1 Operating Systems

Operating System (OS): Manages the allocation of hardware resources and provides an abstraction so that programs can access these resources.

Operating system concepts

- Processes: Running programs
- Address spaces: Memory and storage
- Files and file system: Accessing files
- Input and Output: Read and write operations
- Protection: Security

Processes: A program in execution. A process is associated with an address space and given a set of resources. It can also be considered as container that holds all the information needed to run a program. It is also dynamic as running a process can depend on other factors such as the computer (runtime), environment (windows or mac) and may act differently on different devices.

Program: A set of instructions, considered static as it won't change (by itself). For example, a recipe is static, it'll guide the cook on what to do. Cooking is a dynamic process, it can change depending on the person.

Memory: Where variables and data is stored.

Computer Memory Hierarchy

1. Registers:
   Capacity of less than 1KB, stores things that are accessed very frequently for quick access (1nsec)
2. Cache:
   Capacity of 4MB, access time of 10nsec
3. Main memory:
   Capacity of 1-8GB, where most data is stored and accessed, 10nsec
4. Magnetic disk:
   Capacity of 1-4TB, could be seperate to the computer, 10msec

1.1 Process Management

Processor/CPU: The central processing unit (CPU) fetches instructions from registers (memory) and executes them.

There are two modes of execution:

- Kernel: All instructions can be executed (including passwords so must have security to enter the Kernel execution process)
- User: A subset of instructions can be executed, for most normal operations

Registers: The memory, they keep some process state information. **Process Memory:** Each process has its own address space.

The process memory has three segments (current states):

- Text: Program code, usually read only so it cannot be modified
- Data: Constant data strings, global variables
- Stack: Local variables, information about the variables and their states

Multiple Processes: Gives each process its own virtual CPU, while scheduling processes for execution. If a process wants to read something, then it doesn't actually need the CPU at that moment, so we can assign the CPU to another process.
Multiprogramming increases system efficiency and it is useful as the number of processes will most likely exceed the number of CPUs.

Kernel: ensures that several processes don't get in each others way, it can also provide services such as reading which will be used by application programs and non-priviledges (protected) parts of the operating system. As the kernel cannot be terminated, it is not a process.

User - Kernel distinction: Most CPUs have two modes, the Program Status Word (PSW) register returns the current mode.
Code running in user mode cannot access protected instructions, it can only access the parts of the memory allowed by the kernel mode code.
Code running in kernel mode can issue all instructions and access all memory. Instructions that can interfere with other processes are protected.

System calls: Packages instructions for the kernel mode to execute protected instructions. Eg. Read is a system call in Unix, and the user mode will call the kernel through 'Read', while the kernel executes the necessary operations. Similar to providing an interface or abstraction for users to access.
Eg. printf is a library in C, the library will organize how the printf instruction is executed. The write portion will be delegated to the kernel, whereas other operations such as rounding or formatting will be assigned to the user mode to minimize execution costs and as it is not necessary to access the kernel for those operations.

1.2 Process Lifetime and Threads

Process Creation: Four principal events can cause processes to be created.

- System initialization (eg. turning on a computer)
- Execution of a process creation system call by a running process
- A user request to create a new process (eg. double click or instruction in cmd shell)
- Initiation of a batch job (eg. scheduled tasks)

`fork()`: Creates a new process that is a clone of the parent process, and returns the `pid` value.

`execve()`: Used after a `fork` to replace the child process (clone) with a new program. Creating a clone is useless as it's running the same process twice, `exec` makes `fork` useful as it can replace the child with a different program code.

Process Termination: Typical conditions that terminate a process are:

- Normal exit (voluntary) - eg. end of program
- Error exit (voluntary) - eg. try catch exceptions
- Fatar error (involuntary) - eg. exceptions that aren't caught, seg fault, dividing by 0
- Killed by another process (involuntary) - eg. `kill processid`

`exit()`: Terminates a process. Used if a parent needs a child process to terminate.

`wait()`: Can specify which process we're waiting for to terminate. Provides the process id of a terminated child so the parent process can check if it is the right one.

