Hugh Baldwin
up2157117

Operating Systems and Internetworking

M30233
TB1

University of Portsmouth
BSc Computer Science
2nd Year

Contents

Part I

Operating Systems

Operating Systems Cheatsheet

Term	Definition
Kernel Mode	The CPU mode in which all assembly instructions can be used. This mode is usually used by the Operating System
User Mode	The CPU mode in which some assembly instructions cannot be used, such as IN and OUT. This mode is usually used by Application Software
System Software	The operating system and related utilities that control the computer, and provide essential functionality, such as keyboard and mouse input handling
Application Software	Programs that allow the user to perform specific funtions with a computer, such as word processing or internet browsing
Interrupts	A signal sent to the CPU, usually from an I/O device, that requests the attention of the CPU to handle an urgent operation

Lecture - Introduction

13:00 26/09/23 Tamer Elboghdadly

Operating Systems

- The Operating System sits inbetween the hardware and application software
- It usually is not the actual GUI, but provides functionality for the applications which implement it
- OSes typically provide abstractions for applications so that they can run on different hardware

User and Kernel Mode

- User mode
  - The programs that a user directly interacts with
  - Uses an API to access hardware, rather than having direct access
- Kernel mode
  - The programs that run the operating system
  - Has direct access to hardware

System and Application Software

- Application software
  - Programs that allow a user to perform a task
  - Requires the support of system software to run
- System software
  - Software directly related to the operating system
  - Manages the boot process
  - Hardware drivers
  - File system management

The main functions of an Operating System

- Resource management
  - Manages the memory, CPU and other hardware to allow multiple programs to run concurrently
  - Handles requests from applications to allocate more resources
- ”Extended machine”
  - Handles reading and writing to control registers, handlling interrupts, etc
  - Provides higher level APIs for other software to interact with the hardware

CPU Organisation

Registers

Name	Use	Description
EAX	Accumulator	The default register for many addition and multiplication instructions
EBX	Base	Stores the base address during memory addressing.
ECX	Count	The default counter for repeat (REP) prefix instructions and LOOP instructions.
EDX	Data	Used for multiply and divide operations
ESI	Source Index	Store source index
EDI	Destination Index	Store destination index
EBP	Base Pointer	Mainly helps in referencing the parameter variables passed to a subroutine.
ESP	Stack Pointer	Provides the offset value within the program stack.

Figure 2.1: General purpose registers in an x86 Intel CPU

- There are also special purpose registers, such as the Program Counter (PC) and the Program Status Word
- The Program Status Word sets the mode in which the CPU is operating

Assembly Language

- Assembly Language is the lowest-level programming language before Machine Code. It is slightly abstracted from machine code and uses neumonic symbols to represent instructions
- For example:
  - `MOV EBX, EAX`
  - This copies the value in EAX into the EBX register
  - `ADD EBX, 4`
  - This adds the value 4 to the EBX register
  - This is equivalent to $b = a + 4$
- The MOV instruction can also be used to move values between registers and the main memory
- I/O devices such as hard disks have a set of ports, which can be accessed to control the device and transfer data
- The special instructions IN and OUT are used to read from or write to ports
- For example:
  - `IN EAX, 368`
  - This copies the value from the port 368 into the EAX register
- The IN and OUT instructions can only be used when the CPU is running in Kernel Mode

User and Kernel Mode

- CPUs support running in two different modes: Kernel mode and User mode
- When running in User mode, some instructions (such as IN and OUT) cannot be used
- Typically, the Kernel of an operating system will run mostly or entirely in Kernel Mode

Interrupts

- When an external device needs to gain the attention of the CPU, it sends an interrupt
- A couple of examples are as follows
  - When a disk has requested data in it's buffer
  - When the key on an old PS/2 keyboard is pressed
- When the CPU recieves an interrupt, it must abandon whatever it is currently doing and run an Interrupt Handler routine
- Interrupt Handlers run in Kernel mode
- The CPU places whatever resources it was using onto the stack and executes the interrupt handler. Once the handler has finished, it returns to the execution path stored on the stack

