Revision Notes

Kevalee Shah

March 26, 2019

Contents

1 Basics

1.1 Text

ASCII used to be the most popular, 7-bits, enough for the alphabet, numbers and some punctuation.
Unicode has 8, 12 or 32 bit code which then is able to code for all international languages.
UTF-8 is commonly used - a superset of ASCII and therefore allows for backwards compatability. A variable width character encoding using one to four bytes. It is enough for most Roman languages. Code points with lower numer-

ical values which tend to occur more frequently are encoded using fewer bytes.
Need to be careful with endianess -

- Big Endian - when the most significant bit is first
- Little Endian - when the most significant bit is last
- To overcome this issue often have machine independent representation of data, e.g. external data representation (XDR)

1.2 Numbers

Integers

Unsigned numbers are represented in binary or more commonly hex. $n-$bit register can store numbers from $0 - 2^{n-1}$
Hex - group bit in 4s
For signed numbers in order to prevent a negative zero, 2's complement is used:

- if begins with 0 read like normal
- if begins with 1 invert all bits, add one and then read like normal, but know its a negative number
- represents numbers $-2^{n-1} - (2^{n-1} - 1)$

Floating Points

There are three parts to the floating point representation:
Assume $32-$ bit representation

- Sign - first bit
- Exponent - next 8 bits show the exponent. 127 is the bias, therefore read your number and if greater than 127, subtract it. E.g. if your exponent part of the number was 131 then your actual exponent would be 4
- Mantissa - the remaining 23 bits represent the mantissa. This is equal to 1 + the fractional part shown by the bits.
  e.g. $1010 = 1 + 1 * (\frac{1}{2}) + 0 * (\frac{1}{4}) + 1 * (\frac{1}{8}) + 0 * (\frac{1}{16}) = 1.625$

1.3 Instructions

An instruction is comprised of: `opcode --- operands --- instruction`
Opcodes specify what the instruction is and operands are like parameters of the opcode - the values needed for the instruction to be carried out.
The opcode is like the name of the function. First fetch and decode the opcode, and then if necessary fetch the operands, which are like the arguments of the function.
There are different ways of reading instructions:

- Addressing mode - either part of operand or given explicitly
- Variable length encoding - better density
- Huffman coding - most probable instructions given shorter opcodes

1.4 Computer Overview

CPU executes programs. Memory stores programs and data. Buses transfer information. Devices for input and output. Computer takes values from memory, performs operations, and then stores back in memory. CPU operates on registers (on-chip memory)

1.5 Fetch-Execute Cylce

- `Fetch` - CPU gets instruction from the memory address which is stored in the program counter and is stored into the insturction register. At the end of this phase, it points to the next instruction for then next cycle.
- `Decode` - CPU determines which system components are needed for execution of the instruction stored in the instruction register. Generates control signals and operand info
- `Execute` - Control unit passes the decoded information to the functional units of the CPU, such as the Arithmetic Logic Unit, Branch Unit, Memory Access Unit etc. E.g. ALU performs mathematical/logical functions on values from the operands.
- `Store` - New processes data is written back into main memory (or sent to an output device)

1.6 SSD vs HDD

SSD

- Solid State Drive, very fast as there are no moving parts (no mechanical arm), non-volatile and therefore retains the information.
Controller within SSD performs operations to read and write data into the drive.

HDD

- Hard Disk Drive, slower than SSD as it stores data on rotating disks and there is a read/write header that then is able to access/modify the data. As it requires a lot more physical work it is slower, but also much cheaper.

1.7 Graphics Card

CPU's lack the specialised hardware required to convery binary data into visible pixels. Graphics card have some RAM and a processor and are specialised to do this job. The processor is able to handle the computational requirements for displaying graphics. The RAM is needed to hold the colour of each pixel. It is also connected to the mother board so that is can send signals to through the bus to alter the memory of the graphics card.

1.8 Buses

Shared communication wires:

- address lines - where data should be carried

- data lines - data to be carried
- control lines - what operation to carry on data

Advantages are that low cost, versatile, don't need wires everywhere, good for modularity

1.9 DMA

DMA stands for Direct Memory Access. It is a device that is able to directly read and write into main memory. Once the device has requested memory access it only talks to the DMA controller and bypasses the CPU entirely. The only other interrupt is when the transfer has been complete and an interrupt is generated by the DMA controller. This is a lot faster as there are much fewer interrupts (e.g. DMA does not need to ask the CPU about things like requesting more space) and therefore the CPU can carry on with other tasks. Issues arise when the DMA controller never raises interrupt saying the DMA is done trasnferring. Then the DMA can just carry on dumping data into more memory than the machine can handle.