Process states: There are three states a process may be in:

- Running: actually using the CPU at that instant, an active process.
- Ready: ready to run, temporarily stopped while another process is running.
- Blocked: unable to run until some external event happens (CPU not needed), eg. waiting for input. After the blocked process has it's info, then it is unblocked and goes into a ready state.

If a process goes from the running state to the ready state, then it means the OS terminated the current process to another one. Instead of running one process at once, it shares the CPU's resources across all the processes.

Interrupts: When hardware devices (outside the CPU) needs immediate attention from the CPU, it will generate a signal to interrupt the CPU. An interrupt is asynchronous with the currently executing process. The CPU will remember the current state of the program, save it then load the interrupt handler. Once the interrupt is dealt with, they will return to the previous process.

Interrupt Vector: The address of the interrupt handler.

Interrupt Handler: Required to save the status of the current process, service the interrupt, restore the saved information and execute a return from the interrupt to the prior process.

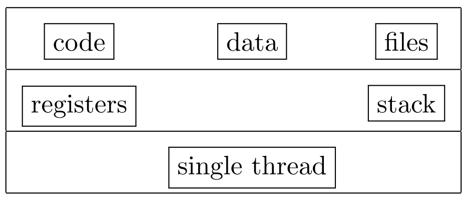
Pseudo-interrupts: When the CPU itself generates an interrupt. Usually a program can generate a pseudo-interrupt, such as divide by zero. Users can also generate one intentionally by using a system call, such as ctrl-c. We can catch interrupts with a trap command.

Process Table: One entry in table per process, contains state information to resume a process, such as memory and file management.

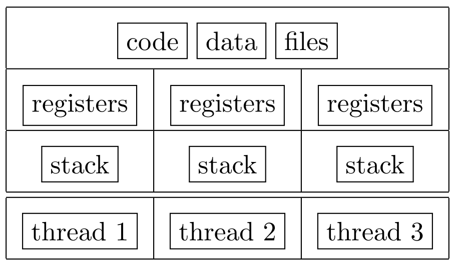
Thread: A sequential execution stream within the process. Threads are like multiple processes run in parallel that have access to the same memory, data and variables (treated as global variables). Threads can also communicate with each other without invoking the kernel.

Thread vs Processes: A process has one container by may have more than one thread, and eahc thread can perform computations relatively independently of other threads in the process. Multiple processes are beneficial in that less time is required to create a new thread, terminate, switch and communicate between threads.

Single Thread Process:



Multi-Threaded Process: Address space and memory are shared by threads, eg.same code, global variables, files etc. Threads own their own copy of registers, stacks (local variables) and state (running, waiting etc).



pthread(): A POSIX standard API for thread creation and synchronisation. All functions start with `pthread`, need to include `pthread.h`, all threads have an id of type `pthread t`.

Thread calls:

Thread Call	Description
`Pthread_create`	Create a new thread
`Pthread_exit`	Terminate the calling thread
`Pthread_join`	Wait for a specific thread to exit, wait command on multi-processes
`Pthread_yield`	Release the CPU to let another thread run
`Pthread_attr_init`	Create and intialize a thread's attribute structure
`Pthread_attr_destroy`	Remove a threads's attribute structure

Thread issues: Global variables are shared across threads, and thread switches can occur at any point. A thread can modify shared data while the thread that is blocked is also modifying it. Therefore it is the programmer's responsibility to synchronise threads and ensure that threads don't override each other's data.

Concurrency: Also another issue with multiple threads. There can be different outputs depending on which order threads are run on. Programmers cannot create a problem where the output is dependent on thread order and consider how threads may interact (eg. race conditions, deadlocks)

1.3 Process Communcation

Race Conditions: When multiple processes have access to the shared object and can read and write it. The race condition arises when the output depends on the order of operations.

Critical Regions: Used to prohibit access to the shared object at the same time. For example, if A is accessing the shared object, then we lock the object so B cannot access it.