Lecture - Concurrency

13:00 03/10/23 Tamer Elboghdadly

- A sequential system is one where multiple threads are exectued to completion, one after the other
- A concurrent system is one where multiple threads are exectued at the same time
- A parallel system is one where multiple threads are executed in parallel, usually on multiple hardware threads
- There are several issues that may arise from concurrency, especially in cases where threads of the same process are run on multiple processors in parallel
  - A thread is a sequence of instructions defined by a program or part there of
  - A process has one or more threads, each of which may be run in parallel
  - Threads within a single process can share resources with each other, but have their own thread of execution
- When multiple threads are run concurrently, the result is usually non-deterministic
  - This means that it is impossible to predict the order in which the instructions will run
  - This can lead to a program giving completely different results depending upon which order the instructions run
- Non-determinism can also lead to race conditions if multiple threads access the same variable
  - A race condition can lead to counting errors, data corruption or a complete deadlock of the program, depending upon which order the instructions execute
  - A specific type of race condition is interference, which is when multiple threads operate on the same variable, and interfere with each others results, such as two threads adding to a value, and the second thread overwriting the value of the first thread
- One method of avoiding non-determinism is to avoid threads sharing any variables at all. This works perfectly for some applications, but is very restrictive and can greatly reduce the efficiency of certain algorithms
- Any part of a program which requires using shared variables is known as the critical section, which must not be run at the same time as the critical section of any other thread

Lecture - Mutual Exclusion

13:00 10/10/23 Tamer Elboghdadly

- Techniques to ensure critical sections do not execute concurrently
- An atomic instruction or code fragment is a piece of code that executes without interruption, meaning that other threads cannot interfere

Naïve ME Solutions

Locks

- This works by using a single boolean variable to determine if the variables are locked by a thread or not
- When the variables are locked, in theory only one thread has access to them
- However, because the threads are running in parallel, there is a chance that multiple threads will read the lock variable as false simultaniously, resulting in two threads entering the critical section at once
- Therefore, this does not achieve Mutual Exclusion

Taking Turns

- This works by using a single variable to store which thread should get the next turn
- Each thread sits in a while loop until its their turn to enter the critical section. After finishing it's critical section, the thread assigns the variable to the next thread
- While this does guarantee Mutual Exclusion, it doesn't guarantee progress, as if any of the threads stalls at any point, all of the other threads will be left waiting for their turn

Interested Flags

- This works similarly to a lock flag, but rather than having only one lock, each thread is assigned an interested variable
- When a thread wants to enter the critical section, it sets it's own interested flag to true, and then waits until all other interested flags are false
- However, it is possible for both threads to set their interested flags at the same time, and therefore both enter a while loop
- Once again, this does provide Mutual Exclusion, but doesn't guarantee progress

Peterson's Algorithm

- This method combines interested flags with taking turns
- Each thread has an interested flag, and when it wants to enter the critical section, it sets it's interested flag, and then foregoes it's turn, assigning it to the other thread
- It then waits for the other thread to take it's turn, or in the case that neither thread is in the critical section, the other thread will reach the critical section and then forego it's turn
- This algorithm does work in theory and practice, but there's no way to generalise it for an arbitrary number of threads. It's very easy to implement for just 2 threads, but it gets exponentially harder as more threads are added

Practical Solutions

Hardware Test and Set

- This is an atomic instruction built into the instruction set of most modern processors
- An atomic instruction is one which performs a complex action, but acting like a single instruction
- That means that it's impossible for more than one thread to write using an atomic instruction at a time
- This basically allows us to use a lock variable, but without causing another race condition
- However, this is not really suitable in a multitasking environment, as it wastes a lot of CPU cycles continuously checking if the lock is available

Semaphores

- A semaphore is an integer variable that can be accessed by any thread, but only using two methods - P and V
- P decreases the value of the semaphore by one, but can never go below zero
- V increases the vaule of the semaphore by one
- If P failes to decrease the value, the thread calling it blocks until another thread increases it by one, allowing it to exit the block
- Since the semaphore starts at one, the first thread to reach the critical section decreases it by one and enters the critical section
- If another thread reaches the critical section, it has to block until the first thread has exited the critical section and increases the semaphore
- When using as a method for mutual exclusion, a binary semaphore is usually used (as the name suggests, this can only be 1 or 0)