1.10 Operating Systems

Needs to protect applications while still allowing them to share physical resources

- share CPU
- provide each application with virtual processor
- divide storage space using filing systems
- present a set of hardware independent virtual devices

2 Protection

2.1 Why?

- Unauthorised release of info - privacy legislation
- Unauthorised modification of info - access rights
- Denial of service - crashes, high load
- Effect of erros - debugging

2.2 Memory Protection

We cannot just make IO instructions priviledges by ensuring that applications cannot mask interrupts and applications cant control IO devices as it still allows some devices to be accessed via memory. Therefore it is important to protect memory as well.

- define a base and a limit for each program and protect access outside range

- every memory address is checked by the hardware for
- access out of range causes interrupt
- only allowed to change base and limit registers when in kernel mode
- only allowed to disable memory protection in kernel mode

2.3 CPU Protection

Prevent any process from hogging the CPU for too long by having a countdown and every tick of the clock causes the timer to decrement until an interrupt is issued when the timer is 0.
Ensure the OS runs periodically and remains in control

2.4 Dual Mode Operation

`User mode` - When an user application is being run the system is in user mode. When the application requests something from the OS or if there is an interrupt/system call then the mode switches to kernel mode
`Kernel mode` - Hardware starts in kernel mode. Priviledges instructions can only execute in kernel mode - handling interrupts, switching from user mode to kernel mode, IO management

System calls are made from the user mode to ask to switch to kernel mode.

2.5 Microkernel

User services and kernel servies are implemented in different address spaces - reduces the size of the kernel and the size of the OS. It only provides the minimal service of process and memory management and therefore relies more on communication between applications and services running in the user address space. This reduces the speed of execution. However advantages are that if one user service fails it doesnt cause the kernel service to fail as well, as they are isolated. It is also easily extendable as new servers can be added to new user address spaces and doesn't require kernel space modification. A disadvantage is that need to worry about synchronisation.
Most important services:

- Inter-process communication (IPC) - this does add a lot of overhead
- Memory management - often a lot of memory waste as you end up with redundant copies of OS data structures
- CPU Scheduling

2.6 Virtual Machine

Program or OS that exhibits the behavious of a separate computer and is abel to run application and programs like a separate computer. It is created within another computing environment and is called a host. The VM is a guest. They are used to perform certain taskts that are different to those performed in the host environment.

- `System VM` - system platform that supports the sharing of the host's physical resources between the many VMs which have their own OS
- `Process VM` - provides a platform-independent programming environment that hides the underlying hardware/ OS and allows the program to execute in the same way as on any other platform

Another form of protection in order to authenticate user to system is by using passwords

2.7 Access Matrix

Matrix of subjects vs objects. This is too large and sparse, therefore two common ways of representing:

- Access Control Lists (ACLs) - list of objects with subjects and rights (e.g. files are rows and users are columns), it is checked when a file is opened to be read, written or executed.

  Preffered for object centric operations - adding or removing objects which tend to be more common as less modification is required compared to capability lists

- Capabilities - list of subjects with objects and rights (e.g. users are rows and files are columns)

  have special hardware instructions to restrict capabilities

  good for distributed systems

  good for user centric operations - adding/removing users

3 Processes

Processes are NOT programs; programs are static and on-disk, whereas processes are dynamic - processes are programs in execution.
Each process is executed on a virtual processor so that each process feels like its got its own machine.
Each thread of a process has:

- Program Counter - which instruction is running
- Stack - for temp variables, parameters etc.
- Data Section - global variables shared between threads

3.1 Process Lifecycle

3.1.1 Creation

Most systems are hierarchical - parent processes create child processes
Parent processes may share all, some, no resources with child process.
Parent and child processes execute concurrently or parent process waits till child process has terminated.

Address space either shared with child process, or child gets its own address space.

3.1.2 Termination

Three ways a process terminates:

- Executes last statement and asks OS to delete it - deallocation of resources
- Process executes an illegal operation
- Parent terminates the child process

3.2 Types of Processes

- `IO bound` - spends more time doing IO than computation, many short CPU bursts
- `CPU bound` - more time spent doing computation, few long CPU bursts

3.3 Process Management

3.3.1 Process Control Block

Data structure that contains information about the process related to it including:

- process state
- process number
- program counter
- registers
- memory limits

The process context is the program counter and the registers.

