing in main memory and are ready and waiting to execute
- **Device Queue** - A program wishes to make an I/O request to a shared device - A list of processes waiting to use a device.

Lecture 9 - Threads

Sequential Execution - One task (job/process) at a time. Needs to complete execution for the previous task for another to start.

Concurrent Execution - Multiple tasks/subtasks run in parallel, thanks to CPU time-slicing. Runs part of a task, then goes into a **waiting state** to wait.

Parallel Execution - Multiple tasks/subtasks **actually run at the same time!** - Requires two or more CPUs/cores.

BOTH CONCURRENT AND PARALLEL EXECUTION NEED SYNCHRONISATION

An application…:

- Can be **neither parallel - nor concurrent**
- Can be **concurrent - but not parallel**
- Can be **parallel - but not concurrent**
- Can be **both parallel and concurrent**

Parallelism - Application processes multiple sub-tasks of a task in a multi-core CPU at the same time.

Thread - A basic unit of CPU utilization - A single sequence stream within a process - shares with other threads belonging to the same process its **code and data section, and other OS resources**. Consists of:

- **Thread ID -** Uniquely identifies each thread created during the lifetime of a process.
- **Program Counter** - A counter indicating the address of the next instruction to be executed for this process.
- **Register Set -** Includes temporary memory locations used to store data and processing results from the thread.

- **Stack -** Temporary data, **such as function parameters, local variables and return addresses**

PROCESS MEANS A PROGRAM IN EXECUTION

THREAD MEANS A SEGMENT OF A PROCESS

There are many benefits to **threads**:

- **Responsiveness** - Multithreading allows for programs to continue running, even if part of it is blocked or running a lengthy operation
- **Resource Sharing** - Threads share the memory and resources of the process and which they belong to by default - allowing for an application to have several different threads of activity within the same address space
- **Economy** - Threads shares the resource of the process to which they belong, thus its more economical to create and context-switch threads.
- **Scalability** - Threads may be running in parallel on different processing cores, thanks to multithreading. A single-threading processes can only run on one processor, however.

Multithreading - Allows for the simultaneous execution of two or more parts of a program, aiming at maximising CPU time utilisation. Contains two or more parts that can run concurrently - each part of that program is called a **thread**.

Java Thread States - Allows for a program to operate more efficiently by doing multiple things at the same time. The states of which are:

- **NEW** - A thread that has not yet started
- **RUNNABLE -** A thread executing in the Java virtual machine
- **BLOCKED** - A thread that is blocked, waiting for a monitor lock

- **WAITING -** A thread that is definitely waiting for another thread to finish
- **TIMED_WAITING -** A thread that is waiting, but for a specified time.
- **TERMINATED -** A thread that has exited.

Lecture 10 - Synchronisation

A common problem is that co-operating processes can share data. Concurrent access to that shared data may result in **data inconsistency**, which is the process of identifying discrepancies in data.

Producer-Consumer Problem (Bounded-Buffer Problem) - A concurrency issue where multiple producers and consumers share a common buffer. Producers **add** items to the buffer, whereas consumers **remove** items from the buffer:

- The problem lies in the fact that both **producer** and **consumer** can update the counter variable in an uncontrolled manner.
- Can cause a **race condition** - When several processes manipulate the same data concurrently and the outcome depends on the order in which the access took place.
- To fix all of this, we need **synchronisation!**

Critical Section Problem -

- A **critical section** is a segment of code within a process or thread where it accesses shared resources.
- When multiple processes/threads execute **concurrently**, more than one might try to enter its critical section ***at the same time***. If this happens, and they modify the shared resource simultaneously, it can lead to **unpredictable outcomes**

- The primary goal is to design a solution (e.g., using locks, semaphores, monitors) that enforces rules for **entering** and **exiting** critical sections.