Avoiding Race Conditions: Basically putting atomic code into the critical region. Any code that is required to be run to completion, but we want to minimize the amount of code we put there.

1. No two processes may be simultaneously inside their critical regions (can't lock an object twice).
2. No assumptions can be made about speeds or the number of CPUs.
3. No process running outside its critical region may block other processes.
4. No process should have to wait forever to enter its critical region (cannot always block the process from ending, whatever process is using the critical region has to end).

Strict Alternation: A technique for avoiding race conditions, it gives each process a turn (eg.ABAB instead of AAAB). It may however get stuck if a process is stuck, such as B blocking A while it is in a non-critical region like waiting for input.

Test and Set Lock (TSL): Another technique for avoiding race conditions. Most CPUs come with a test and set lock instruction.

· RX: Register to read the lock value

· TSL: Tests the lock value, if LOCK = 0, then we can set it to 1 and continue with the code. If it is 1 then we cannot proceed and have to test again.

· LOCK: Value to check if it is locked or not

Busy Waiting: Technique for avoiding race conditions, when a process wants to enter a critical section it checks if the entry is allowed. If the entry isn't allowed then the process executes a loop which constantly checks if it is allowed to enter. The con of this is that it is a waste of CPU, and that high priority processes can interrupt low priority processes causing them to starve (and never be executed).

Blocking: Uses events to check if a process can enter a critical section, similar to sleeping in a queue. It attempts to enter a critical section, if it is available it will enter. If not, it registers interest in the critical section and blocks the process. When the critical section becomes available, the OS will unblock a process waiting for the critical section if it exists. Eg. Sleep() and Wakeup(). Blocking improves CPU utilization.

Deadlocks: A set of processes is deadlocked if each process in the set is waiting for an event that only another process in the set can cause. Eg. if process A locks 1, and process B locks 2. However process A needs 2 to continue and process B needs 1. As neither can unlock objects 1 and 2 without the other, they are in a deadlock.

Avoiding Deadlocks:

- Ignore the problem
- Detection (eg. graph algorithms) and recovery
- Avoid by careful resource allocation (requires you to know the resources beforehand)
- Prevention (All code in the critical region, may be inefficient)

1.4 Process Scheduling

Process Scheduler: The scheduler picks which process to run and for how long based on a scheduling algorithm.

It runs on:

- Process creation (Asks if it should run the new process or not)
- Process exit (Selectes the process to run next)
- Process blocks (When the current process is blocked, the CPU is not doing anything so the scheduler decides what to do next)
- Interrupt (Run process from before or the interrupt process)

Scheduler Input: Contains the processes in ready state (kept in the run queue)

CPU Bound: A type of process behaviour. The process uses a lot of CPU at a block of time, not on a constant basis (Long CPU burst, timewise).

I/O Bound: A type of process behaviour. The processes uses a bit of CPU at a time and then waits for I/O (eg. video games that require user input). Short time blocks of CPU.

Environments:

1. Batch: Schedules processes at once, eg. periodic analytic tasks
2. Interactive: Schedules processes as a user requests them eg. user facing
3. Real Time: Schedules processes as they are received eg. needing to meet a deadline

Scheduler Objectives: Note that some of these can conflict

- Fairness: All processes get a fair share of the CPU
- Throughput: Number of processes that complete per unit time, higher processes per time
- Turnaround Time: Time from process start to its completion, low completion time for processes
- Response Time: Time from when a request was first submitted until the first response is produced

Non-preemptive Algorithms: Process will run until it is finished or blocked. The CPU is given to the process and not taken away unless the proccess blocks itself.

Preemptive Algorithms: Process can be suspended after some time interval (indicated by a clock interrupt).

Process/Context Switch: Switching between processes, involves saving and reloading process states (variables, data, files etc). Only useful if the time that the process is running can be amortized, ie. if running the process continuously is longer than context switching between other processes.

First-come First-served Algorithm: Queues processes, if a process is blocked it goes to the end of the queue. It is simple and fair to all processes, however if a process was blocked and only need a short amount of time to finish then adding it to the end of the queue would make the turnaround time longer.

