Lesson 1

Os: Exploits the hardware resources of a computer system and provides services to the user.
- ↳ Error detection
- ↳ Communication
- ↳ Program execution

Von Neumann:

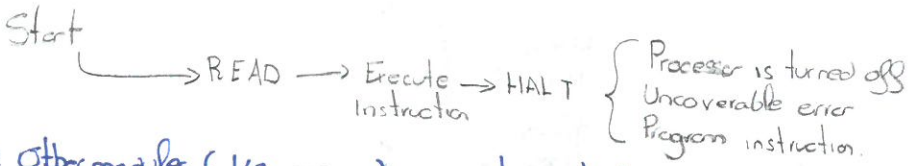
CPU (Central Processing Unit): Has control and operation functions
I/O modules: Moves data inside and outside
Main memory: Stores data and programs. Volatile RAM + ROM

PSW: Program Status Word → Stores information of the result of arithmetic operations
PC: Program Counter → Contains the address of the instruction is being executed
IR: Instruction Register → Holds the instruction executed

User // Kernel
Nonprivilege instruc. Privilege instruc.

Program: Set of instructions stored in memory
Not stored → not executable

BIC: Basic Instruction Cycle



Interrupts: Other modules (I/O, memory), may interrupt the normal sequence of processor → modify de BIC

6 steps { 1-4 → Hardware
5-6 → Software

* There's a table with interrupts and address *

4 types of interrupts:

Synchronous:
- Program: Problem that occurs, such as 0/0, overflow

Asynchronous:
- Timer:
- I/O:
- Hardware failure: power failure

Memory hierarchy:

Register
Cache Memory
Main memory
} Inboard memory

Magnetic/optical Disk } Outboard Storage

Magnetical tape } off-line Storage

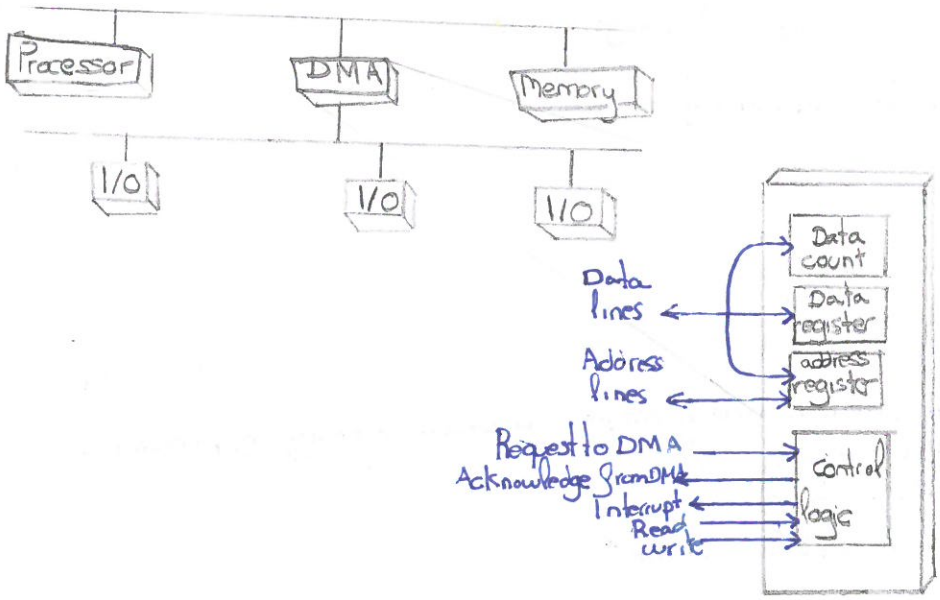
Input/Output. Direct Acess Memory

Programmed I/O: Checks the I/O status (Active waiting) and transfers data

Interrupt driven I/O: Sends a command to the I/O module to prepare the I/O operation. When ready, interrupts the processor (passive waiting)

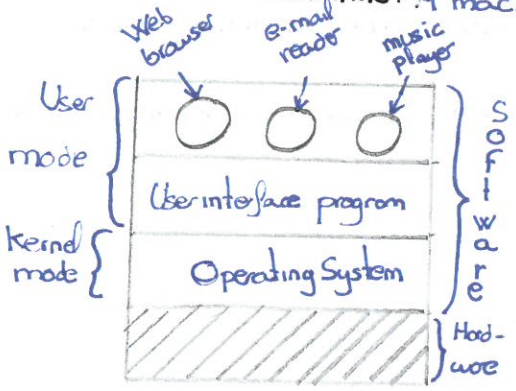
* Direct Memory Acess: Able to move data from-to memory directly. DMA interrupts when the whole operation has been completed

*DMA Scheme



Lesson 2

Naked machine: A machine without operating system



Design principles of an Operating System: Three objective

Convenience: An OS is the bridge between the software and the hardware for the user, which one doesn't know the details of the hardware. The OS provides a set of tools as "Application Programming Interface called API, and other libraries or utilities.

- Shell: Interface for access to an operating system's services.
- Kernel: Central part of an OS. Manages operations and hardware

Services provided by the OS:

1- Program development: editors, debuggers...
2- Program execution: Set of steps such as load instruc. and data into the main memory. (If the program is not into the main memory, it cannot be executed), I/O devices prepared...
3- Acess to I/O devices
4- Controlled access to files
5- System access, providing protection to resources
6- Error detection and response
7- Process communication and synchronization
8- Resources assignment

All those services could be group into three categories. 1-Services for running programs. 2- Services for the execution of commands 3. Services for managing computer resources

Efficiency: Use of resources in an efficient manner. A computer is a set of resources for the movement, storage and processing of data and control. The resources are the CPU, the I/O and main memory This includes the kernel, with the main functions in the OS. Scheduling them... The OS has a set of structures such as I/O devies assigned, portion of main memory assign...

Ability to evolve: The OS should be able to implement new system functions without interfering the service. There are three main reasons for the OS evolve.

1- Hardware upgrades, + new types of hardware
2- New services, to improve performance
3- Fixes to correct the faults of an OS. Which may introduce new faults

Evolution of OS I

Serial processing (1940s - mid 1950s)

No direct interaction between programmer and the OS. It happens thanks to the Naked machine. A console with displaying lights, toggle switches, a printer... Cards introduced in machine. If it was correct it would print, if not, with lights the problem would be indicated. There were two problems.