Lecture - Synchronisation and Deadlock

13:00 17/10/23 Tamer Elboghdadly

Synchronisation

- Mutual Exclusion is just one type of synchronisation, but there are others, such as:
  - Messaging - Threads can send messages to each other. The recieving thread cannot continue until it gets a message from the sending thread
  - Join synchronisation - The parent thread can wait for the child thread to terminate
  - Barrier synchronisation - All threads must wait for all other threads to reach a barrier operation before they can continue
- Semaphores can be used to achieve multiple types of synchronisation
  - When using for Mutual Exclusion, the semaphore usually starts at 1
  - It could start at 0, and when a thread completes an operation, it raises the semaphore using V, allowing another thread to know it has completed an operation
  - This would allow you to guarantee the order in which the operations are run, rather than just that they don't run at the same time

Deadlock

- The resources of a computer can only be accessed by one thread at a time
- Mutual Exclusion is one example of this, as only one thread can access the areas in memory belonging to a data structre at a time
- Other examples are devices such as a printer or scanner
- A situation which could cause a deadlock is as follows
  - Two threads, A and B, need access to resources R and S
  - They both need access to the resources, but attempt to acquire them in different orders
  - Thread A acquires R and B acquires S
  - Thread A now attempts to acquire S, and B attempts to acquire R
  - Each thread is waiting for the other thread to release the resource
  - This causes a complete deadlock as neither thread will finish and release the resource

Deadlock Modelling

- Resource deadlock can be modelled using a resource allocation graph
- This shows
  - Which processes are requesting which resources
  - Which resources have been assigned to which processes
- If this graph contains a cycle, there will be a deadlock. If not, there won't be a deadlock

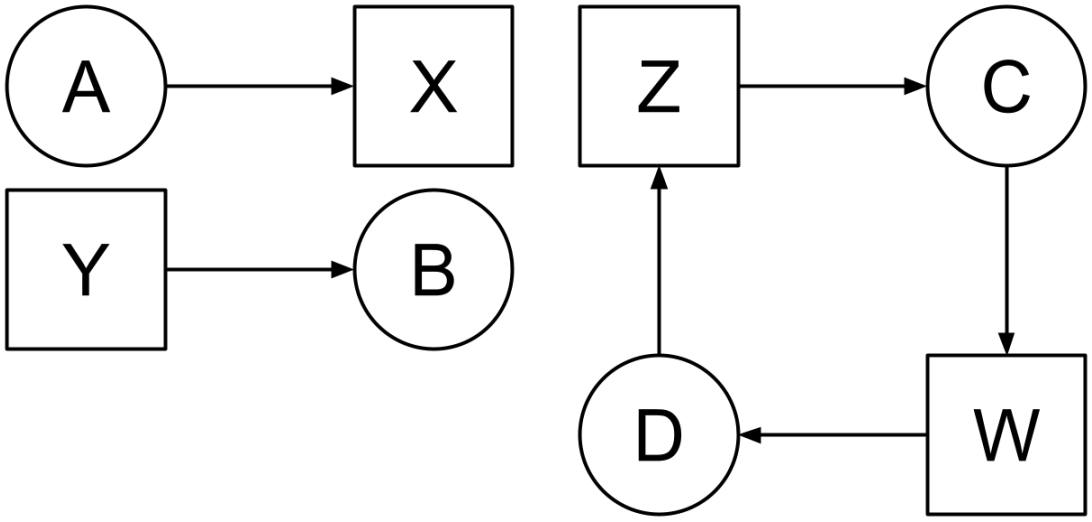

Figure 5.1: A resource allocation graph - Process A is waiting for resource X; Resource Y is assigned to process B; Process C and D are in a deadlock, waiting for resources Z and W

