Course General Notes

Contents

[Week 1 1](#_Toc132214889)

[Week 13 3](#_Toc132214890)

[Concurrency and Deadlocks 3](#_Toc132214891)

[Memory Management 10](#_Toc132214892)

[Week 14 17](#_Toc132214893)

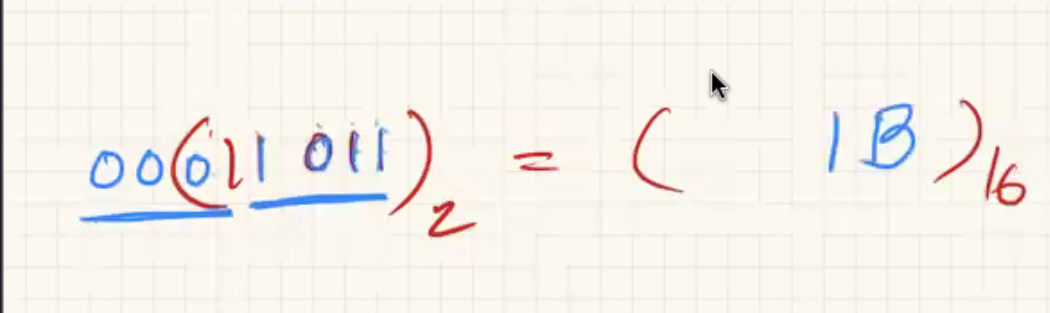
[Computer Networking 17](#_Toc132214894)

# Week 1

Easy way to do Binary to Hex:

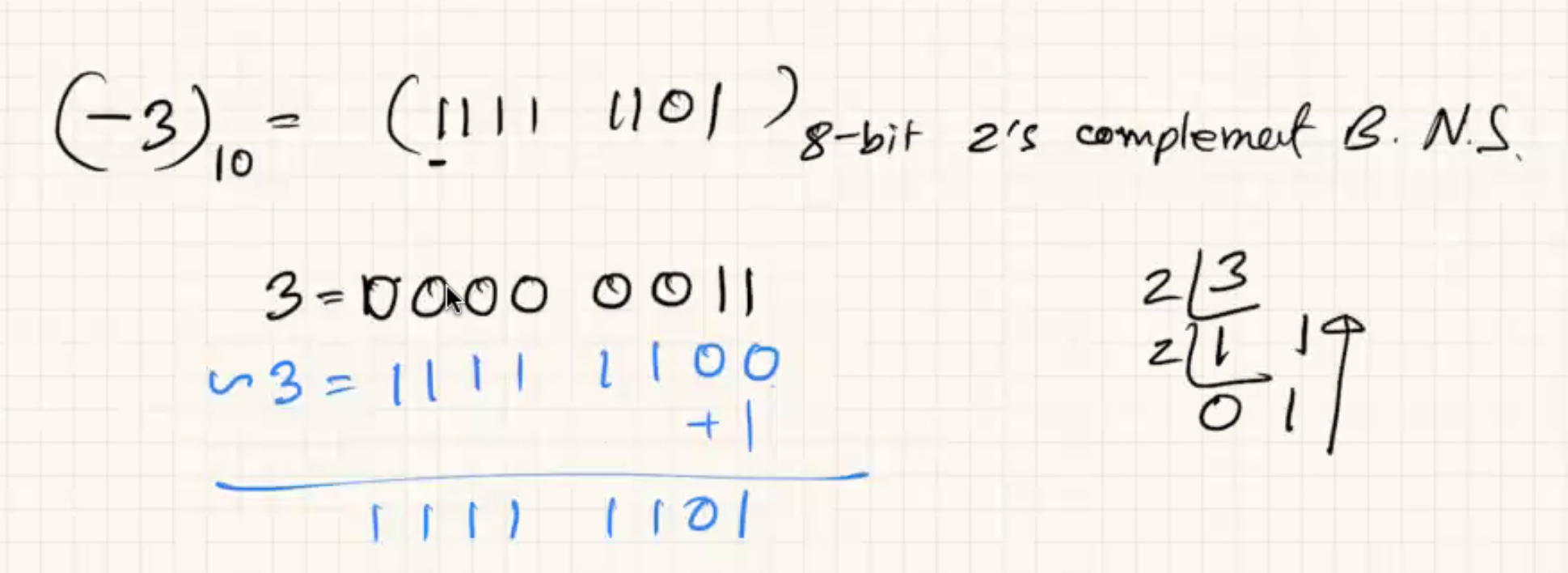
A screenshot of a computer

Description automatically generated with medium confidence

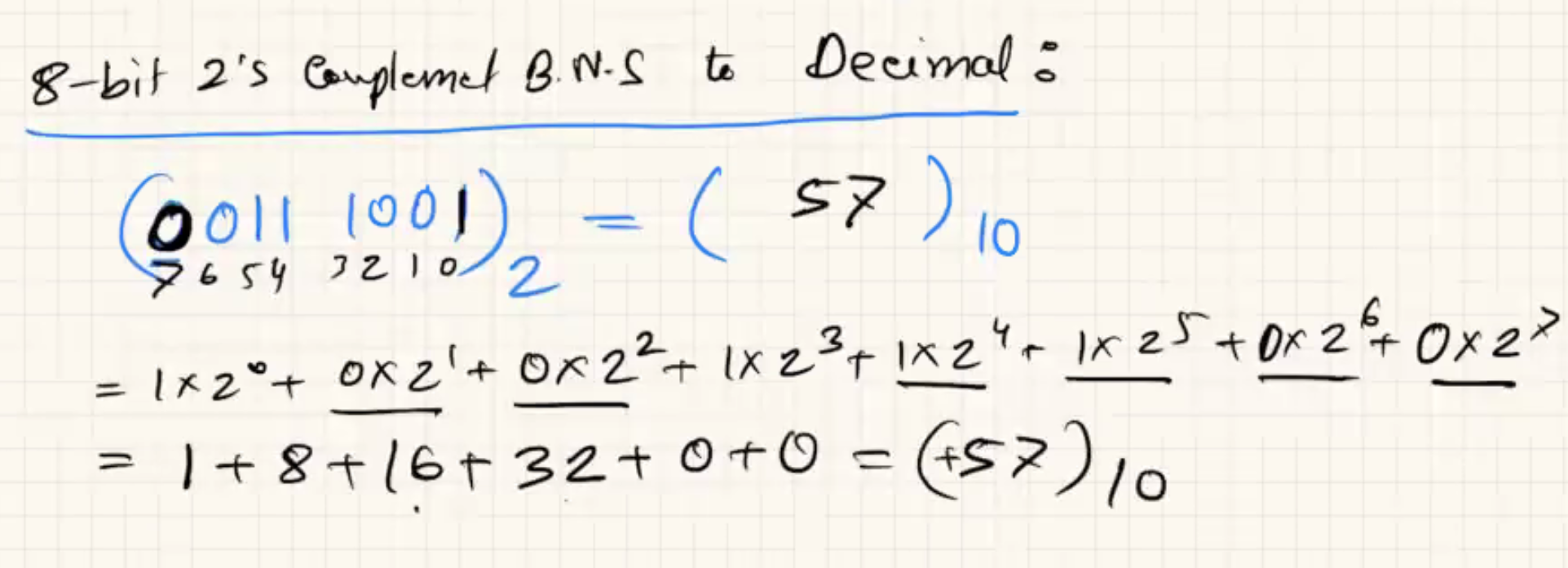


Calculating 2s complement

* Remember positive numbers start with 0, negative numbers start with 1.
* To get to one or the other, take the inverse of the number and add 1.



Converting Back:



A picture containing text, whiteboard

Description automatically generated

Table

Description automatically generated

# Week 13

## Concurrency and Deadlocks

* Reminder about Threads
  + Threads all share the resources of the process.
  + Threads run as if they were a separate program.
  + Threads can run asynchronously. Which means that all threads can run at the same time and accessing the same resources.
    - This creates potential conflicts in the shared resources and data between them since one can access the data and change it while another might have the data but not see the change.
* Features of having multiple threads
  + Ease of communication between threads since they all share the process data.
  + Effective solution to prevent blocking while reading data from an I/O device.
    - The thread reading data could be blocked but the processing thread can read the data that has already been loaded and do something or analyze the data.
  + Threads are easy to create and take little memory.
  + However, they run the risk of asynchrony.
* Asynchrony
  + Occurs when two threads are running seemingly at the same time.
  + For example. On a single processor a thread can time out or become blocked and another thread that is in the ready state can be called to run.

Diagram

Description automatically generated

* + Another example with multiple CPUs is that we can have these threads running simultaneously in different CPUs

Diagram

Description automatically generated

* Critical Sections
  + Sometimes code will be written with the expectation that once we start a particular piece of code it will run through to completion without being interrupted.
  + IF this code IS interrupted, asynchrony can occur.
  + In a worst-case scenario, data can become corrupted.
  + The programmer must identify “Critical Sections” of code which, once entered, must prohibit any other thread from entering a critical section on the same resource.
    - If we have two threads, thread A and thread B, if thread A enters a critical sector and is interrupted, sector B cannot run and use the critical sector until sector A finishes passes through that critical Section.
* Example of the Supplier/Demander
  + Example of threads being a supplier or a demander. They share two variables, a buffer count and a buffer.
  + The supply thread puts things in the buffer and increments the buffer counter if the supply thread is not done.
  + The demand thread chekcs to see if something is in the buffer and returns it.