- Scheduling: There was a hardcopy sign-up sheet to reserve time in slots for 30 m. This made problems when they used 15 slots
- Setup time: Lot of time spent to set up the program to run. If something goes wrong it comes to the start

In general there was a huge waste of time

Simple batch systems (mid 1950s - early 1960)

Software called "monitor": The user lost the access to the processor. The user submits the job on cards to a computer operator, who batches the job sequencially. It gets back to the monitor when it completes processing, and then the next program starts

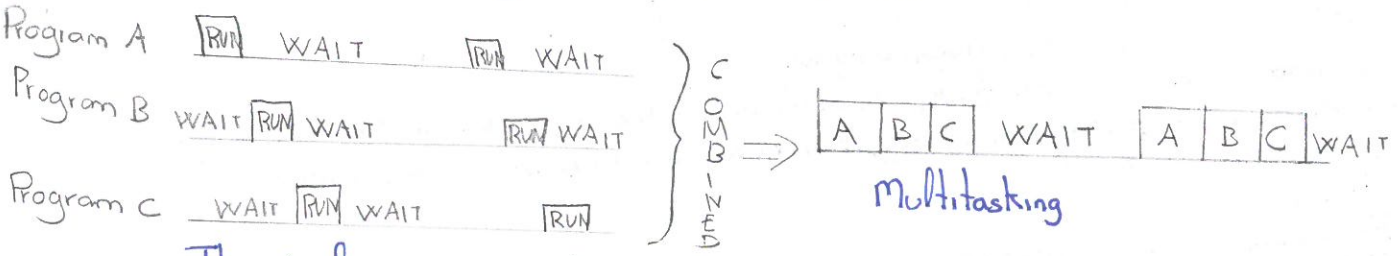
The main memory of the monitor is called "resident monitor": Some parts are introduced such as:

- Timer, to avoid that one job monopolize the system
- Memory protection, user can access memory
- Privileged instructions for the monitor
- Interrupts (user mode or kernel mode)

Evolution of OS II

Multiprogrammed batch systems (1965-1980)

More process in to memory since it has the capacity



They developed a more efficient memory management to allocate more than one process, a Direct Memory access to make the processor independent from I/O devices and the scheduling

Time sharing systems (1965-1980)

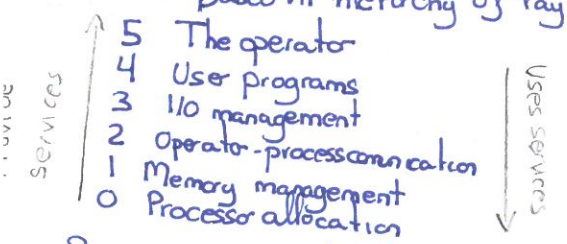
Similar to multiprogrammed systems with some sleigth differences:

- Multiprogrammed systems but for interactive applications
- Time is shared among the users and it gets adapted for this kind of interactive applications

Operating system Structure

- **Monolithic systems**: The entire OS is ran as a single program in kernel mode as a collection procedures that are conected between them. Very efficient but it is a problem when there are thousands of procedures and one small problem could get the whole OS to shutdown
- **Layered systems**.

Based in hierarchy of layers



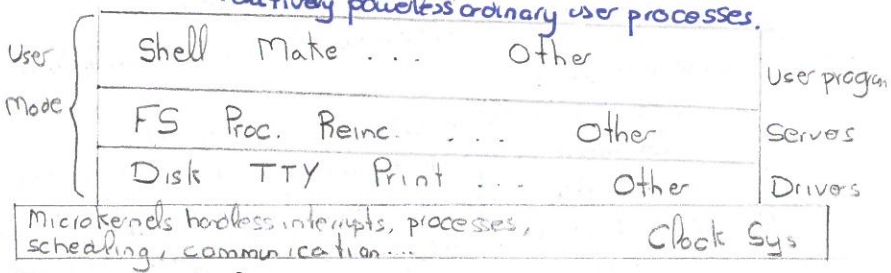
Provides services to the layer above, uses services from below. Where to put kernel line?

- **Client-server model**

Two classes of processes, the severs and the uses. One gives and the other uses services. The communication is made by messages. The user sends the requirement and the server does the work and responses with the answer. This mechanism could be implemented in a single machine. Same idea as a website

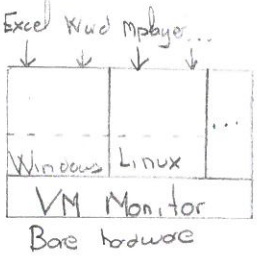
- **Microkernels**

High reliability by splitting the operat. system up into small, well-defined modules, only which runs in kernel mode and the rest run as relatively powerless ordinary user processes.



- **Virtual machines**

A copy of the bare hardware, including the kernel/user mode, I/O, interrupts...



It can provides several VMs. Every VM is isolated from the other ones, helping in the security

Lesson 3

Process concept Program + Data + state

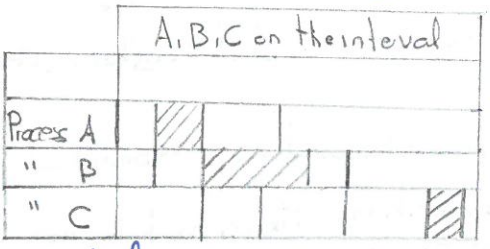
Also it can be defined as the processing unit managed by the OS. The process must be allocated in main memory, called memory image, if we want to execute it. The information about the state is stored in a data structure called Process Control Blod (PCB), there it's stored the:

- Identifier: Each process has an uniquely identifier
- State: (See later)
- Priority: To make some processes more relevant, important or urgent
- Program counter
- Memory pointers: To code, to data...
- Context data: Data in the registers of the processor
- I/O status info: Files in use, I/O requests...
- Accounting info: Remember to improve performance

This information in the PCB is crucial when an interrupts must be assisted. Also important to multiprogramming

Multitasking

↳ Real parallelism between I/O and processor, alternating processes in I/O and processing phases and memory capable of storing several processes



One more than one process in the system. If we add a lot of process we can have the opposite effect, overpassing the limit of the system, accesing to secondary memories, producing a collapse.

- The more processes, the more likely to find processes to be executed
- The more processes, the more memory needs
- The more processes, the less memory for each process
- The less memory for each process, the more memory faults
- The more memory fault, the less performance

This is in a virtual memory, if we used the real memory wall limit us. Resident set is the entire process

Process information

The OS is the entity that manages the use of system resources by processes. It schedules and dispatches processes for execution by processes and responds to request by user. But, what information does the OS need to control processes and manage resources for them? There are three main groups:

- Processor status: Set a register (PSW) as well as a set of attributes of the process store in the PCB

When executing is in the processor, but when it leaves the state must be in PCB

To the process concept a new element could be add, the Stack = used to store parameters and calling addresses for procedures. Nowa process image is a program + data + PCB + stack. Process attributes could be defined in:

1- Process identification, which includes:
   - Process identifier
   - Parent process identifier
   - User identifier
2- Process state information including
   - User visible registers
   - Control and status registers
     - Program counter
     - Condition codes: Result of the most recent arithmetic or logical operation
     - Status information: Interrupt enable/disable, execution mode...
   - Stack pointer: Points to the top of the stack.
3- Process control information
   - Scheduling and state information
     - Process state, priority, scheduling, Event...
   - Inter process communication
   - Process privileges
   - Memory management
   - Resources ownership and utilization

Types of Operating Systems

- Mainframe OS: Room-sized computers in major corporate data centers with millions of Gb of data. Oriented toward processing many jobs at once with prodigias amounts of I/O offering three kinds of services:
  - Batch processing (no interact with user)
  - Transaction processing with a large number of requests
  - Timesharing allowing hundreds of users working at the same time

  Most of them are based on UNIX OS
- Server OS: Very large personal computers, workstations or even mainframes. Serve to multiple users at once over a network and shares hardware and software resources. Services like printer services, web services... OS like Solaris, Linux or Window Serve.
- Multiprocessor OS: High computational power by join processors. We can distinguish amont multiprocessor (different nodes interconnected by a network to share) or multicores (the different cores in a processor share the main memory). Hard to have apps make use of all this computing power. Most popular: Linux and Windows
- Personal computer (desktop) OS: It supports to a single user, but allows multiprogramming. Used for word processing, games and Internet access. Market dominated by Microsoft Windows.
- Handheld computer OS: Tablets, smartphones (known as PDAs). Charectize by owning multicore CPUs, GPS, cameras... and sophisticated OS. Market dominated by Android
- Embedded OS: On computers that control devices which dont accept user-installed software (it is in the ROM): Microwaves, TVs, cars, MP3 players...
- Sensor-Node OS: Sensors are everywhere, to detect fires, measure temperatures... A smart city is carried out thanks to different kinds of sensors.
  They have limited power and must work for long periods of time in harsh conditions. The OS has to be small and simple because the nodes have little RAM and batteries as a major issue
- Real-time OS: They have time as a key parameter, as controlling a factory to make some actions in hard deadlines to avoid catastrophic consequences, something crucial. The OS could be eCos just a library linked in with the application programs, with everything tightly coupled and no protection between pots
- Smart Card OS: The smallest ones, with credit-cards-sized containing a CPU chip. They have severe processing power and memory constraints.