Deadlock Detection and Recovery

- This is a process which allows the system to enter a deadlocked state
- A deadlock detection algorithm is run periodically
- If a deadlock is detected by finding a cycle in the resource allocation graph, process that make up the cycle are randomly killed until the deadlock is resolved
- Any remaining processes are allowed to use the requested resources
- This is very inefficient and may result in data loss depending upon the type of processes being killed
- It is commonly used in relational databases, as any transactions which create a deadlock can be rolled back, allowing the system to continue

Deadlock Avoidance

- It would be better if the system never entered a deadlock state
- This works as follows:
  - The system analyses the resource usage of all threads of the system
  - This allows it to predict when threads will need access to a specific resource
  - If two threads need access to two resources in confliting orders, the system will pause one thread until the other is finished with both resources

  - This avoids entering an unsafe execution path in which a deadlock is guaranteed

- One such deadlock avoidance algorithm is Dijkstra's Banker's algorithm
  - All processes declare which resources they may need access to, and how much of said resources they may need
  - The algorithm then keeps track of the current allocation for each process
  - When a request for more resources is received, it pretends to honor the request, before attempting to fulfill the needs of all other processes
  - If it is unsafe to do so, deny the request, else grant it

Lecture - Processes and Scheduling

13:00 31/10/23 Tamer Elboghdadly

Processes

- A process is a collection of threads which belong to a program
- Several copies of the same program can be running at once, using separate processes
- A process has a space in memory which is split into multiple sections
  - Program code - The actual code for the program
  - Data section - Global and static variables
  - Heap - Memory which can be dynamically allocated during runtime
  - Stack - Memory used temporarily during runtime when functions are called
- Part of this memory is known as the execution state, which includes the program counter, stack and data section

Process Control Blocks

- The operating system maintains a process table
- Within this process table, each process has a Process Control Block
- For processes which are not currently running, the PCB contains
  - The contents of all registers at the time the process was last running
  - Pointers associated with memory management for the process
  - Any other information which may be needed to restore the process to a running state, exactly as it was before
  - It does not contain variables, as these are assumed to be in the processes' main memory block

Dispatcher

- The dispatcher actually gives control of the processor over to the process selected by the scheduler
- This involves
  - Stopping the currently running process
  - Storing it's information in the PCB associated with it
  - Re-loading the information in the new process' PCB
  - Switching between Kernel and User mode
  - Jumping to the correct point in memory to resume execution
- This set of tasks is collectively known as context switching

Scheduling

- The scheduler is responsible for selecting the next process to run on the processor
- To do this, it uses a scheduling algorithm, which takes into account the following:
  - When should processes be swapped?
  - Under what circumstance should the process be swapped? (Time used, if a process stalls, etc)
  - How long should threads be given the CPU?
  - Which order should processes run in?
- Processes which are ready to be run, but waiting for processor time are placed into the ready queue
- The scheduling algorithm is responsible for deciding when to swap the currently running process and the one at the start of the ready queue
- Typically, when a process is stopped, it is placed at the rear of the ready queue, unless it was stopped to wait for a resource
- There is usually also a waiting queue, where processes which are not yet ready to continue go to wait for whatever is stalling them

System Calls

- Since user processes are run in user mode, they do not have direct access to hardware or other resources
- Therefore, the must user so-called system calls to request that the operating system performs a function on their behalf
- System calls are usually implemented using software interrupts (sometimes called traps) so that the running process is immediately suspended while the operating system runs the function
- System calls include the following functions:
  - Creating a new process
  - Waiting for another process to exit
  - Exiting the current process
  - Creating or opening a file
  - Closing a file
  - Deleting an existing file
  - Reading from or writing to a file
  - Creating or removing a directory
  - Setting the working directory
  - etc