Text

Description automatically generatedGraphical user interface, text, application

Description automatically generated

* + There are multiple of these supplier and multiple demander threads that are all “active” in the system. However, this is a uniprocessor system so only one can run at a time.
  + The bufferCount should NEVER go beyond 500 as there are only 500 buffer positions.
  + IF we do not take care of asynchrony, the bufferCount will be able to exceed 500.
  + How can we produce the previous problem?
    - If a supplier thread called supplier1 is paused right after the “bufferCount < 500” if statement (which returned true because the buffer was 499.
    - That state is saved and another thread is called.
    - If another supplyThread, called supplier2, is called and run to completion the bufferCount will be equal to 500.
    - Once we return to supplier1 the if statement had been completed so we restart at the incrementor for the buffer count. This increases bufferCount to 501 and creates a problem in the code due to asynchrony.
  + Another potential problem is a Double Update/Missing Update.
  + Here is an example of the previous:
    - If we have two threads one that adds to a balance and one that removes from a balance.

Text, letter

Description automatically generated

* + - If a code is run in the previous order (# after the comment indicates order). Then we will have an issue as the new balance will utilize the pre-existing balanced before updated by “//3” which will create a wrong amount of balance at the end of the sequence.
* Critical Sections Identified
  + In the supplier/demander example the checking oand changing of the buffer and bufferCount has to be “atomic”.
  + Since we cannot do an “atomic” call in that we have to create a protected way of running those sectors of code so they cannot be interrupted.
  + This is similar in the double update/missing problem where we have to run the entire function to completion (aka “atomically”) in order to avoid a problem.
* Mutual Exclusion Rules
  + No two threads may be in a critical section at the same time.
  + When no thread is in a critical section any threads that request entry must be allowed without delay.
  + If the critical section is occupied the thread will become blocked and then get sent to the ready state.
  + A thread may remain inside the critical section only for a small amount of time in order to avoid backlogging.
* Fundamental Mutual Exclusion
  + The system bus provides a fundamental system for providing mutual exclusion, since by fundamental design only one process or thread will be able enter a memory location at one time to change that memory location.
  + This is good for low level but we need something for higher level programming in order to deal with these critical states.
* Software Solutions for Mutual Exclusion
  + This is the **old** **solution** for the mutual exclusion problem.
  + They utilized the Peterson’s algorithm which provided a fundamental way to protect two threads from accessing the same resource at the same time.
  + Peterson provided each thread with a Boolean flag to indicate that thread wanted to access its critical section.
    - Once a thread wants to enter a critical section they set their flag and look on every other thread to see if they are occupying the critical section.
    - If nobody is they enter, if somebody else is they become blocked and are moved from the blocked state to the ready state and into the queue.
  + Additionally, there was a turn variable to overcome the problem of checking/changing the flag being a critical section itself. When a thread wants to enter a critical section, it raises its flag and offers the turn to the other threads. Which helps the access problem in which two threads simultaneously ask the system to enter into the critical section.
* Hardware Solutions
  + More modern solutions involve using hardware solutions.
  + **Disable interrupts**
    - Prevent the CPU from being interrupted and you prevent asynchrony.
    - This is not the best solution but it is extremely simple.
  + **Test and Set**
    - Provide a single ML instruction to check a memory location (Boolean flag) and set it if it is no set.
    - A result is returned to indicate success or failure.
    - This single variable is atomic in the sense that it will check and update the flag without being interrupted.
    - All threads have access to this flag and can check it without having to ask all existing threads.
  + **Exchange**
    - The location in memory (maybe a flag) is swapped with a register.
    - The register will contain the state of the critical section after the switch is completed while the thread will hold the access.
    - Even if the thread is interrupted the register will remain unchanged until the thread is completed.
* Semahpores
  + Common high level solutions that is very commonly utilized today.
  + The OS provides some of the service but most work is done in the user space.
    - Primarily two functions are used, signal and wait.
    - Signals can be queued, or, if a thread is waiting, a signal will cause it to be released.
    - Wait causes the thread to block if there are no signals to be consumed, the thread is queued to await the next signal.
  + How to use them:
    - When we create the semaphore, we send a signal to it. Since no thread is waiting on it, it simply goes to the top of the semaphore queue.
    - The thread calls the wait and consumes the top signal in the queue.
    - If another thread calls wait and attempts to consume the top signal and there is no signal available, then the thread will be placed in a blocked state and placed in a blocked semaphore queue until a signal is sent into the semaphore.