Activation of the Operating System

We have three sources to activate the OS:

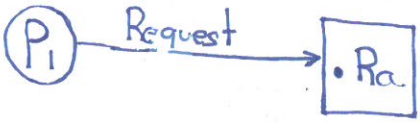
- System calls
- External interrupts → I/O, clock, another processor
- Hardware exceptions

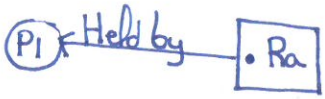
The OS activations follows the following steps:

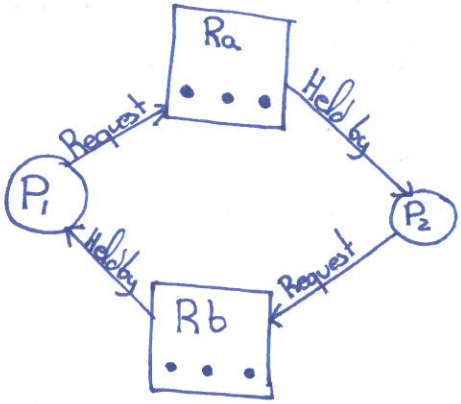
1- Interruption requires OS pays attention. Change to kernel mode
2- OS runs and saves the state of process in execution, before jumping to the interrupt subroutine
3- OS carries out the require work (subroutine)
4- When the OS finishes, Scheaulr choose a process to be executed
5- Distpacher restore the state of selected process and change to user mode.

Lesson 4

Rag representation







How many processes? 2
How many resources? 2
How many instances per resources? [3,2] [Ra, Rb]
How many request arrows? 2
How many held by arrows? 2

1. If the RAG doesn't contain loops, then there's no deadlock situation
2. If the RAG has a loop and the resources only have 1 instance ⊡, there's a deadlock situation

Coffman conditions

- Mutual exclusion 1 resource at a time
- Hold and wait:
- No preemption:
- Circular wait: A closed chain exists, each process holds at least one resource needed by the next process in the chain

When this happen, we can prevent, avoid or detect. How?

→ Mutual exclusion: Not possible
→ Hold and wait: Pedir todos los recursos y hacerlos a cuando pueda realizarlos simultaneamente
→ No preemption: 2 ways
→ Circular wait: Matching a linear schemes

Sabemos que diagonales de un hexágono = 9

n = número de lados

$$\frac{n(n-3)}{2} = \text{diagonales} \Rightarrow \frac{n(n-3)}{2} = 9 \; ; \; n(n-3) = 9 \cdot 2 \; ; \; n^2 - 3n = 18$$

aplicamos ecuación de segundo grado

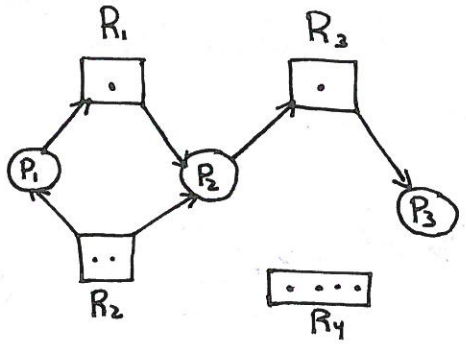
$$n^2 - 3n - 18$$

$$n = -3$$

$$\boxed{n = 6}$$

Deadlock avoidance

- Vector R (total Resources)
- Vector V (total aVailable resources)
- Matrix C (Claim)
- Matrix A (Allocation) → Held by
- Matrix N (Needed) = N = C-A



R = }1, 2, 1, 4{ V = }0, 0, 0, 4{

A:

	R_1	R_2	R_3	R_4
P_1	0	1	0	0
P_2	1	1	0	0
P_3	0	0	1	0

C:

	R_1	R_2	R_3	R_4
P_1	1	1	0	0
P_2	1	1	1	0
P_3	0	1	0	0

Banker's Algoritm

C

	R_1	R_2	R_3
P_1	3	2	2
P_2	6	1	3
P_3	3	1	4
P_4	4	2	2

A

1	0	0
6	1	2
2	1	1
0	0	2

N = C-A

2	2	2
0	0	1
1	0	3
4	2	0

Vec R = }9, 3, 6{

Vec V = }0, 1, 1{

a) V}0, 1, 1{ > N_2 (}0, 0, 1{) → V = V + A_2 = }0, 1, 1{ + }6, 1, 2{ = }6, 2, 3{

V}6, 2, 3{ > N_1 (12, 2, 2{) → V = V + A_1 = }6, 2, 3{ + }1, 0, 0{ = }7, 2, 3{

V}7, 2, 3{ > N_3 (}1, 0, 3{) → V = V + A_3 = }7, 2, 3{ + }2, 1, 1{ = }9, 3, 4{

V}9, 3, 4{ > N_4 (}9, 2, 0{) → V = V + A_4 = }9, 3, 4{ + }0, 0, 2{ = }9, 3, 6{ = Vec R

We are in a safe state since we used all the avaible resarces. [P_2, P_1, P_3, P_4]

↳ or safe sequence

Example of Banker's algorithm

- Could a request be guaranteed?
- Safe state or not?

Estado virtual:

$A_1 = A_1$ + request

$N_1 = N_1$ - request

$V_1 = V_1$ - request

Lesson 5: Scheduling

The task to choose a process or scheduling, is carried out by the scheduler
The info related to the process is store in the PCB. With this info, the scheduler can decide which processes have to be choosen. It is selected by the dispatcher, restoring the context and giving up control for its execution. There are many reasons to choose a process

- Termination of one process leading to the choice of another to take over the CPU
- End of the time assigned to a process, so another one is needed
- Arrival of a process with higher priority
- The process that occupies the CPU has to perform I/O operations and therefore another process would copy the CPU