Shortest Job First Algorithm: Runs the processes in the order of the shortest job first. It has a short turnaround time however makes the assumption that the jobs arrive at the same time and that we know the length of each process. If jobs did not arrive at the same time, we could have a situation where the new ones are shorter than some processes in the queue. and the final process could be starved.

Shortest Remaining Time Next: Preemptive version of shortest job first. Similar to a queue sorted by shortest time, however if a process is unblocked and shorter than the rest of the queue, it will move to the front.

Round-Robin Scheduling: Has a list of runnable processes, and gives a bit of CPU time to each process, then switches to the next one. If the process hasn't finished then it is added to the end of the queue. We have to set the *quantum* which is the amount of time after it switches. This may be system dependent and has to be longer than the time context switching is going to take.

Priority Scheduling: Important jobs always runs before less important jobs. It also has a *quantum* and if the process is not finished, then it is downgraded to a lower priority level. It will clear all the processes in the highest priority before moving down into the next priority level. An issue that could arise is if we always have more important jobs arriving, then lower priority jobs will not execute.

Memory Management

Memory Manager: Abstraction of memory management, the size of a file should not be a concern to a programmer.

Responsibilities:

- Allocating memory to processes when they require it
- Deallocates memory when finished
- Protects memory against unauthorised access
- Simulates the appearance of a bigger main memory by automatically moving data between the main memory and disk
- Keeps track of which parts of memory are free and which parts are allocated to a process

Base Register: The beginning of a process's address space.

Limit Register: The upper limit of a process's address space. Address spaces should be within the base and limit registers. $Limit = Base + Memory\ Space$

Swapping: If a new process requires memory and the OS has no space for it, it will replace a process's memory space that is not currently required. Swaps those processes in and out. We should also note that splitting the memory is also not possible as we use base and limit registers.

Bitmap Memory Management: A data structure for checking if an address space is already occupied. For every address we can use `1` or `0` to denote if it is occupied or not. However search for an empty block of memory using a bitmap is very expensive and can be better optimised.

Linked List Memory Management: Another data structure used for storing the state of an address space. Instead of a bit map which stores every address, instead we can store blocks of memory in a linked list as addresses are consecutive.

Eg. $\boxed{\text{P}}\boxed{0}\boxed{5}\boxed{\text{next node}}$

where type = P (process), start address = 0, size = 5

Eg. $\boxed{\text{H}}\boxed{5}\boxed{3}\boxed{\text{next node}}$

where type = H (hole), start address = 5, size = 3

Search: We can jump around the linked list looking for holes with a specific size when we have a new process that requires memory.

Replacing a terminating process: Given a terminating process X, we can combine empty blocks

1. $\boxed{\text{A}}\boxed{\text{X}}\boxed{\text{B}}$ becomes $\boxed{\text{A}}\boxed{\text{Empty}}\boxed{\text{B}}$
2. $\boxed{\text{A}}\boxed{\text{X}}\boxed{\text{Empty}}$ becomes $\boxed{\text{A}}\boxed{\text{Empty}}$
3. $\boxed{\text{Empty}}\boxed{\text{X}}\boxed{\text{B}}$ becomes $\boxed{\text{Empty}}\boxed{\text{B}}$
4. $\boxed{\text{Empty}}\boxed{\text{X}}\boxed{\text{Empty}}$ becomes $\boxed{\text{Empty}}$

Memory Management Algorithms: An algorithm to determine which block of memory to choose.

· First fit: Finds the first hole that fits and allocates it. Pros: Easy to implement and search, Cons: May assign it to a hole much bigger than necessary.

· Best fit: Finds the closest sized hole that fits and allocates it. Cons: May end up with tiny memory fragments that are useless for other processes.

· Worst fit: Finds the biggest hole and allocates it. Pro/Con: Empty blocks of memory.

Fragmentation: An issue that arises when assigning memory is that memory fragements will appear due to memory blocks being split up. There may be enough empty space for a new process, however as it is split across fragments we cannot use it.