  + Semaphore Internals
    - Internally a semaphore usually keeps a counter of the number of signals which have been sent.
    - Wait causes the counter to decrement.
    - If the counter ever becomes negative, the thread which caused it to go negative, and all subsequent threads, will be blocked and placed on a queue. This is where the OS is invoked to perform the block.
    - When a signal is sent, the counter is incremented and if the counter is not positive, the next thread on the queue is released.
    - Since the act of checking and modifying the counter is asynchronous, we have to use hardware options in order to make the action atomical in order to avoid issues.
* Deadlocks
  + The deadlock occurs in a system when all the threads in the process are waiting for each other for them to complete.
  + Usually, this results from one thread waiting for another thread to release a resource.
  + This is a permanent block which cannot resolve itself overtime.
  + There are no good solutions today to deadlocks.
* Deadlock Resource Types
  + Reusable
    - Once the use of the resource is complete, the resource can be used again by another thread.
    - Examples include memory, devices, data structures, semaphores, etc.
  + Consumable
    - Once used, the resource is gone.
    - Examples include interrupts, signals, messages, and data in IO buffers.
      * Once the data is read there is no easy way to put it back in the buffer.
  + We deal with deadlocks differently depending on what resource type we are utilizing.
* **Items Required for a Deadlock**. All four need to occur in order to have a deadlock.
  + Mutual Exclusion
    - Only one thread can use a resource at a given time.
  + Hold-and-wait
    - While a thread is “waiting” on the allocation of a new resource, it retains all existing locks.
  + No Preemption
    - A resource cannot be removed from a thread forcefully.
    - For example, on a semaphore, once we have decremented the semaphore counter to 0 there is no way for the OS to intervene and remove the lock on the semaphore.
  + Circular Wait
    - A closed chain of threads in which the last thread is waiting on the list.
    - It is usually bad luck.
* Solutions for Deadlocks
  + Deadlock Prevention
    - Eliminate one of the four requirements for a deadlock.
    - Eliminating Mutual Exclusion:
      * This is not often possible as mutual exclusion is usually required.
    - Hold and Wait
      * The system can require that all locks can be requested simultaneously.
      * If a thread holds a lock, it cannot obtain another lock.
    - No preemption
      * If a thread requests a new lock, and the request is denied, then the thread must release all existing locks.
      * The OS may have authority to remove existing locks. Although this is a bad idea.
    - Circular Wait
      * The resources are ordered such that a thread holds a lock on a higher numbered resource, it cannot request a lower numbered resource.
      * There are different levels of locks which prevents higher locks from accessing lower-level locks.
      * This is usually what is most recommended by Computer Scientists.
  + Deadlock Avoidance
    - Before each allocation check to make sure that a deadlock is not possible, do not make an allocation that could cause a deadlock.
    - Required that the OS knows ahead of time which resources a thread will request before it is finished.
    - Every time a thread makes a new request, the OS runs an algorithm called a Banker’s algorithm to see if that request will cause the system to deadlock.
      * If it will, the request is denied or queued.
  + Deadlock Detection
    - Allow deadlocks to happen and then resolve it when it does.
    - This is the easiest method to implement but it is also the most difficult to deal with after the fact.
    - Checks periodically for an existing deadlock.
    - Once the detection system finds a deadlock, we can do a couple of things:
      * Rollback to a previous unlocked state (requires transaction logs)
      * Abort all deadlocked threads.
      * Abort a thread, check for deadlock, repeat until correct.
      * Preempt resources until the deadlock is resolved.
    - This is the most common solution to deadlocks in common apps and systems.
      * Apple beachball or frozen applications. Once you quit them or eliminate them in the task manager you are basically removing all frozen deadlocked threads.
* Dining Philosophers Problem
  + Presented by Dr. Edsger Dijkstra as a concurrency control problem.
  + There is a house with 5 philosophers numbered 0 thru 4. They all alternate between thinking and eating. When it is time to eat, a circular table is set for them with each philosopher having a setting consisting of one plate and one fork. The meal they will heat is, a particularly difficult kind of, spaghetti. To eat this meal they must have two forks. There are two forks next to each plate, so that presents no difficulty, as a consequence, however, no two neighbors may be eating simultaneously.
  + The problem is that if all philosophers come to eat at the same time, they will all pick up 1 fork each and will be unable to eat (creating a circular wait) since none of them can put down the fork and allow another one to eat.
  + The solution Dijkstra proposed was resource ordering the forks. SAO that the last philosopher had to request the zero numbered fork before he could request the numbered 4 fork. This will give one philosopher the capacity to eat and then another will eat thereon.
  + Another solution is to have a semaphore to prevent the fifth philosopher from entering the room.
    - A semaphore is constructed with 4 signals queued and the philosopher must request access to the room before requesting access to any fork.