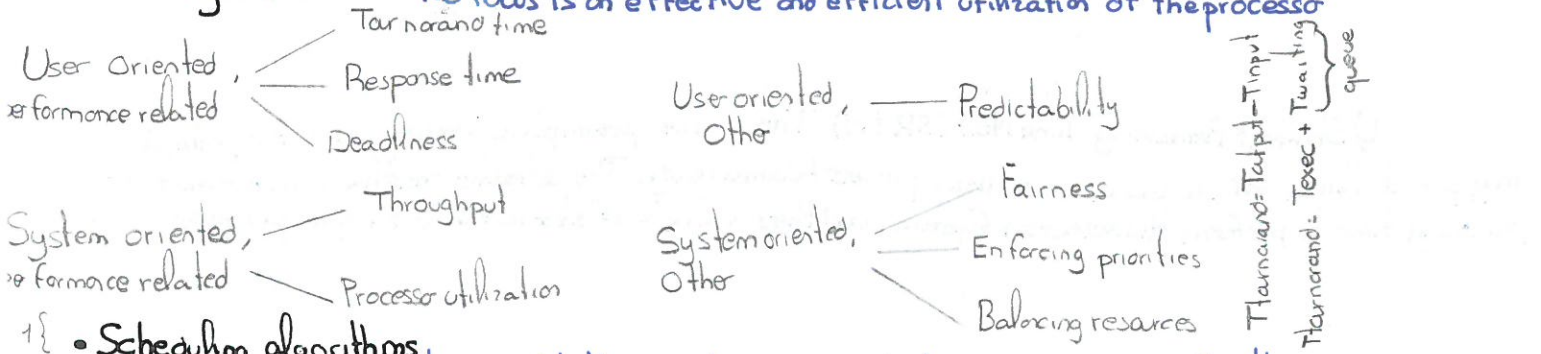
There are three types of scheduling depending on the decisions taken:

- **Long-term scheduling**: Takes decisions about the admission of new processes. The process comes when a process wants to start but also when a process finishes. If the CPU utilization is lower it could admit more processes (multiprogramming)
- **Medium-term scheduling**: Takes the decision about which processes are moved to secondary storage and viceversa. Takes the info from the PCB
- **Short-term scheduling**: Chooses what process enters to Running State. Invoke at each change of context (system call, interrupts due to I/O operations...)

However, in a computer there's a variety of resources to be scheduled:

a) **Memory**: In systems with virtual memory, an exchange of the page frames is produced
b) **Disk**: Processes make disk access requests that must be planned to determine in what order they are served
c) **Timing mechanisms**: When a process releases a mutex that was closed or when you release a lock associated with a file, you have to plan which of the processes that are waiting to use this resources

- **Performance criteria** (two groups)
  - **User oriented**: Related to the behavior of the system as perceived by the individual user or process
  - **System oriented**: The focus is on effective and efficient utilization of the processor

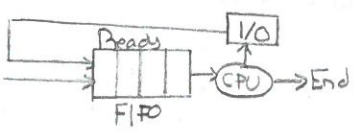


- **Scheduling algorithms**

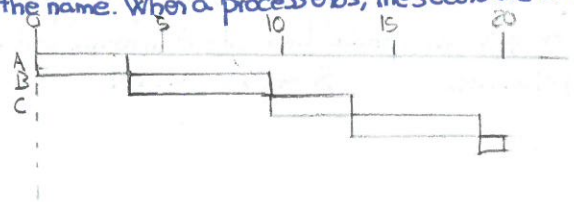
In some cases we have to preempt the CPU to a process, that is remove a process from the processor

Non-preemptive algorithms: The executed process is not moved from the Running State until it terminates or blocks itself

a) **First Come First Served (FCFS)**: Described by the name. When a process ends, the second one starts



Process	Arrival time	Service time
A	0	3
B	2	6
C	4	4
D	6	5
E	8	2

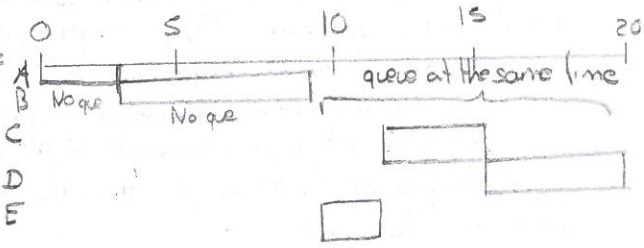


Process	Arrival time	Finish time	Turnaround time	Served time	Waiting time
A	0	3	3	3	0
B	2	9	7	6	1
C	4	13	9	4	5
D	6	18	12	5	7
E	8	20	12	2	10

Turnaround time = Finish time − Arrival time
Waiting time = Turnaround − Served time

b) Shortest Process Next (SPN): Time as a key factor It benefits to CPU-oriented process, those that don't spend so much time in I/O operations. This takes the shorter process and jumps them in to the head of the queue. Can hapen starvation (longer processes will never execute. Another issue is the difficult of estimating the requiring process time to order the queue

Process	Arrival time	Finish time	Trarund time	Servel time	Waiting time
A	0	3	3	3	0
B	2	9	7	6	1
C	4	15	11	4	7
D	6	20	14	5	9
E	8	11	3	2	1



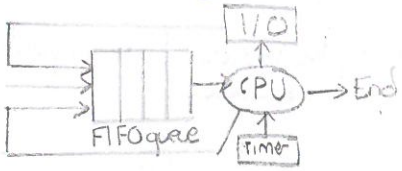
c) Priorities: Each process is assigned a priority and the scheduler always chooses the process with the higher one, if two processes have the same priority, FCFS is applied

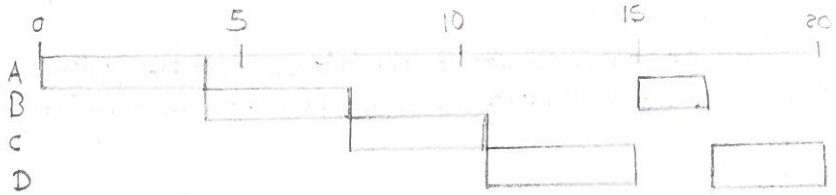
Priority can be assigned internally, taking in to account parameters that the OS can measure (limit of time or memory requirements). External assignment takes into account external parameters that OS cannot measure such as billing for using the computational resources

We can distinguish between static priorities which don't change during the life of a process (it leads to starvation to the low priority process) and dynamic priority processes where the priorities could be modified during the life of a process in order to avoid starvation

Preemptive algorithms: Those algorithms remove a process from the processor. Maybe interrupt, it happens when an interrupts occurs, that can be also periodically, based on a clock interrupt

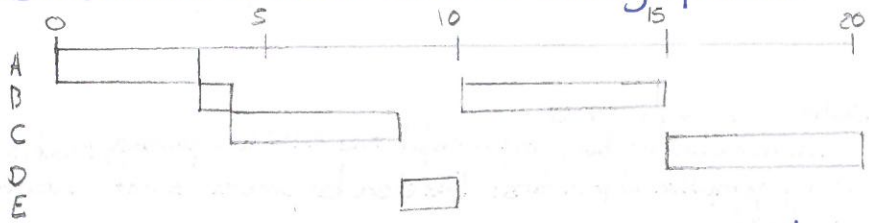
a) Rand Robin (RR): A clock interrupt is generated at periodic intervals called quantum (q), when this occurs the current process is placed in the queue and the next ready is chosen using FCFS. The problem is if quantum is actually short, the sort process will move through the system quickly and if the quantum is so long, RR becomes FCFS





b) Shortest Remaining Time Next (SRTN): This is the preemptive version of SPN, scheduler may preemt the current process when a new process becomes ready. The scheduler must have an estimate of processing time to perform the selection function, and there's a risk of starvation of longer processes