3.3.2 Context Switch

The OS must save the current context and switch it for the context of the process being resumed. This time is wasted time as nothing useful is actually happening. The time it takes depends on hardware support.

3.3.3 Threads vs Processes

Threads of the same process run in a shared memory space while processes run in different memory spaces. Threads have a TCB (thread control block) with metadata associated with it. During context switches between two threads process state does not need to change.

3.4 Communication

3.4.1 Inter-Process Communication

Importatn to remember `syntax`, `semantics` and `sychronisation`
OS is involved in the communication between processes, otherwise it might violate the protection rules
Process often need to share memory

3.4.2 Inter-Thread Communication

Two running threads would need to communicate - e.g. coordiante access to shared variable
can lead to concurrency issues if not done properly

3.4.3 Inter-host Communcation

Transfer of data between two hosts; physical and nowadays virtual.
No shared memory and therefore other transmission medium must be used - issues with lost data and asynchronousity

3.4.4 Methods of Communication

- Fork and wait - allows process to clone itself - parent then waits for child to termianate or separates from it entirely
- Signals - notification to a process indicating the occurrence of an event, also known as a software interrupt, cannot predict when it happens and therefore asynchronous. Signals are specified with a name/number. Three things can happen:
  - Default action
  - Handle the signal
  - Ignore the signal
- Pipes - between threads or between separate processes like parent-child or grandparent-child. One process writes into the pipe and the other process reads out of the pipe. Output of one is input for the other. To get pipe system call - create two files - one to write into and one to read from.
- Shared memory segments - Get a segment of memory, attach processes onto it, impose concurrency control to ensure no collisions, once processes have been executed then detach and destroy when you know no other process is using it.
- Sockets - provide point to point, two way communication between processes. Different types of sockets - only communicate between sockets of the same type. Allows a flow of data
- Files - using files is a type of shared memory - instead of allocating a common memory buffer in RAM, a common file is used. Not really great method for performance, but it is a way.

4 Scheduling

4.1 Queues

Three Queues:

- Job Queue - batch processes awaiting admission
- Ready Queue - prcoesses in main memory waiting to be executed
- Wait Queue - processes waiting for IO

The Job scheduler decides which process to put onto the ready queue. The CPU scheduler selects the next process to be executed.

4.2 Types of Scheduling

There are different factors that come into play to decide what sort of scheduling algorithm you want:

- Maximise CPU utilisation - used to be important when CPUs were expensive
- Throughput - maximise rate of process completion; but may penalise longer running processes
- Turnaround time - minimise time taken to complete a process
- Waiting time - minismise time spent in the waiting queue
- Response time - minimise time for first response from when the request was made

4.2.1 Non-preemptive

Used when a process terminates or switches to a waiting state. A process holds onto the CPU until it no longer needs it. There is no interruption in the middle of a process. No overhead from switching the process from running to ready. It is rigid scheduling. Very simple to implement, but open to denial of service. Scheduling algorithms include:

- First Come First Serve - easy but high average wait time
- Shortest Job First - processor has to know in advance how much time each process will take (not possible in interactive systems), but minimises waiting time
- Priority - each process assigned priority and highest priority executed first

4.2.2 Preemptive

When a process switches from running to ready or waiting to ready. CPU is allowed to the process for a limited time. Interrupts in the middle of execution and context switch occurs. There is an overhead for this - much harder to implement, need timer etc. However solves denial of service. Scheduling algorithms include:

- Shortest Time Remaining First - preemptive version of SJF. preempt running process if the new process will have CPU burst length less than remaining time of current process, but again need to know burst time from before

  In order to predict the burst lenght we can use exponential averaging:

  $$\tau_{n+1} = \alpha t_n + (1 - \alpha)\tau_n$$

  predicted value for next CPU burst = alpha * actual length of previous CPU burst + (1 - alpha) * predicted time for previous CPU burst

- Round robin - each process gets (up to) a fixed quantum of time - not as good turnaround time but better average response time, balance between too long quanta that theres no difference from FCFS and time taken for context switches

- Dynamic Priority Scheduling - preemptive version as this allows the priorities to change over time and therefore there aren't some low priority processing waiting a long time

4.3 Idling

When there are no processes to run, the CPU is idle.