Virtual Memory: Allows programs to continue with the assumption that code and data does not have to be in the main memory when the program is running, and they do not have to be stored in contiguous locations. We assign each program its own address space, and break it up into chunks (same size) that are called pages.

Virtual Address Spaces: The set of addresses that programs on a machine may generate. Each process has its own virtual address space, and virtual address spaces may be bigger than the physical (CPU) address space.

Paging: A paged system divides both virtual and physical address spaces into fixed size pages. A virtual page can then be mapped to a physical page.

Page frame: Another name for a physical page, the job of a physical page is to hold a virtual page. Physical pages can be remapped to different virtual pages as required.

Memory Management Unit: Translates a virtual address to a physical one. CPU sends virtual addresses to the MMU and the MMU sends physical addresses to the memory.

Page Mapping: The mapping is generated by the MMU, and any actual memory access is done with the physical address value.

For example, the CPU requests the virtual address `8196`
The page memory is `4096` bits
To find the area of the page: $8196 \% 4096 = 4$
To find the physical memory location: $6 * 4096 = 24576$
Finding the physical address: $24576 + 4 = 24580$

Page Table Map: Maps a virtual address to a physical address. Dynamically updated, so we don't always know which pages are mapped to the physical space.

Paging Operations: The MMU must always check that the page table entry is present and permissions are allowed to access the memory. If it is, then it will construct the physical address otherwise returns a page fault.

Translation Lookaside Buffer: Stores frequently accessed pages into a data structure (if it is not in the buffer then we have to use the page table). The TLB directly stores the virtual page and the corresponding physical page. It is much quicker to access as we don't have to calculate the physical address.

Page Fault Handling: If the page is not supposed to be access then the handler will terminate the process. If a page fault arises because it doesn't have the memory to store a new page, then the OS needs to decide which page to evict from the physical memory and replace.

Local Page Replacement: Replaces a page that is already assigned to this process

Global Page Replacement: Replaces a page from other processes

Page Replacement Algorithms: Decide on the type (global or local), discards a page if not modified or writes to disk if the page was modified.

Not Recently Used Algorithm: Categorises pages based on the current values of their `R` and `M` bits. First to be removed is Class 0 and works it's way down.

- Class 0: Not referenced, not modified
- Class 1: Not referenced, modified
- Class 2: Referenced, not modified
- Class 3: Referenced, modified

Second Chance Algorithm: Pages are sorted in FIFO order. If the top page has been referenced in the past, then it is added to the end of the queue but it's `R` bit is set to `0`. If it arrives at the top of the queue again and isn't referenced, then it is removed.

Working Set: Pages currently being used by a process, and being read sequentially. If we constantly evict the next pages, then it can create frequent page faults.

Locality: Data access is not uniform throughout memory.

Temporal Locality: If it has been accessed in the past, and is likely to be accessed again.

Spatial Locality: Pages next to each other are likely to be accessed soon (eg. for loop)

2 Cryptography

Encryption: Hiding data from everyone except thoughs who hold the decryption key. Output is a cipher text.

Decryption: Recovering the orginal data from a cipher text using the key. Output is plaintext.

Symmetric Cryptography: Same key is used for encryption and decryption.
`CipherText := Encrypt(SecretKey, Message)`
`Message := Decrypt(SecretKey, CipherText)`

Asymmetric Cryptography: A private key is used for decryption, a public key is used for encryption. Slower than symmetic, and also cannot encrypt large amounts of data.
`CipherText := Encrypt(PublicKey, Message)`
`Message := Decrypt(PrivateKey, CipherText)`

Advanced Encryption Standard: Breaks data into blocks and encrypts each block.

Electronic Codebook: Simple, parallelisable process. Takes plaintext as inputs, uses block cipher encryption to output the cipher text. However, it shouldn't be used as it produces the same output for the same words, and can be understood through pattern comparisons.

Probabilistic Encryption: A more secure notion of encryption as we get a different output when running the encryption on the same message multiple times.