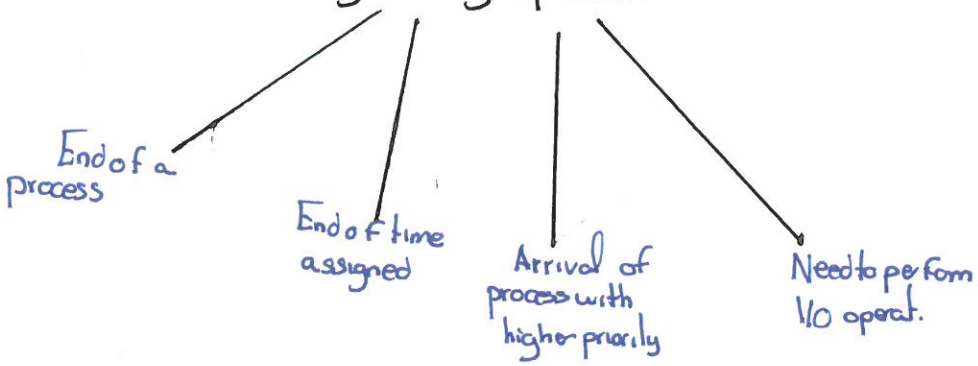
Process	Arrival Time	Served time
A	0	3
B	2	6
C	4	4
D	6	5
E	8	2



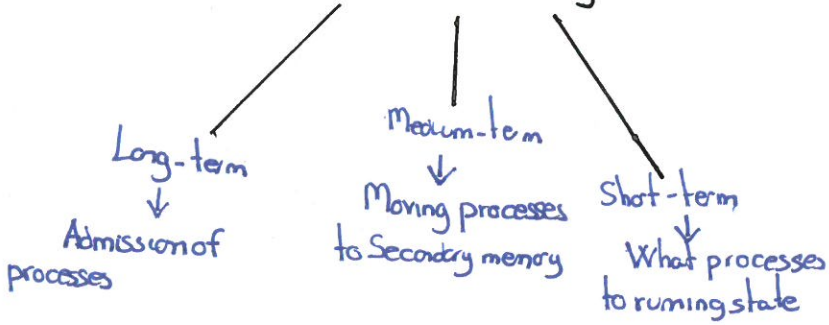
c) Priorities: This algorithm is the preemptive version of the non-preemptive one. The same concepts are applied here about internal and external assigments or priorities as well as static or dynamic priorities. Same as SRTN

Lesson 5: Scheduling

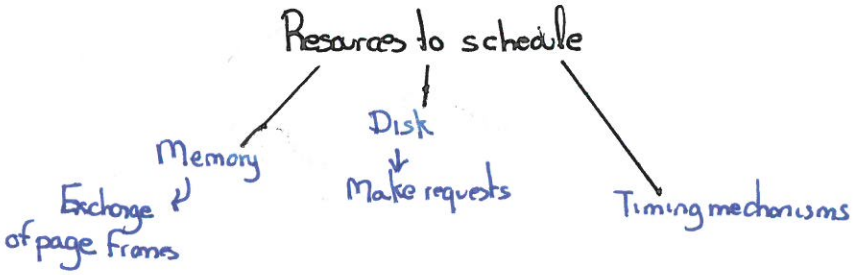
Why choosing a process?



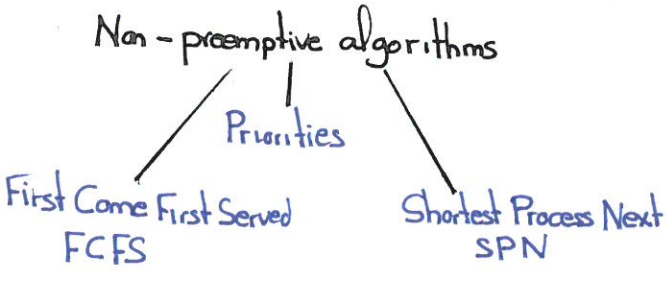
Types of scheduling



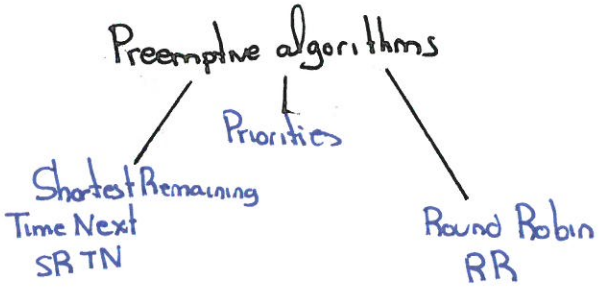
Resources to schedule



Non-preemptive algorithms



Preemptive algorithms



Lesson 6: Memory Management

- General aspects of memory management

MM is divided in the OS and a part of a program currently being executed. In uniprogramming System there's only one program in system, in multiprogramming the "user" part is subdivided to accomodate multiple processes

"User" part in memory must be further subdivided to accommodate multiple processes, memory management is in charge of control and coordinating the memory, assigning portions called blocks to the running programs to optimize performance. It wants to satisfy:

- **Relocation**: The processor HW and OS must be able to translate the memory references in the program into the actual physical memory addresses
- **Protection**: We must prevent the OS access and also to other partitions, so all generations must be checked at run time to ensure that they refer to the memory space allocated to that process
- **Sharing**: Any protection mechanism must be flexible as some processes access the same part of memory if they cooperate together

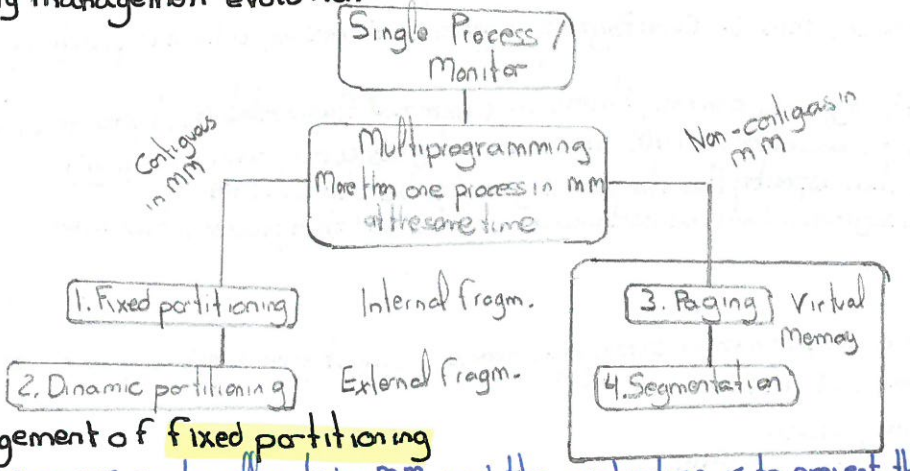
- **Swapping**: It's the step of moving processes to secondary memory in order to admit new processes, this process is close to relocation and supports multiprogramming

Using swapping we must make two decisions, the process which goes to secondary memory and the one which is relocated from secondary memory to MM. Depending of the relationship between those concepts we must know:

- **Logical address**: reference to a memory location independent of the current assignment of data to men a traslation must be made to a physical address before making the men access
- **Relative address**: Expressed as a location relative to some known point, usually a value in processor regist
- **Physical address/ absolute address**: Actual location in MM

Address translations can be done before executing the program (static relocation) or during the execution (dynamic relocation) Static relocation has necessary to translate all addresses each time the program is loaded into MM. and only a few addresses will be used. Dynamic relocation is done thanks to HW support in order to be quicker and be able to do the translation during the exec.