## Memory Management

* Reasons of Memory Management
  + In multiprogramming system, there will never be enough memory.
  + The OS will need to allocate memory to multiple programs running in the system.
  + The OS will need to move parts between main memory and secondary memory.
* Memory Management Requirements
  + Relocation
    - A process could be loaded into any location in main memory.
    - A process can be moved during its lifetime.
    - This is no longer true since we have a lot of programs running and we do not know the order of the programs as well as what is already occupied when the program starts.
  + Protection
    - A process should not be allowed to access memory outside of its allocation.
    - The OS cannot pre-screen the memory accesses, it must be dynamically. As you make a memory request the CPU must check to see if it is within the process memory space or outside.
  + Sharing
    - Some processes might use the same code/data.
  + Local Organization
    - Modules, Shared Objects or Dynamically linked libraries should be allowed.
  + Physical Organization
    - Mapping logical memory to physical memory.
* Logical vs. Physical Adresses
  + Due, primarily, to relocation, a program will have to use a logical address to access memory.
  + Logical addresses need to be converted, dynamically, to physical addresses.
  + A simple solution to logical addresses could be the offset from beginning of the program.
  + The CPU’s Hardware Memroy Management unit is responsible for converting, during runtime, the logical address to a physical address.
  + A simple example of this is when having a pointer in a code, the logical address will be A. Once the hardware gets this pointer and expects the program expects the data at the address, the CPU’s HMM unit will add the processes’ base memory address, B, to the logical address to get the physical address.
* Partitioning Strategies
  + Fixed Partitioning
    - We can divide main memory into a static number of parts.
    - A process will be allocated one part of the equal or greater size than it needs.
    - Equal sized partitioning
      * Causes internal fragmentation – Wasted space inside a partition from a process not using all the memory.
    - Unequal Sized Partitioning
      * Need to worry about which partition to place the process in.
      * This creates limitations as there will be a limited number of adequate partitions for the process to take.
  + Dynamic Partitioning
    - Partitions are created for exactly as much memory as the process needs.
    - Causes external fragmentation – Wasted memory between allocations.
      * If we have little slivers of memory spaces left over then we will not be able to efficiently run code.
    - Requires compaction, a costly process, in which we get rid of all these slivers.
    - OS data structures are complex because a process could start anywhere.
    - Where does the OS place a partition?
      * Best-Fit – The area with the size closest (causes external fragmentation)
      * First-Fit – The first spot in memory that is large enough gets selected.
      * Next-Fit – Begin looking from the location of last allocation.
    - Has too much overhead so it is not done anymore.
  + Buddy System
    - Compromise between fixed and dynamic partitioning.
    - Linux uses it to create memory for the kernel.
    - This is not done on a large scale but it does have its niche uses.
    - How is it done?
      * Suppose we have a memory requirement of 100kb.
      * Memory is divided recurringly in multiples of 2.
        + Example a 2 MB data will be divided into 1024kb, 512kb, 256kb, etc.
      * Once we reach 128kb the memory cannot be divided as it will not be able to fit the 100kb process.
    - This decreases, but not eliminates, the internal fragmentation.
    - This makes the OS structure easier because processes begin and end on a 2^n boundary.
  + Simple Paging
    - Break it down into a lot of equal size frames (say 4kb)
    - A process is broken down into the same size pages.
    - Since this is RAM, no frame is better suited for the task of storing the page than any other, all have O(1) access time.
    - Pages are loaded into non-contiguous frames.
    - The OS records the frame number for each page in a Page Map Table stored in the PCB.
    - The hardware MMU needs to be able to query the PMT so the format has to be agreed on.