Cipher Block Chaining: Starts with a random Initialization Vector. Encryption is done sequentially, the first block takes the IV $\oplus$ plaintext as inputs and encrypts it using the block cipher encryption. It outputs a ciphertext, which is then used as an input to the second block and so on. Decryption can be done in parallel, and if a ciphertext block is lost, it only affects the current block and the next block.

Public Key Signature: `SignKey := Generate()` is private
`VerifKey := Generate()` is public
`s := Sign(SignKey,m)` the message is signed with the private SignKey
`Verify(VerifyKey, s, message)` is used to validate the message sender

Cryptographic Hashing: Should look random, and have collision resistance.

Key Sizes: RSA requires a minimum key size of 2048 bits while AES requires only 128 bits. Asymmetric is generally longer than Symmetric. Calculated based on mathematical probability of brute forcing a key.

Randomness: Cryptography is dependent on randomness, eg. IV for ciphers. We should use pseudo-random generators and they shouldn't be reused or discoverable. Generally, just use libraries as they are much better than human error.

3 Secure Communication

Adversary: An abstraction of issues that can happen in connections.

Confidentiality: A pillar of secure communication. We want to ensure that no one other than the intended recipient has read the message.

Authentication: Ensures that the right person has sent the message. Establishes the identities of one or both of theend points.

Integrity: Ensures that the data that was sent from one end point was not changed in transit. Eg. the message was not intercepted by an adversary and modified.

Cipher Block Chaining Tampering: If the attacker reorders ciphertext or flips bits then we cannot decrypt the next block.

Message Authentication Code (MAC): Detects if a message has been tampered with. Verifies the integrity of a message using a secret key. The adversary cannot create a new tag without the secret key, and therefore cannot change the message.

- s: MAC's secret key
- m: message
- t: tag, `t:=Mac(s,m);`
- b: verification (returns `0/1`), `b:=Verify(s,m,t);`

CBC-MAC: Based on encryption, secure for fixed variable messages. For variable length messages, we $\oplus$ the messages, so if it's too long/variable it won't work.

HMAC: Industry standard and widely used in practice. It generates a MAC tag t using a hash function and a combination of padded secret keys concatenated with the message.

```
t := Hash( (s⊕opad) || Hash((s⊕ipad) || m ))
```

where ipad and opad are fixed constants used for padding
`||` means concatenation

Authenticated Encryption: Generally, we encrypt then MAC. It's the only secure combination of these encryption processes.

```
c:=Encrypt(SK,m)
t:=Mac(s,c)
```

where s is a different secret key from SK

Diffie-Hellman Key Exchange: Allows both parties to agree on a shared key, then sends information in a way that both parties can calculate a shared key. Additionally, if a private key were to be exposed, previous sessions won't be as long as their secrets (s) have been discarded.

1. Generate a large prime p and a generator g
2. Alice picks a random value x and computes X $= g^x \bmod p$
3. Alice sends X to Bob
4. Bob picks a random value y and computes Y $= g^y \bmod p$
5. Bob sends Y to Alice
6. Alice calculates the secret key s $= Y^x \bmod p = g^{yx} \bmod p$
7. Bob calculates the secret key s $= X^y \bmod p = g^{xy} \bmod p$
8. $g^{yx} \bmod p = g^{xy} \bmod p$ so both Bob and Alice have the secret key

Human in the middle Attack: Network traffic between two endpoints goes via an adversary who is able to intercept their communication.

Impersonation: An adversary pretending to be an endpoint and sends and receives messages as said endpoint.

Eavesdropping: An adversary observing the messages between two endpoints. They intercept the messages and have knowledge of the contents, then sends it on to its intended endpoint without either party knowing.

3.1 Certificates

Certificates: Securely associates identities with cryptographic public keys. The certificate itself is signed by a third party. The most common standard format of a certificate is the X.509 which contains serial number, algorithm ID, etc.

```
certificate = (Issuer, $PK_{alice}$, Alice, ...  details)
signature = Sign($SK_{issuer}$, certificate)
verification = Verify($PK_{issuer}$, s, certificate)
```

Certificate Hierarchies: Certificates can be chainged, for example if A signs B's certificate, then B can sign C's certificate, implying that A trusts C. When signing certificates, they should also specify a limit for chains.