- Memory management evolution



- Management of fixed partitioning

Only one process can be allocated in MM and the protection is to prevent the access of addresses generated by user program into monitor space with the following mechanisms:

- **Boundary registers**: There's a register indicating the address of the end of SO, so when a process generate an address must be higher than the boundary register
- **Based on protection bits**: Addresses associated to OS have a value of 0 and associated to processes, 1.
- **Base and limit registers**: Similar to boundary registers

Some partitions could be shared among the processes. The OS takes which processes could access to those partitio
Other drawbacks like in the MM the degree of multiprogramming is limited due to the number of partitions.
If we allocate a program smaller than the partition difference between the partition and the program is waste.
This is called fragmentation. Options to assign processes:

- **Best fit**: We choose the partition where less memory is wasted
- **First fit**: The process is assigned to the first partition where it fits

- **Management of dynamic partitioning**

Partitions created on demand and if there's enough memory space. To avoid external fragmentation we use compaction, which consits in shift all processes contiguous. We fond these strats:

- **Best fit**: Check the size of all avaible main memory in order to choose where less mony is wasted
- **Worst fit**: Choose where more memory is wasted. Doesn't produce small pieces in the sort term but in the long term it will
- **First fit**: Process is assigned in the first space where it fits
- **Next fit**: We start looking for the next space chosen before

	First Fit	Next fit	Best fit	Worst fit
3Kb				
20Kb	9Kb	9Kb		9Kb
11Kb	41Kb		10Kb	
18Kb	12Kb	12Kb		12Kb
5Kb		9Kb	41Kb	
12Kb	10Kb	10Kb	12Kb	16Kb
13Kb				41Kb
9Kb			9Kb	

9K, 12K, 9K and 10K

a) **Protection and translation aspects**: We use base and limit registe. We use Partition Descripcon Table (PDT) but without assigned a base and limit because it's dynamic

b) **Sharing aspects**: They are declared as shorable or not and the OS takes note of which processes could access blocks and which not

c) **Drawbacks**: External fragmentation, compaction overhead, complex structures. But the main problem is that processes are contigous in MM so its not possible to take advantage of small spaces of memory reducing the suffering of the CPU

- **Paging**

Dividing processes into parts and allocating those parts. We split MM into fixed-lenght blocks called frames, while processes are divided into pages, the blocks reside in secondary memory and the page tempoly copied to MM

a) **Protection and translation aspects**: Each process has a Table Page which list in which frames the pages are stored

b) **Sharing aspects**: Same pages could be declared as shorable and then processes sharing them take note into its Page Table

c) **Drawbacks**: We must access MM for Table Page and physical addresses, which is pretty slow

- **Segmentation**

No partitions are made so it's dynamic scheme, there's only external fragmentation. A process is split in segments. These segment could be allocated in MM in a non-contigous way, similar as paging

a) **Protection and translation aspects**: Has its own segment table stored in MM

b) **Sharing aspects**: Some segments could be declared as shorable and then processes that share them take note into its segment Table

- **Virtual Memory**

Complements paging and segmentation managing secondary storage. Using it processes are allocated in Secondary Memory and part of them in MM (resident sets)

Consequences of using secondary store:

- More processes may be maintained in MM
- A process may be larger than MM
- If you want to increase the multiprogramming degree → Less pieces in MM. If you want to improve the performance degree → More pieces allocated in real memory
- Take care of thrashing (overload of multiprogramming producing bad performance)

To improve the Virtual Memory:

- **Fetch**: Determines when a page should be brough into MM
- **Placement**: Determines where in real memory a process piece is to reside
- **Replacement**: Deals with the selection of a page in MM to be replaced when a new page must be brought
- **Assigment** of the resident set: How many pieces of MM are assigned to an active process

Notice if a page has been modified when it's in MM and must be replaced, before moving from MM to Secondoy memory, it is necessoy to copy the modified page into secondoy memory in order to keep data consistent. To know which pages have been modified, and then a copy must be carried out and which not a second bit is introduced in the Page Table called Modified

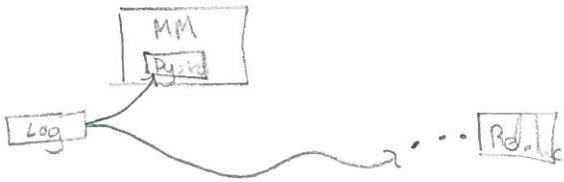
Lesson 6: Memory management

Wants to satisfy

- Relocation → translation
- Protection → To OS and other partition
- Sharing → Between processes

Swapping: Moving to secondary memory to accept more processes SM ⇄ MM

- Logical address → Non location indept. current assignment
- Relative address → location relative to known point
- Physical address → Actual location in MM



Fixed partitioning = Only 1 process in MM. Mechanism
- ↳ Boundary registers { } Boundary registers
- ↳ Protection bits: 0 associated, 1 free
- ↳ Base and limit registers

Best fit & next fit

Dynamic partition: To avoid external fragmentation, compaction

Best, Worst, First, Next } fit

Paging: Diving into parts and allocating
- ↳ Problem: access MM and physical addresses

Segmentation → No partitions

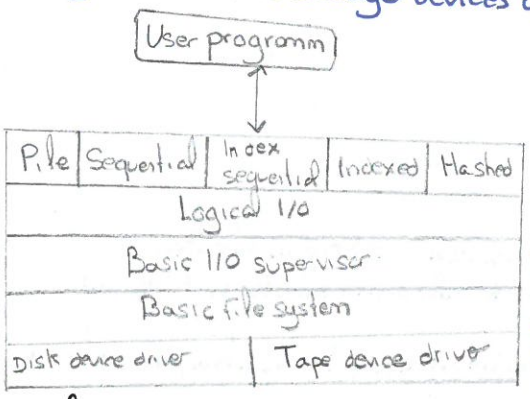
Lesson 7: File management

- Functions and structure of a File system

User uses a file as an input to applications as well as outputs where the result are putting them in Lot of operations such as create, delete, open, close, read and write a file. Then the file system permits user to create files with desiderable properties such as:

- Long-term existence: Files are stored on disk or other secondary storage and don't disappear when user logs off, in contrast of MM that is volatile
- Sharable between processes: Can have associated access permissions that permit controlled sharing, as other resources. "rwx on linux"
- Structure: A file can have an internal structure that is convenient for particular applications. In addition, files can be organized into hierarchical or more complex structure to reflex the relationships between files.

- File management System (FMS): Set of system software that provides services to users and applications in the use of files, providing the resource abstractions typically associated with secondary store, runs in kernel mode
  - Main objectives:
    - Responsible for the management of the phisical supports of the information, hiding the user the details of the physical organization of the files
    - Responsible for guaranteeing the safety and protection of the files, ensuring the integrity of the files in the event of an incident
    - Implements the high-level part of device-independant I/O
    - Performs correspondence between the logical and physical organization of the files. The logical organization will be defined by the users according to the nature of the specific application in question. Directly related to the storage devices users



Device drivers communicate directly with peripheral devices. Responsible for stating I/O operation or requests

Basic file system deals with blocks of data exchanged by disks or tapes Allocates them in secondary memory but doesn't understand the data

Basic I/O supervisor is responsible for all file I/O initiation and termination. Needed for file status, scheduling...