Diagram

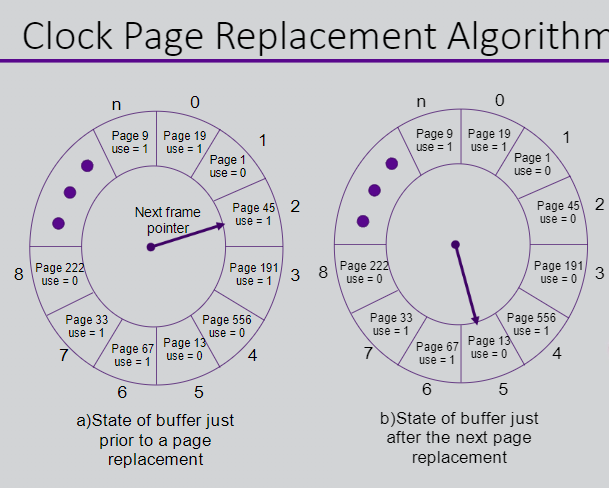
Description automatically generated

* + Simple Segmentation
    - Not that different from pages but now we are able to have unequal sized partitions.
    - Logical Adress now needs to be Segment# + Offset.
    - Storage is similar to Dynamic Partitioning.
    - This is not as universal as a paging system. Normally the CPU designer will give options for page sizes instead of doing segmentation.
* BENNEFITS OF PAGING:
  + No external fragmentation since everything is always compacted.
  + A minimum of internal fragmentation
  + Easy protection.
    - If the process is making a request beyond its page table then we know immediately that the request is erroneous.
  + Easy relocation.
    - We can now relocate pages instead of the entire process so it is much less cost intensive.
    - There is also very little reason to do relocation.
  + Easy sharing.
    - If we have to processes that are allowed to share a page then we can just add the page frame into the page table and that will allow sharing immediately.
* Converting Logical to Physical Adress
  + The logical address is relative to the start of the process.
  + Calculation to get physical address:
  + The divisions and the modulus operations are simple bit shifts since the PAGE numbers are always multiples of 2.
  + To calculate the OFFSET into the page, we can use an XOR.
  + The page number can be used as an index (like an array) into the PMT to find the Frame number in main memory.
  + The CPU will then bit shift that frame number and add the offset to find the physical address.
  + More examples: <https://www.youtube.com/watch?v=c3ZHYjLQof4&ab_channel=LearningWithMahamud>
* Virtual Memory (also called Swap Space)
  + **Assumes paging or segmentation.**
  + Since the process will only ever access one memory location at a given time, the rest of the memory of the process doesn’t need to be in the MAIN memory.
  + We can swap pages of the process out onto the hard drive and bring them back later, if needed.
  + Example: The splash screen at startup of a process isn’t ever needed again after the process is running.
  + What stays in main memory?
    - Resident Set – The portion of the process that is in main memory.
    - Working Set – The portion of the process which is in use. In order for the process to run the working set must be in the resident set.
    - PMT
      * The PMT contains a “Present” bit to indicate what pages are present in main memory.
      * The PMT contains a “Modify” bit to indicate if the copy of the page on the hard drive is the same as the copy in main memory.
        + If the modify bit is unset, then we can simply erase the page on main memory instead of having to invoke I/O to copy the modified data to main memory.
      * If the MMU encounters a page which the process wants but is NOT in the resident set, it is called a “Page fault” and the CPU switches to running the OS (like an interrupt).
  + Benefits of Virtual Memory
    - The programmer perceives a much larger memory space.
    - The system can remove unused pages.
    - More processes can run in the system resulting in better performance.
  + Lookup Problems
    - Now any memory access by the process requires that the MMU determine the physical address, which requires a query to the PMT.
    - Each memory access requires two memory accesses.
    - To save time, the CPU implements a Translation Lookaside Buffer which is a type of content addressable memory which stores a cache of those entries in the PMT which have been recently retrieved.

Diagram

Description automatically generated

* Replacement Policy
  + When main memory runs out, the OS needs to remove some pages. This is called stealing.
  + If we choose poorly, the result will be worse performance.
  + The optimal choice would be to steal the page which is not going to be used for the longest period of time.
  + Possible Algorithms
    - Optimal – Not feasible.
      * Crystal ball algorithm. The name clearly implies that we have to know future execution of all processes which is not possible.
    - LRU – Not feasible.
      * Least recently used. Timestamp on each page to see when it was last accessed.
      * This creates too much overhead in the CPU and is unfeasible.
    - FIFO – Easy to implement but poor performance.
    - Clock – Easy to implement but requires USE bits.
      * USE bits indicate when a page was last utilized.
      * This lets us know which pages are likely in use and which are not.



More info: <https://www.geeksforgeeks.org/second-chance-or-clock-page-replacement-policy/>

* Resident Set Management
  + Two problems
    - How much main memory does each process deserve?
      * Smaller allocations mean more processes.
      * Larger allocations mean less page faults.
    - When is it time to steal, which process do we steal from?
      * Local – the process page faulting will lose a page.
      * Global – Any process can lose a page.
  + Controlling Page Faults by Resident Set Size
    - If a process fits entirely in main memory, it will not generate any page faults.
    - If a process only has one frame, it will generate the maximum number of page faults.
    - There is a non linear progression between those two points.

Chart, line chart

Description automatically generated

* + PFF Algorithm
    - Examine the time between the last page fault and the current one.
      * If less than a preset F value, add frame.
      * If greater than the preset F value, discard all pages with use bit 0, reset all use bits to 0 and continue.
    - Setting an upper bound and lower bound F would be better than just F.
    - Unfortunately, it performs poorly during locality shifts.
  + VSWS Algorithm
    - WSWS solves PFF’s problems by setting:
      * A minimum value for clearing unused pages (M)
      * A minimum number of page faults which must have occurred before clearing unused pages (Q).
      * A maximum duration before clearing unused pages (L)
    - Below M time units, always add a frame.
    - Between M and L
      * If there have been less than Q page faults, add a frame.
      * IF there have been more than Q page faults, discard unused pages, reset use bits, and reset the timer and Q counter.
    - At L
      * Discard unused pages, reset use bits, and reset the timer and Q counter.
  + Load Control
    - Lowest Priority
    - Faulting Process
      * Greater probability that it does not have working set resident.
    - Last Process Activated
      * Least likely to have working resident set.
    - Smallest Local Resident Set
      * Requires the least effort to reload at a later time.
    - Largest Process
      * Most bang for your buck.
    - Largest Remaining Execution Window
      * Has the longest processing time.
* Shared Pages
  + Using virtual memory and/or paging, we can understand how sharing of structures can occur, both processes have entries in their page table for the same frames.
    - Code – Since code doesn’t change, sharing of code pages is easy.
    - Data – Since data changes frequently we can adapt a Copy On Write Scenario
      * Copy On Write – Allows sharing of pages.
        + The page table is expanded to allow a “read only” attribute.
        + If a page is marked RO, any attempt to change that page will result in a trap to the OS.
        + If not, the OS can then copy the page and mark both the original and the copy as RW since it is no longer shared.
  + The UNIX Fork Function
    - Useful in systems which don’t have thread support, the UNIX fork function creates an exact duplicate copy of the current process.
    - This is most easily done by creating a new process, and of course a new page table, but duplicating the data in the original page table.
    - Both tables, in their entirety, **are marked copy on write**.
    - There’s no need to actually COPY the memory until something changes, and then only the page changing must be copied.