- Constantly check if something is ready to be run - very quick react time but just a bad idea

- Halt processor until something to run - saves power, increases processor lifetime, but very slow response time

- Have idle processes that have the lowest priority, and therefore are only run if there is nothing else, processes which cna always be run but not important. e.g. maintaining system, checking disk. It does take up some memory and increases the number of context switches, but fairly good response time.

5 Virtual Addressing and Virtual Memory

5.1 Overview

`Virtual Memory` - memory management by the OS which compensates for physical memory shortages by transferring data from RAM to disk storage. A section of the hard drive is used to emulate RAM. This allows many programs to run at the same time as each thinks that it has infinite memory. Benefits include portability, convenience and efficiency.
`Virtual Addresses` - A binary number in virtual memory that enables a process to use a location in main memory independently of other processes and use more space than actually exists in main memory. It does not represent the actual physical location of an object in memory, but there is a data structure in the OS that translates virtual addresses to real addresses.

5.2 Issues with Direct Access of Physical Memory

- Contiguous Allocation - need a large lump, leads to external fragmentation
- Address Binding - see below
- Portability - standard machine has limited memory

5.3 Resolving Address Binding Problem

- Compile Time - Hard code addresses and always load executables so that the addresses match up e.g. DOS, .COM
- Load Time - Position in memory found after the program has been loaded and code is updated with the new addresses. Needs to be done every time the program has been loaded. Need to consider swapping time and binding time. Loading is from hardrive and therefore slow. Extra hardware required for this.
- Run Time - Hardware automatically translates between program and real address. This requires MMU, but is very flexible, no change to the code required

Virtual and physical addresses are the same in compile-time and load-time address-binding schemes, but different for execute-time address-binding schemes.

5.4 Memory Management Techniques

5.4.1 Contiguous Allocation

Small part of the memory reserved for the OS. The rest is available to a single application. For multiple proceesses need to swap contents or need to swap users.
e.g. MS-DOS

5.4.2 Partitioned Allocation

Divide primary memory in multiple contiguous memory partitions which contain all the information needed for a specific job or task. Needs hardware to stop them interfering with each other. Needs to unallocate parition when the process ends.

- Static Partitions - defined at boot time or by the operator
- Dynamic Partitions - automatically created for a specific job. e.g. for a new process OS searches for larger enough memory slot:

  first fit - first slot that is big enough

  best fit - search through everything to find bet fitting slot

  worst fit - find biggest hole

Dynamic partitioning leads to external fragmentation - when there is enough memory for another process, but it is unusable as it has all been split into smaller slots

5.4.3 Paging

This allows a process to exist non contiguously. Physical memory is divided into frames. Logical memory is divided into pages. Where size of page is bigger than size of frame.
Logical address comprises of a page number and a page offset.
Each process has a page table which have page table entries (PTEs) - great overhead and adds to the information that needs to be recorded in the PCB
Solves external fragmentation but can lead to internal fragmentation where a process might not use up the whole page.
OS has to keep track of free memory in a frame table.
There is an issue with protection - introduce protection bits with each page, stored in the page table (3 bits for read, write and execute)
Data sharing is done easily with paging as you can have two logical addresses mapping to the same physical address

Demand Paging To load a new process to be executed:

- Create address space - page tables
- Mark PTEs are invalid or non-resident
- Add PCB to scheduler

If a page fault is received, check PTE. If it is an invalide reference then kill the process. Otherwise:

- If there is a free frame use it
- Else, select vicitm page
- Write victim page into disk
- Mark it as invalid in page table
- Page in desired page into the free frame
- Restart faulting process

Page Replacement It is important to choose victim page wisely - minimise page fault rate.

- FIFO - keep a queue of pages and discard from the head. Issues as we dont know if discarded page will need to be used again soon, or if it is actually currently in use.
- Optimal Algorithm - replace the page which will not be used for the longest period of time; can't be done without hindsight therefore just a baseline for other algorithms
- Least Recently Used - replace page which has not been used for the longest time. Hardware required, need to determine ordering, still can replace pages that are to be used next

`counter` - whenever page is referenced PTE updated to clock value. Need to then doa search to find the minimum counter value, every memory reference requires a write to memory, clock overflow etc. therefore impractical

`stack` - maintain stack of pages with most recently used on the top, and discard from the bottom. However this can required changing many pointers per new reference, very slow without extensive hardware. therefore impractical

`clock algorithm` - implementation of approximating LRU. Arrange physical pages in a circle. Each page has a use bit set to 1. If it is 0, the page has not been used in some time. When there is a page fault, clock hand starts to sweep. If a bit is set to 1, it sets it to 0. If it encounters a 0, that page is the victim page.