Critical Section Processes:

- **Entry Section -** Must request permission to enter the critical section
- **Critical Section -** The protected segment of code
- **Exit Section** - Notify everyone that exiting the critical section
- **Remainder Code** - The rest of process's code

Critical Section Processes:

- **Mutual Exclusion -** If a process is executing in its critical section, then no other processes can be executing in theirs.
- **Progress -** If no processes are executing in their critical section, and there exists some processes that want to then… **Only those processes that are not executing in their remainder sections can participate in deciding which will enter its critical section next**
- **Bounded Waiting** - A bound must exist on the number of times that other processes are allowed to enter their critical sections after a process has made a request to enter its critical section and before that request is granted

Peterson's Solution - An algorithmic description of solving the critical section problem - Restricted to two processes that alternate between their critical and remainder sections:

- **Int turn;** - Indicates whose turn it is to enter its critical section
- **Boolean flag[2]; -** Indicates if a process is ready to enter its critical section

Mutual Exclusion (Mutex) Locks - A software tool to help solve the critical section problem:

- Protect a critical section by **acquire()** a lock
- And then you **release()** the lock
- Must be **atomic** - ALL of the operation must happen at once, or not at all
- Requires **busy waiting -** process waits and keeps checking for the condition to be satisfied before going ahead with execution
- **Spinlock** - A lock that causes a thread trying to acquire it to wait in a loop, while repeatedly checking if that lock is available

Semaphores - More sophisticated form of synchronization than Mutex locks. It's an **integer variable,** that aside from initialisation **(starting)**, is only accessed through two standard atomic operations:

- **wait()** - While **S** is less than or equal to **0**, then be busy and wait.
- **signal()** - While signalling, add S++ to the counter.
- **Counting Semaphore -** Integer value can be anything over 1
- **Binary Semaphore -** Integer value can only be between 0 and 1

Starvation - Indefinite blocking - A process may never be removed from the semaphore queue in which it is suspended

Priority Inversion- Scheduling problem when lower-priority processes holds a lock needed by high-priority processes

Monitors - A high-level abstraction that provides a convenient/effective mechanism for process synchronisation:

- Only **one process** may be active within the monitor at a time
- Internal variables **only accessible** by code within the procedure
- Not sufficiently powerful to model synchronisation, but can still add detail

Lecture 10 - Security

The Security Problem - A system is secure if its resources are used and accessed as intended under all circumstances - though **total security cannot be achieved**:

- **Attack** - An attempt to breach security
- **Threat -** A **potential** security violation - may or may not happen but can cause serious damage
- **Intruder** - A person who attempts to gain unauthorised access to a system. They will either **violate security** or **damage a system/disturb the data**

Types of Security Violation:

- **Breach of Confidentiality -** Unauthorised **access/theft** of information/data, such as credit card info, identity, etc.
- **Breach of Integrity** - Unauthorised modification of data - e.g., altering the text of a message/website
- **Breach of Availability -** Unauthorised destruction of data, e.g. website defacement, deleting files
- **Theft of Service** - Unauthorised use of services - using a service without paying for it
- **Denial of Service** - Prevention of legitimate use of a system through the use of illegitimate traffic spam.

Security Violation Methods:

- **Masquerading -** Pretending to be someone else e.g., another host/person to gain access - **a.k.a Breach of Authentication**
- **Replay Attack -** Malicious or fraudulent repeat of a valid data transmission.
- **Man in the Middle Attack -** An attacker sits in the data flow of a communication, acting as the sender to the receiver, and vice versa

- **Session Hijacking** - Intercepting an active communication to bypass authentication

Security Measure Levels - It's impossible to have absolute security, so settle for having as much as possible at the very least. Security must occur at four levels to be effective:

- **Physical** - Data centers, servers, connected terminals
- **Human** - Phishing, dumpster diving, etc.
- **Operating System** - Protection Mechanisms, debugging
- **Network -** Intercepted communications, DDOSing, interruptions

Program Threats:

- **Trojan Horse** - A malicious program that pretends to be a legitimate program - installs backdoors and other nasties.
- **Trap Door** - Leave a "hole" for accessing the system
- **Logic Bomb -** Activated under certain circumstances, which triggers a malicious function when conditions are met
- **Stack and Buffer Overflow -** Writes arguments into the return address on stack

Viruses - A fragment of code embedded in a legitimate program, that can spread over a network and self-replicate.:

- **File/Parasitic -** Attached to .exe files and spreads from there
- **Boot/Memory Viruses** - Infected the Master Boot Record (MBR), so that malicious code is loaded into memory when booting.
- **Macros -** Viruses written in macro languages that can infect documents. When a document is opened, the malicious code is run.
- **Source Code -** A virus that attempts to modify the source code of a program.

System and Network Threats - These threats create a situation in which OS resources and user files are misused:

- **Worms -** Self replicating standalone threats that spawn copies of itself and consume resources.
- **Port Scanning** - Automated attempt to connect to a port on a range of IP addresses.
- **Denial of Service -** Overloading a target computer to prevent it from being used.

Cryptography - The technique of secret writing, especially code and cipher systems, methods, and so on…:

- **Encryption -** Encoding messages/information in a way that unauthorised people cannot access it.
- **Decryption** - The process of decoding a message using a key.
- **Cipher/Cypher -** An algorithm used to perform encryption or decryption.
- **Plaintext -** The original, unencrypted message.
- **Ciphertext -** The coded, encrypted message.
- **Cryptosystem -** A set of algorithms for performing cryptography actions.
- **Cryptanalysis -** The study of how to crack encryption algorithms.

We **need** Cryptography because…:

- We need to protect against unauthorized people who know the source/destination of messages.
- No reliable way of determining the sender.
- No way of knowing if there's an eavesdropper.
- IP addresses can be spoofed, thus you can't (reliably) tell who has sent/received a request/response.

NO NETWORK CAN BE TRUSTED

Cryptography eliminates the need to trust the network for secure communication. It enables:

- A sender to encode a message, so that only a computer/user with a certain key can **decode the message**
- A recipient to verify that the message was created by some computer possessing a certain key

Encryption - Transforming data into an **Ciphertext** through the use of a cryptographic key, preventing unauthorized access and storage. An encryption algorithm consists of the following components:

- **K -** Keys
- **M** - Messages
- **C -** Ciphertexts
- An encrypting function **K→(M→C) - k∈K, Ek** generates ciphertexts from messages
- A decrypting function **K→(C→M) -** for each **k∈K, Dk** generates messages from ciphertext

An encryption algorithm must provide the following essentially property:

- Given a ciphertext **c∈C**, a computer can compute a plaintext **m** such that **Ek(m)=c** ONLY IF it possesses the key **k**

There exists two main types of encryption algorithms:

- **Symmetric -** The **same key** is used to encrypt/decrypt a message. Therefore, key **k** must be a shared secret between both parties. Such symmetric encryption algorithms that exist are: **Block Ciphers** (Triple DES, AES) and **Stream Ciphers** (RC4)
- **Asymmetric -** a.k.a **public-key encryption.** There are **different encryption** and **decryption** keys. The one who receives encrypted messages generates a pair of private - public keys. The public key is **made available and can be used by anyone**.

Only the **private key holder can decrypt these messages, though.**

RSA Algorithm - A user creates and publishes a public key based on **two large prime numbers** and an **auxiliary value**. Anyone can use the public key to encrypt a message, although the **prime numbers must be kept secret.** For an example of RSA, see:

- Let's make **p = 7**, and **q = 13**. These are our two prime numbers.
- We then calculate **N = 7 x 13 = 91** and **(p - 1)(q - 1) = 72**
- We next select **Ke**, **coprime** to 72, and **under 72, e.g.** 5.
- We then calculate **Kd**, such that **KeKd mod 72 = 1**, giving us a remainder of 29.
- The public key is then **Ke,N = 5. 91**
- The private key is then **Kd,N = 29, 91**
- Encrypting the message 91 with the public key will thus give you 62.