# Week 14

## Computer Networking

* What is the Internet?

Diagram

Description automatically generated Diagram

Description automatically generated with medium confidence

* + “Nuts and Bolts”

Diagram

Description automatically generated

* + A Service View
    - Infrastructure that provides services to applications
      * Web, VoIP, email, games, e-commerce, social networks...
    - Provides programming interface to apps
      * Hooks that allow sending and receiving app programs to “connect” to the Internet.
    - Provides service options analogous to the postal service.
* What is a Protocol?
  + Protocols define:
    - Format
    - Order of msgs sent and received among Network entities
    - Actions taken on msg transmission, receipt.
    - Examples: TCP, IP, HTTP, HTTPS, Skype, 802.11
  + One example might be: Request access, response, specify request, and get return.

Diagram

Description automatically generated

* Network Structure
  + Elements:
    - Hosts: clients and servers (often data centers)
    - Accesss networks and physical media (wired, wireless communication links).
    - Network Core (Interconnected routers)
  + How do we connect end system to edge router?
    - Residential access nets.
      * Example: Home routers, wireless access points, and ethernet cable.
      * This all leads to a DSL modem which acts as the net.
    - Institutional access nets (school, company).
      * There exists wireless access but most connection through ethernet.
      * There is an institutional router which connects all data and the institutional mail and web servers.
      * And there exists a single institutional link or ISP to the internet.
    - Mobile access nets.
  + Kep in mind:
    - Bandwidth.
    - How many users.
* Physical Media
  + Guided Media
    - Signals propagate in solid media: copper, fiber, coax.
  + Unguided media:
    - Signals propagate freely, e.g., radio.
  + Physical Link
    - What lies between transmitter and receiver.
  + Bit
    - Propagates between transmitter/receiver pairs.
* The Network Core
  + Mesh of interconnected routers.
  + **Packet-switching: hosts break application-layer messages into packets.**
    - Store and forward technique:
      * The entire packet has to arrive at a switch before it is transmitted to the destination link.
      * Example, if we take a packet of size **L = 7.5 Mbits** at a rate of transmission of **R = 1.5 Mbits**, then the one-hop transmission delay will be 5 seconds.
      * End to end delay is (assuming zero propagation delay)
    - Queuing Delay, Loss
      * If arrival rate (in bits) exceeds the transmission rate of the link for a period:
        + Packets will queue, wait to be transmitted.
        + Packets can be dropped (lost) if memory (buffer) fills up.
    - Two Key Network-Core Functions
      * Routing: Determines the souce-destination route taken by packets.
      * Forwarding: Move the packets from the router’s input to appropriate router output.

Diagram

Description automatically generated

* + Forward packets from one router to the next, across links on a path from source to destination.
  + Each packet is transmitted at full link capacity.
* Internet Structure: Network of Networks
  + End systems connect to Internet via access ISPs (Internet Service Providers): Residential, company and university ISPs.
  + Access ISPs in turn must be interconnected, so that nay two hosts can send packets to each other.
  + Resulting in a network of networks that is very complex. Evolution was driven by economics and national policies.
  + Structure:
    - Given the million of ISPs how do we connect them together?
      * Connect ISP into multiple ISPs in multiple parts of the world and then interconnect those larger ISPs through IXPs.
      * Sometimes we need to utilize an inbetween regional net, between an access net and a large ISP.

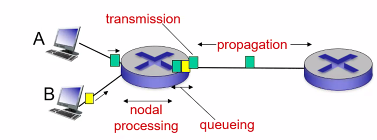
Diagram

Description automatically generated

* + - Within the structure we have content provider networks (e.g., Google, Microsoft, Akamai).
      * These may run their own network, to bring services and content closer to end users.
    - Example of structure:
      * Tier 1 commercial ISPs provide national & international coverage.
      * Content provider networks (e.g., Google): private networks that connect data centers to the internet and often bypass tier 1 regional ISPs.