Part II

Internetworking

Lecture - Network Services

09:00 25/09/23 Athanasios Paraskelidis

DHCP

- DHCP stands for Dynamic Host Configuration Protocol
- The main motivation of DHCP was to provide a set of configuration parameters to automatically configure new devices as they are added to the network. The main parameters are as follows:
  - IP address
  - Gateway address
  - Subnet mask
  - DNS server address
- Before DHCP, either devices were configured manually, or the bootstrap protocol (BOOTP) was used
- The main improvements over BOOTP are:
  - Support for temporary allocation of IP addresses
  - DHCP clients can automatically discover the local DHCP server
  - Once the server is setup, there is almost 0 human interaction unless something goes wrong
  - Still compatible with BOOTP clients
- A lease is the length of time that an allocated IP address can be used before either a new address is needed, or a request to continue using the current address needs to be approved
- An IP address can be released by the client if it is nolonger needed, e.g. the device shuts down or nolonger needs to communicate on the network
- Advantages
  - Saves manually configuring every single device
  - Ability to move to a different network without having to reset any network settings on the device
  - Allows more efficient use of the IP space, as inactive devices do not need to keep a lease on an address
- Disadvantages
  - DHCP uses UDP to configure devices, so the communication is unreliable and insecure
  - Possibly allows unauthorised clients, but this can be avoided using MAC address white/blacklisting
  - Potential for malicious DHCP servers setup on a network that provide incorrect network settings

DNS

- DNS stands for Domain Name System
- This is the system which devices use to convert the human-readable domain names into IP addresses which computers can use to communicate to the server
- DNS is a globally distributed mapping database between domain names and IP addresses
- There are 3 main components
  - A **name space**
  - **Servers** that make the name space available
  - **Resolvers** which make the request from clients to servers
- As DNS is globally distributed, some data is maintained locally, but also retrievable globally
  - No single server holds all DNS records
- A DNS lookup can be performed my any device
  - Remote DNS data is usually cached locally to improve performance
- DNS has 'loose coherency'
  - The database is always internally consistent
  - Each version of the database has a serial number, which is incremented every time the database changes
  - Changes to the master copy of the database are replicated to secondary servers regularly, depending upon the timing set by the zone administrator
  - Cached data expires depending upon timing set by the zone administrator
- There is no limit to the size of the database
  - Having a very large number of records on one server would decrease performance
  - Therefore, the database is spread across many servers around the internet
- There is no limit to the number of queries which can be made at any time, or by a single user
- Queries are usually distributed between multiple DNS servers as well as local caches
  - e.g. nameserver1 and 2
- Clients can query and use the data from any server, primary or secondary
- Clients will typically have their own local cache of more frequently accessed records
- DNS uses both TCP and UDP
  - TCP is used for communication between servers, for example when replicating records from a primary to secondary server
  - UDP is used for communication between clients and servers
- The database can be updated dynamically
  - Add, delete or modify any record on the server
  - These only need to be performed on the main server, as the secondary servers will replicate the changes over time

- There are two main types of servers
  - Authoritative
    * Primary server - Where data is added and modified
    * Secondary server - Servers which replicate the primary server to share the load
  - Non-authoritative
    * Caching servers - temporarily retain records from authoritative servers to improve resolving performance and reduce load on authoritative servers
- Domains can be resolved either Iteratively or Recursively
  - Iteratively
    * The client's domain name resolver starts by querying the root nameserver
    * The root nameserver responds with the address of the nameserver on the next level down
    * The domain name resolver then queries this nameserver, and so on until the nameserver with the full domain is found
  - Recursively
    * The client's domain name resolver queries the root nameserver
    * The root nameserver itself queries the nameserver on the next level down
    * This process repeats until the nameserver with the full address is found
    * The IP address of the domain is then passed back up the chain and to the requester

Domain Names

- A domain name is the sequence of labels from a node to the root, separated by dots
  - e.g. port.ac.uk has 3 labels
    * port
    * ac
    * uk
  - There can be up to 127 labels in a domain name
  - But there can only be 255 characters in the domain overall
- The root domain or Top Level Domain (TLD) of a domain is the final label, e.g. the TLD of port.ac.uk would be uk
- A subdomain is any domain which resides below the TLD. In the case of port.ac.uk,
  - uk is the TLD
  - ac is the Second Level Domain
  - port is the actual domain name
- Name servers store information about domains in units called zones
  - Each zone usually corresponds to a subdomain, for example the .uk TLD has many sub-zones, such as .ac.uk, .gov.uk and .co.uk