`Least Frequently Used` - keep count of number of references on each page and replace page with smallest count

`Most Frequently Used` - keep count of number of references on each page and replace page with highest count

Frame Allocation

- Global - all pages considered for replacement; process cannot control its own page fault rate; performance depends on other processes
- Local - victim page chosen from process' own pages. performance only depends on process behaviour, but can hinder progress as not utilising other free pages.

Thrashing Occurs when memory has become exhausted and too limited to perform needed operations. Therefore when a request is made the OS tried to find free pages but as there are none, it swaps with a page from another process. When that process issues a request is has to swap with another process. And therefore overall very little progress is made.

Working Sets Collection of pages the process is actively using must be kept in memory in order to prevent the process from thrashing. If the sum of the working sets for each running thread is greater than memory, stop running some threads for a while. When a process becomes inactive, move its working set onto disk. There is a `balance set` which is the collection of active processes. As some processes move out of the balance set, need to move other inactive ones into the balance set.

Computing Working Sets Define a window size of Δ of most recent page reference. If page is in use then it is in the working set. All pages reference in the last T(working set parameter) seconds are in the working set. This gives an approximate to the locality of the program. Can get the working set of each process and then can determine if some need to be suspended for a while.

When a process is resume, we can remember its working set and pre page the frames into memory.

Page Size Larger page size means there will be fewer fault, however can also lead to internal fragmentation where the process doesn't require all the space given to it. Values are usually between $512B$ to $16kB$

5.4.4 Segmentation

Overview Memory is devided into variable sized segments. Details of the segment are stored in `Segment Table`, such as base address and limit. The logical address produced by the CPU consists of a segment number and an offset. Segment number mapped to segment table. If `offset` $<$ `limit` then valid address. It is then translated into the physical address - `base` + `offset`. Segment table of current process is kept in main memory, and it is referenced by the Segment Table Base Register, and the Segment Table Length Register.

Advantages

- No internal fragmentation
- Protection between components - protection provided per segment, protection bits associated with ST
- Sharing - enabled when two processes have entries for the same physical location

Disadvantages

- External Fragmentation
- Need costly algorithms to fix that - costly

  Best/first fit algos

  Compaction - free memory collected in big chunk, but CPU heavy
- Hard to allocate a lot of contiguous memory

5.4.5 Segmentation vs Paging

`segmentation` - better for protection and sharing
`paging` - better for allocation and demand access
Some OS such as Multics use a combination - such a paged segments where each segment is split into pages. However lots of hardware required and not very portable
Other OS (most of them) have software segments where a group of pages is considered to be a segment. Simple and portable but loss of granularity.

5.5 Dynamic Linking and Dynamic Loading

`Dynamic Linking` - During execution the system library is loaded into main memory and the function call inside program is linked to function definition inside library

`Dynamic Loading` - Load the main module of a program first and then during execution load other modules for the process when it is needed

6 IO Subsystem

There are 4 types of IO Devices: `block` (disk drivers), `character` (keyboard), `network`, and `miscellaneous` (timers)

6.1 Polling

Polling is when the CPU periodically checks if IO devices need the CPU. Every device has a `command-ready` bit which indicated the status of the device. The CPU has a `busy` bit which indicates whether the CPU is executing a process or not. Algorithm:

1. IO Device checks CPU busy bit and waits for it to be clear
2. It writes into its data-out register and set the command-ready bit to 1
3. CPU sees that the command-ready bit is 1 and sets its busy bit to 1
4. CPU reads command register and executes the command
5. CPU sets its busy bit, the device's command-ready and error bit to 0

6.2 Interrupt Driven

Interrupts are hardware mechanisms that enable a CPU to see that a device needs it. At the end of each instruction the processor just needs to check interrupt-request line. CPU stops the currently executing task and responds to the interrupt by saving the program counter, changes processor mode, and gives control to interrupt handler. Once the interrupt handler has completed its job its hands control back to the CPU to carry on with the process it was executing.

6.3 IO System Calls

6.3.1 Blocking

Blocking IO is when a process is suspended until there is some data to read, or the data is fully written. Until the operation is complete the thread must wait. Blocking IO is inefficient as often IO operations can be very slow and therefore totally stopping the process means many resources remain idle for a long time.