Logical I/O, enables users and applications to access records. Records must be translated into blocks to be understood by the I/O device

Access method provides a standard interface for apps and devices

- Files

Users must be able to manage files without numeric identifiers, the transation of numeric identifier to name is made by the FMS and each file must have a unique name to avoid ambiguity. Usually a file has 8-14 characters and 3 additional ones called extensions

FMS must also keep extra-info called attributes such as type, location, permissions, dates, sizes... Respect operations FMS must provide the utilities of create, delete, open, close, read and write among others

- Directories

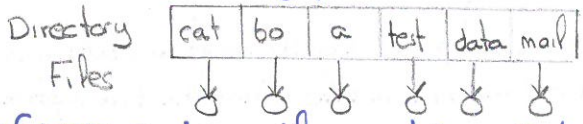
Is a file that contains information about the files, including attributes, location and ownership The information stored in a file directory cald be classified as:

- Basic info as file name or type
- Address info as the allocated size...
- Access control info such as owner, permissions...
- Usage info such as date created, date last modification...

A set of operations can be done with file directories:

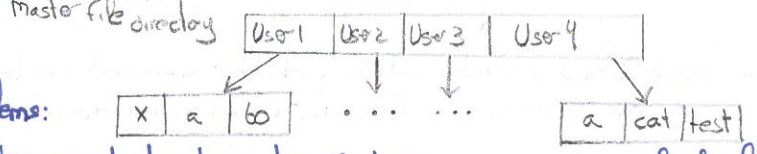
- Search
- Create/Delete a file
- List/update a directory
- Change directory

- Directory structure
  - Simple structure: One entry per file



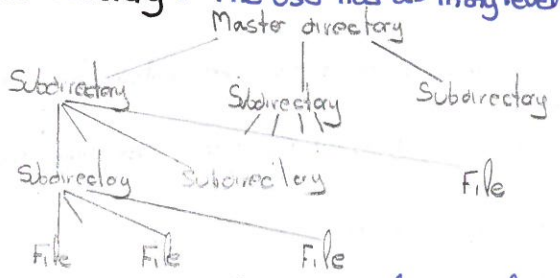
Problems:
- Amount of names can be quickly run out (max. 8 characters)
- No files with same name
- User cannot organize its own info (hell)

  - Two level structure: Different users could have same file name since they are separated



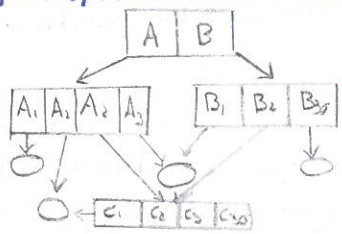
Problems:
- User cannot structure its info because a user has only level one

  - Tree-structured directory: The user has as many levels as we want making it easy to organize its info



Using this structure appears the concept of absolute and relative path as well as root directory and working directory. The problem is that we can't share files

  - Acyclic graph: Used by Unix, in this case files and file directory are organized as an acyclic graph.



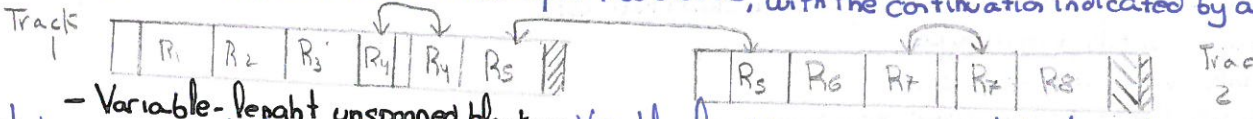
- Physical storage files

Logical information is stored in records meanwhile physical information is stored in blocks. Therefore a translation must be done by the FMS who is responsible for allocating blocks to files as well as managing free blocks in secondary storage

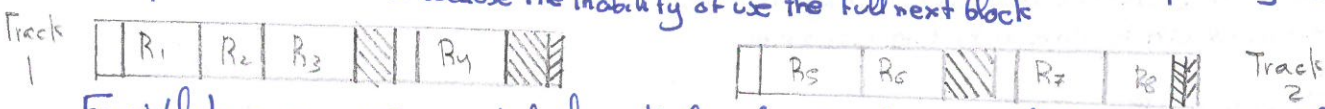
- Block allocation: Blocks has a fixed lenght to make its management easy. In addition, the relative size of a block should be compared to the average record size:
  - If we have large blocks and the records are accessed sequentially, then with just one block, we have a lot of info and the performance is improved
  - If the blocks are accessed randomly, then we have to transfer a large block with not so much useful info. The performance decreases

  - Fixed blocking: Fixed-lenght records are used, and an integral number of records are stored in blocks

  - Variable-lenght spanned blocking: Variable-lenght records are used and packed into blocks with no unused space. Thus, some records must span two blocks, with the continuation indicated by a pointer



  - Variable-lenght unspanned blocking: Variable-lenght records are used, but not packing. There is wasted space in most blocks because the inability of use the full next block



Fixed blocking is used for sequential files with fixed-lenghts. The spanned blocks technique is useful and efficient but hard to implement. The not spanned results in a wasted space and limits record size to the size of a block

Memory image:

"Set of memory addresses that a process can address". To execute a process it must be in main memory. If it's not in its space memory a failure will happen and an interrupt is sent

The space may vary, if it's in real memory all memory will be in main memory. If we talk in virtual memory, only some parts will be in main memory. There are some memory management schemes:

- One region of fixed size
- One region of variable size
- A set of fix regions of fixed size
- A set of variable regions of variable size

OS tables

- Memory tables: To keep tracking of main and secondary memory supporting swapping mechanism, which involves moving part or all process from main memory to disk
  - Allocation of main and secondary memory to process
  - Protection mechanism
  - Information to manage virtual memory
- I/O tables: Used to manage the I/O devices. It could be assigned to any process. Also it will help us to know the status and the location
- File tables: Provide information about the existence of files, their location on secondary memory, their status and other attributes

Life cycle of a process (I). Introduction (Easy)

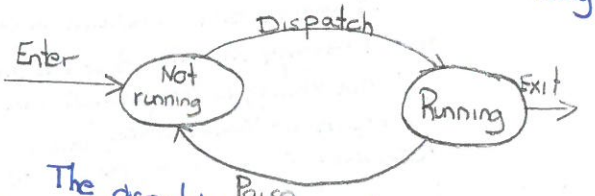
- Creation: By the user, the OS, a new batch job or another process.

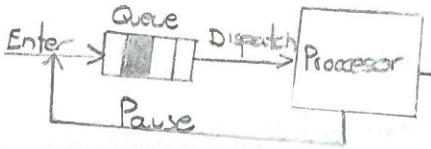
When it's created we need to assign a unique process identifier, allocate space, initialize the PCB...

- Process switching and dispatching
- Termination, due to normal completition, time limit exceded, protection error, parent termination...