Example:
A signs B's certificate, B signs C's certificate:

A: $s_B = \text{sign}(SK_A, certificate_B)$, $certificate_B$ contains PK_B
B: $s_C = \text{sign}(SK_B, certificate_C)$, $certificate_C$ contains Charlie's domain address www.website.com

Alice wants to verify www.website.com using Charlie's certificate

Verify$(PK_B, s_C, certificate_C)$
Verify$(PK_A, s_B, certificate_B)$

Certificate Authorities (CA): Also known as root certificates, which are entities that are explicitly trusted. Generally they sign certificates for others, or have sub CA that signs on their behalf. Root certificates are also self signed, so we know when to stop chain verification.

Certificate Issuance: Generally has to validates that the PK and domain address belong to the same individual.

- Domain Validation (DV): Most common form of certificate issuance. The CA ties a certificate to a domain and checks that the requester has some control over the domain. Generally done by 'challenging' the domain, such as sign a random string and put a random string on your website.
- Organisation Validation (OV): Ties a certificate to a domain and legal entity (eg. a business)
- Extended Validation (EV): Establishes legal entity, jurisdiction and presence of an authorised officer (by address, phone etc). Involves and offline process and can be expensive.

Certification Revocation: Occurs when a certificate is mistakenly issued, a private key is compromised or certificates are expired. Uses a Certification Revocation List and OCSP (Online Certificate Status Protocol). The list is a giant list that anyone can use to search (questionable performance), whereas the OCSP is similar to asking a CA if a certificate is revoked (not as private, and requires CA to provide a response).

Certificate Transparency: An open framework used for monitoring and auditing certificates. Detects misissued certifcates, maliciously acquired certificates and rogue CAs. Uses a cryptographic append-only log to record the issuance of certificates. If it exists, then the server sends a **signed certificate timestamp**(SCT).

3.2 SSL/TLS

Transport Layer Security protocol: A protocol for secure communication over the internet. Previously known as SSL (Secure Socket Layer).

Handshake Protocol: Initial establishment of connection between the client and the server. Asks which cryptographic algorithms they support, their version of TLS (in case of downgrade), and creates a shared secret key (using Diffie - Hellman). Optionally, they can also authenticate each other's identities using digital certificates.

Record Protocol: Uses the secret keys created in the handshake protocol to protect confidentiality, integrity and authenticity of data exchange between the client and the server.

Downgrade attack: When the handshake protocol requires a downgrade to an older version of TLS for client network communication. They will then target known vulnerabilities in previous versions which makes it unsecure.

Client - Server Interaction:

Client

- Sends protocol version
- Sends cryptographic algorithms
- Sends a random nonce (allows them to establish a new communication channel and avoids the reuse of old interactions)

Server

- Sends the highest protocol version supported by both parties
- Chooses the strongest crytographic suite selected by those offered by the client
- Server also sends its public key certificate, or its Diffie-Hellman public key (depending on the suite selected). The Client can then use this to validate the certificate.

Key Exchange

- The client can then generate a secret key and send it to the server encrypted with the public key (asymmetric/symmetric), or g^x if using DF.
- The key is useful for a session.

Local TLS Interception: Sometimes a local entity may need to intercept communication. An example is anti-virus software, which acts as a human in the middle (but non malicious). It can request to make changes to your computer, and install it's own certificate on your machine, appending itself as a root CA. It then has the ability to intercept all communication from a browser to the user, and make sure all messages are safe. An issue that can arise is if multiple machines have the same P_k issued by the software, in which case other people can impersonate them.

Server-side issues: Proxies are used for privacy. The break the concept of end-to-end encryption. It'll analyse traffic, know which ips to block and filter, then forwards it to a server. An issue is that it has a lot of information from different sites, so if it exposes it, there will be a lot of potential breaches.