6.3.2 Non-Blocking

In non-blocking IO the system call returns as much as available immediately, which means there isn't a long wait time, but often not all data is retreived.

6.3.3 Asynchronous

Asynchronous is a form of IO processing that allows other process to carry on executing while the transmission is being carried out. The process which doesn't require the operation to be completed carries on being run, and just the specific parts that need the IO to have completed is left waiting. This is a lot more efficient, but is much more complex to implement.

6.4 Buffering

A `buffer` is when a set amount of data is stored to preload the data before it goes to the CPU. It acts as a temporary placeholder for data coming in from all devices that operate at different speeds. It allows specific programs and devices to run more efficiently.

Single Buffer .

Double Buffer .

Circular Buffer .

6.5 Other Issues

There are other IO related issues to think about:

- Caching - diverts IO to cache in order to reduce IO operations to external storage
- Scheduling - reorders the IO requests coming in to make the search for the data on disk to be as fast as possible/ to prioritise certains IO requests
- Spooling - queue output for device; buffer that holds output e.g. for a printer

6.6 Performance

IO is a huge factor for overall performance due to context switches due to interrupts, data copying (from the hard drive), CPU having to execute device drivers etc.
In order to improve performance:

- Reduce number of context switches
- Reduce amount of data copying

- Reduce number of interrupts

  large transfers

  smart controller

  Polling

  DMA

7 Filing

7.1 Files

Files are an abstraction for non-volatile storage. They are of different types such as data, program or documents and can have a range of complex inner structures.

Names Files usually tend to have two names: `System File Identifier` - integer value associated with file used within file system, `Human name` - name given by user. Mapping from SFID to human is recorded in the directory.

File Control Block - SFID maps to its file control block. This is where the file metadata is stored. This includes:

- Pointer to file location on device
- File size
- Type
- Protection - who has read/write/execute access
- Timestamps

7.2 Directories

These provide the means for a user name to be translated to the location of the file on-disk. They need to:

- Efficient - need to locate quickly
- Naming - need to allow users to have the same named files, to have one file have different names
- Grouping - need to be able to group by properties

Tree structure is used to represent directories. Allows for efficient grouping and searching. The path to the directory is the human name which is a bit long and therefore a slight issue that can be resolved with relative naming/current working directory. It is a DAG.

Directories are non-volatile and therefore are also a type of file and have their own SFID.

7.3 File Access

Sequential Access - information in the file id processes one record after another. `read next` operation reads the next position of the file and advances file pointer. `write next` operation allocates memory and moves pointer to the end of the file. Method okay for tape.

Direct Access - allows program to read and write record fast. There is no restriction in the order of reading and writing the file

Index Sequential - Index has pointers to various blocks and therefore we look at the index and then with the pointer access the file directly.

8 Definitions

- `layering` - to manage complexity by controlling interactions between components - e.g. arrange in a stack and can only access from one below it.
- `synchronous` - shared clock
- `asynchronous` - no shared clock
- `latency` - how long something takes
- `bandwidth` - the rate at which something occurs
- `jitter` - variation in latency
- `caching` - fastens the access speed of repeatedly used data by storing a copy of the original data
- `buffering` - matches the speed between the sender and recevier of the data stream. It is an area in the RAM that stores the data temporarily when it is being transferred between two devices
- `bottleneck` - the one resource that is the most constrained in a system
- `Operating System` - program controlling the execution of all other programs
- `covert channel` - type of attack that creates a capability to transfer information objects between processes that are not supposed to be allowed to communicate; information leakage by side-effects
- `mode bit` - a bit added to the hardware of the computer to indicate the current mode
- `Mandatory Access Control (MAC)` - type of access control where the OS constrains the ability of a subject to access or perform an operation on the object
- `Subject` - also known as `principal`. Users/executing processes
- `Objects` - things like files, devices, domains, message ports, sockets etc.

- `Address Binding Problem` - The issues of not knowing where in memory the program will be loaded and therefore not knowing what addresses to assign to the variables pointing to values
- `MMU` - memory management unit; Performs the translation between logical addresses and physical addresses.
- `Transaltional Lookaside Buffer` (TLB) - stores a cache of recently used mappings from the page table
- `Locality of reference` - tendency of a processor to access the same set of memory locations repeatedly over a short period of time

9 Diagrams

- Fetch-Exectue Cycle
- Process lifecycle
- Page table
- Multilevel Page table
- Clock algorithm