Lecture - IP Addresses and Subnetting I

09:00 02/10/23 Athanasios Paraskelidis

- Layer 3 networking is the layer that handles routing data between networks, such as a local and wide area network
- Layer 3 functionality is spread over the network - in routers as well as client devices

Internet Protocol

- Connectionless
- Best effort delivery
  - Unreliable
  - Packets may not arrive at all, or in the wrong order
  - There is no built-in error handling other than error checking
- When transmitting
  - Encapsulates data from the transport layer in datagrams
  - Adds header data to the datagrams (source and destination addresses, time to live, etc)
  - Apply the routing algorithm once on the internet
  - Will continue to route the packet until either it reaches the target network card, or exceedes it's time to live
- When receiving
  - Check the validity of all datagrams
  - Reads the header and checks if forwarding is needed
    - If the destination address is on the LAN, send the datagram to the destination
    - If it isn't, relay it to the next router on it's route

IPv4 & Subnetting

- 32-bit string
  - 4*8 bit integers (1 byte, or 1 octet)
  - Each byte can be 0-255
  - Intended to be human-readable
  - Allows for 2^{32} or $4,294,967,296$ theoretical addresses, less in practice
- The IP address is split into two fields
  - The Network Prefix
  - The Host ID

  - Any devices with the same network prefix are on one network
  - Two devices cannot have the same Host ID
- The network prefix can be 8, 16 or 24 bits in length, leaving 24, 16 or 8 bits for Host IDs
- There are 3 main classes of IP addresses
  - Class A: Network prefix always starts with 0, followed by 7 bits, and 24 bits for the Host ID (1.X.X.X - 126.X.X.X)
  - Class B: Network prefix always starts with 10, followed by 14 bits, and 16 bits for the Host ID (128.0.X.X.X - 191.255.X.X)
  - Class C: Network prefix always starts with 110, followed by 21 bits, and 8 bits for the Host ID (192.0.0.X - 223.255.255.X)
- There are also 2 more reserved classes
  - Class D: Network prefix starts with 1110, followed by a 28 bit multicast ID (224.X.X.X - 239.X.X.X)
  - Class E: Network prefix starts with 1111, followed by 28 bits reserved for experimentation (240.X.X.X - 255.X.X.X)
- The Class A network 127.X.X.X is reserved for loopback addresses, every computer has 127.0.0.1 as a loopback address, some computers may have more than 1
- Each class of network has 2 less usable host addresses than would be expected, e.g.
  - A class A network provides $2^{24} - 2$ host addresses $(16,777,214)$
  - A class B network provides $2^{16} - 2$ host addresses $(65,634)$
  - A class C network provides $2^{8} - 2$ host addresses (254)
- This is because X.X.X.0 is the network address, which cannot be assigned to a computer, and X.X.X.255 is used as the network's broadcast address
- There are 3 ranges of IP addresses that can be used by anyone on a local network, one in class A, B and C
  - Class A: 10.0.0.0 - 10.255.255.255
  - Class B: 172.16.0.0 - 172.32.255.255
  - Class C: 192.168.0.0 - 192.168.255.255
- In order to connect to devices on the internet, a NAT (Network Address Translation) service is needed on the network (usually on the router)

Subnet Masks

- A subnet mask tells the computer which parts of the IP address are reserved for the network prefix
- The default subnet mask for each class is as follows:
  - Class A: 255.0.0.0
  - Class B: 255.255.0.0
  - Class C: 255.255.255.0
- The subnet mask also determines how many devices can be on the network at a time, as it sets how many bits are left for the Host ID
- A network address is the network prefix, with all host bits set to 0, e.g.