Diagram

Description automatically generated

* Delay, Loss, Throughput in Networks
  + Loss and Delay:
    - Packet arrival rate to link (temporarily) exceeds output link capacity.
    - Packets queue, and wait for their turn.
    - All data that has already been sent has to be stored somewhere.
    - **Four sources of Delay:  
      **
      * Drop: Nodal processing
        + Check bit errors.
        + Determine output link.
        + Typically takes less than a millisecond.
      * Queueing Delay:
        + Time waiting at an output link for transmission.
        + Depends on the congestion level of the router.
      * Transmission Delay:
        + We have the length of the packet over the link bandwidth and that dictates how much it takes to get through the transmission process.
        + L: packet length (bits)
        + R: link bandwidth (bps)
      * Propagation Delay:
        + How long it takes our packet it takes to move through the entire link.
        + This cannot be avoided since it is due to the laws of physics. Literal propagation speed.
        + d: length of physical link
        + s: propagation speed in medium (~)
      * Caravan Analogy:

Diagram

Description automatically generatedDiagram

Description automatically generated

* + - * + First image shows 100km propagation speed while the second increases to 1000km.
        + Even though propagation speed has increased significantly there is still a bottleneck at the toll booth and performance is still affected by it.
    - Queuing Delay (Revisited)

Diagram

Description automatically generated

* + - **Packet Loss:**
      * Queue (aka buffer) preceding link in buffer has finite capacity.
      * Packet arriving to a full queue is dropped (aka lost).
      * Throughput: Thre reate (bits/time unit) at which bits are transferred between sender/receiver
        + We will look at this as the average rat3e over a longer period at which bits are transferred.
        + The instantaneous rate determines at a single moment what the rate was, but this is unreliable.
* ISO/OSI Reference Model

## Computer Networking 2 – Application Layer

* Network Application Basics
  + Applications sit outside the network and not on top of it.
  + Client-Server Architectures
    - Clients communicate to servers and vice versa.
    - Servers:
      * Always on host
      * Permanent IP address.
      * Data centers for scaling
    - Clients:
      * Communicate with server
      * May be intermittently connected.
      * May have dynamic IP addresses.
      * Two clients never communicate directly with each other.
  + P2P Architecture
    - …
  + Processes Communicating
    - Proces: Program running within a host.
      * Within same host, two processes communicate using inter-process communication.
      * Processes in different hosts communicate by exchanging messages.
    - Client Servres:
      * Client Process: process that initiates communication
      * Server process: process that waits to be contacted
    - Aside: Apps with a P2P architecture have client processes & server processes.
  + Sockets
    - Process sends/receives messages to/from its socket.
    - Socket analogous to door
      * Sending process shoves message out door.
      * The sending process relies on transport of infrastructure on the other side of the door to deliver message to socket at receiving process.
      * The socket removes the complexities of the internet by encapsulating the complexities of the overall network/internet. The OS needs to have both transport links, network addresses, the actual link and the physical connection.

Diagram

Description automatically generated

* + Addressing Process
    - To receive messages, we must have an identifier.
    - The host device has a unique 32-bit IP address.
    - The identifier includes both IP address and port numbers associated with the process on host.
  + Internet Transport Protocols Services (MORE ON THAT NEXT WEEK)
    - TCP Service
      * Reliable transport
      * Flow control
      * Congestion control
      * Does not provide: timing, minimum throughput, guarantee, security.
      * Connection-oriented: setup required between client and server processes.
    - UDP Service
      * Unreliable data transfer
      * Does not provide: reliability, flow control, congestion, timing, throughput guarantee, security, or connection setup.
* Web and HTTP
  + HTTP: hypertext transfer protocol
  + HTTP only has request and response model.
  + Client/Server model.
  + Uses TCP.
  + HTTP Connections
    - Non-persistent HTTP (old system)
      * After data transfer is completed the connection is broken.
      * Creates a lot of overhead
    - Persistent HTTP
      * Does not break the connection so that multiple requests can be made over the same line.
* Electronic Main
  + SMTP is still used, POP3 and IMAP no longer used (don’t bother)
  + STMP is an ASCII based protocol, no way to transmit binary data over SMTP. This means you cannot send a binary attachment via email.
    - How STMP accounts for this is that it changes every 6 bits of binary data into 8 bits of ASCII data. (33% expansion).
* Domain Name System (DNS)
  + The most essential protocol to the entire world that we live in.
  + Distributed, hierarchical database.

Diagram

Description automatically generated

* + LEARN
    - UNDERSTAND HOW THE QUERRIES HAPPEN
    - 13 NAME SERVERS, the loss of one does not take down the entire internet.
    - Understand recursive query vs (other one)
    - DNS Caching, updating records
      * You get a TTL (time to live) which the server hold on to to remember the solution to certain queries.
    - DNS Records
      * Type A
        + Hostname records
        + Result in an address
      * Type-CNAME
        + Cannonical name
        + It is an alias for the full url with [www.nyu.edu](http://www.nyu.edu) = fadflkjasd523sdfa or something like that.
      * Type-NS
        + Name is domain (e.g. foo.com)
        + Value is host name for the server.
      * Type-MX
        + Email record for domain, it is a mail server
* P2P Applications
* Video Streaming and CDNs