  - The network address of 10.128.47.87 is 10.0.0.0
  - The network address of 192.168.10.104 is 192.168.10.0

- But why do we need subnets?
  - If you are assigned a Class B network address (e.g. 128.147.0.0), you have up to 65534 host addresses available
  - It's not possible to effectively manage that many clients on a single physical network, and would be a waste of most of those IP addresses
  - Therefore, we should split the network into subnets which each have a much more managable number of devices
  - e.g. we could use the 3rd byte of the address as a subnet ID and have 256 subnets, each with 254 hosts
  - Since the subnet ID is configurable, we could also use 4 bits for the subnet ID, allowing 16 subnets with 4094 hosts each
- There is an alternative way to write subnet masks, which is called the slash notation
  - This is noted by adding the number of bits reserved for the network address after a slash at the end of the IP
  - For example, you could say that you have a Class B network with the ID 148.197.0.0 and therefore subnet mask 255.255.0.0, or you could simply write it as 148.197.0.0/16
- This allows us to use custom length subnet masks when creating subnets
- For example, you may have a Class B address and want to have 32 subnets. To do this, you would need an additional 5 bits for a subnet ID, and therefore should use the subnet mask /21. This allows for 32 subnets with $2^{11} - 2$ or $2,046$ hosts each

Lecture - Variable Length Subnet Masks

09:00 16/10/23 Athanasios Paraskelidis

Subnetting with VLSM

- Using classful subnetting wastes a lot of IP addresses, so a new approach was needed - VLSM
- VLSM allows for more than one subnet mask within the same network, allowing for different subnets to be different sizes, based on their needs
- This also allows for much more efficient used of the IP address space
- Also allows route aggregation, meaning that less routing information is needed
- Links between routers are in and of themselves networks - although they only need enough IP addresses for the routers, so a direct link only needs 2 usable addresses
- The smallest subnet you can have is a /30 subnet, which has 4 IP addresses, with only 2 usable
  - This is usually used only for point-to-point links, such as a connection between 2 routers
- The routing protocol needs to support VLSM, by carrying extended network prefixes
  - Some routing protocols that support VLSM are: RIPv2, OSPF, EIGRP, BGP
- When using VLSM, we can completely ignore the standard classes of IP addresses, as long as we have been assigned the full address range
- When assigning subnets with VLSM, you should consider the largest subnet first, and then use the remaining space to create any additional subnets
- If needed, you can create sub-subnets within larger subnets to allow creating more, smaller subnets
- VLSM uses a longest match forwarding algorithm
- This means that it looks through the routing table to find the longest match
  - The longest match is the entry which reduces the size of the network as much as possible - find the smallest network to route to

Supernetting

- Supernetting is combining multiple small (usually class C) networks into one larger one
- This creates a single address space with more addresses
- You can only use supernetting if the networks you wish to combine are contiguous
- Convert the addresses into binary, and count from left to right the number of identical bits. This gives you the supernet mask for the network

- The supernet address will always be the network address of the first of the contiguous networks
- You need to make sure that the supernet mask does not allow you to address IPs outside of your contiguous networks

Lecture - Supernetting & Classless Inter-Domain Routing

09:00 30/10/23 Athanasios Paraskelidis

CIDR

- Also known as Supernetting
- Considered as a fundamental solution to reducing the size of routing tables
- Also used as a temporary solution to the lack of internet addresses in IPv4
- A Class C address is too small for most companies, so most requested a class B address, but in the process wasted lots of addresses, as they did not need as many addresses as it provided
- CIDR is similar to VLSM, but applies to the entire internet rather than one specific network
- For CIDR to work, the network prefix must always be specified
- Routers on the internet nolonger use classes, and only the network prefix is important
- Since the network prefix is important for routers to know, the only requirement for CIDR to work is for the routing protocols to forward network prefixes
- CIDR will only provide any routing advantages if topologically significant addresses are used (addresses which all reside within one routing area)

Supernetting

- Supernetting is combining multiple small (usually Class C) networks into a larger one to create a larger range of addresses
- The IP addresses you wish to supernet must be contiguous
- The exact opposite of subnetting, and reserves part of the network address to use as host addresses
- You must have control over enough networks to make up an entire bit (if you wanted to reserve 2 bits, you would need 4 contiguous addresses)
- Your supernet address becomes the first IP address, followed by the new subnet prefix