Life cycle of a process (II). Two-state and five-state process model

- Two-state process model: Running or not running. Simple



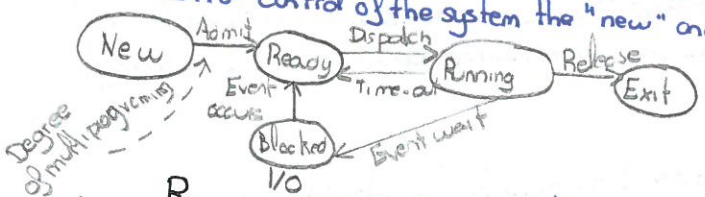


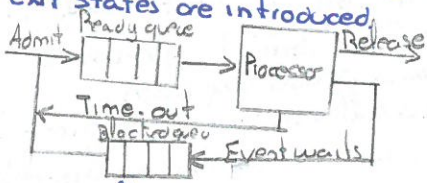
In the queue we can find processes waiting for a I/O event and some ready to be executed

The dispatcher will need to scan the list looking for the process that is not blocked and has been in the queue for the longest time. That motivated the introduction of the five-state process model.

- Five-state process model: To fix the queue problem we added the "ready" and "blocked" states.

For a better control of the system the "new" and "exit" states are introduced





Running: What is being executed in main memory. Follows the Round-Robin scheduling policy where a fixed quantum of processor time is assigned.

Ready: Process waiting to be executed in main memory. The dispatcher selects processes to move

Blocked: Process unable to execute, until an event occurs, then it's moved to the ready state

New: New processes not loaded into the main memory and not being add in the pool of executable processes

Exit: Released from the pool of executable processes after completing its execution

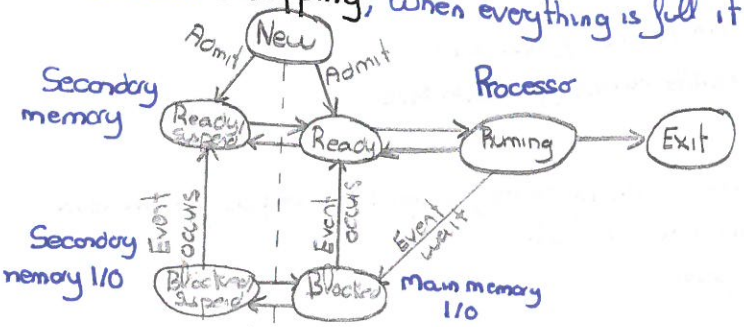
The ready and exit are very useful for the wait of enough space.

If a parent terminates, all the child processes associated with that parent may be terminated

Life cycle of a process (III). Seven-state process model

Seven-state process model: Imagine all the processes in the blocked queue, no more processes could be admited because there wouldn't be more memory. This would happen because the processor is much faster than the I/O devices, so even employing the multiprogramming could be idle

The solution resides in adding two more processes using the virtual memory. The solution is called swapping, when everything is full it moves some blocked processes to the suspended state



Process switching

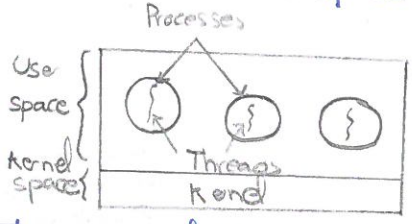
At some point a runnig process is interrupted and the OS assigns another process, so every time the OS gains the control over the current running process is called process switching

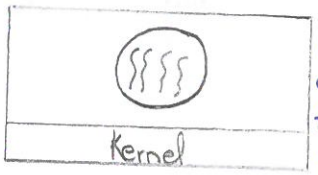
The steps involved are as follows:

1- Save the context of the processor, including the PC
2- Update the PCB currently running
3- Move the PCB to the appropriate queue
4- Select another process and its corresponding PCB
5- Update the PCB of the new process
6- Update memory management structures
7- Restore the context of the processor for the new process

Threads

Address space and a single thread of control. In many situation, it is desireable to have multiple threads of control in the same address space running in quasi-parallel, as though they were almost separate processes, except for shared address space





Reasons for having these miniprocesses
- Decomposing makes it a programming model become simpler. Also the threads add the ability for the parallel entities to share an address space, being essential for some applications

- Threads are lighter weight than processes, they are faster to create and destroy than processes (x10-100 times)

Shared: address, child processes, global variables...
Unique: PC, registers, stack, state

Services

fork(): Create a child process identical to the parent. This call returns a value of zero to child process and different zero to parent process. In this way, each process could follow its own way.
wait(): Wait for a child to terminate
exit(): Terminates process execution
getppid(): This call returns the identifier of the parent process that invokes it.
getpid(): This call returns the identifier of the process that invokes it.

When a file is created to know if the maximum space is required we should know if some information the file size is known or not. A preallocation policy requires that the maximum size of a file be declared at the time of the file creation request. However it's hard to know so we use dynamic allocation

Space is allocated to a file as one or more contiguous units, which we shall refer to as portions. The size can vary from a single block to the entire file. We must take into account:

a) Contiguity of space increases performance

b) Having large number of small portions increases the size of tables needed to manage allocation

c) Having fixed-size portions simplifies the reallocation of space

d) Having variable-size or small fixed-size minimizes waste of unused storage due to overallocation.

- Variable, large contiguous portions: Better performance, avoiding wastes. However, space is hard to reuse
- Blocks: Small fixed portions giving flexibility. They may require large tables or complex structures

File Allocation Table (FAT): To manage the blocks allocated to a file

- Contiguous allocation: A single contiguous set of blocks is allocated at a time. Pre-allocation strategy using variable-size portions

	Start block	Lenght
File A	2	5
File B	9	5

Advantages: Best one for individual sequential file because multiple blocks can be read at a time to improve I/O performance. Easy to retrive only one block

Disadvantages: Hard to find enough lenght for next files. Another problem is when files increase Also uses preallocation having the problem that the size of a file is not always known

- Chained allocation: This technique allocates individual blocks, and each block has a pointer to the next one in the chain

Advantage: It allows both pre allocation and dynamic allocation

Disadvantage: Due to the dispersion it makes difficult to follow the principle of locality

- Indexed allocation: Tries to overcome the problems with the others allocation. One block related with the rest. In a more complex situation a block can be related for several groups of blocks and lenght

External fragmentation is eliminated. The variable lenght establish an improve in locality We also need to waste a block for the index block (it determines the biggest file). This file size could be very reduce for current applications. Three main indexed allocation alternatives

a) Multilevel indexed allocation: An index block has pointes to another index block (in this case secondary stage is wasted)

↝ Example: Size block of 1024 bytes and each address ocupies 4 bytes

256 entries (1024/4). If we have 2 levels: 256x256 = 64k blocks

Data block size of 1024 bytes. Biggest size of 64 Mbytes

b) Linked indexed allocation: The directory entry points to a index block and this index blocks points to data blocks, excepting the last entry of the index block that is reserved to link another inde This technique allows to grow a file. It difficults direct accesses

c) Combined indexed allocation: Combines the two above. To allocate small files only direct access is required For medium files it's possible to use direct access plus one or two indexed levels. For big files direct plus all levels of indexed allocation.

- Free space management

A disk allocation table is needed to know which blocks of a disk are free or not

0 = Free block 1 = used block

Other method like chained free portions. The free portions may be chained together by using a pointer and lenght value in each free portion, don't need a disk allocation time, just a pointer to the next free block.

Issues → Disk quite fragmented and many portions will be a single block long. This is a very time consuming

Other method is indexing, consists of a list block where in each block we store the addresses of n free blocks. The last address of this block is used to link to another block that also contains free blocks information

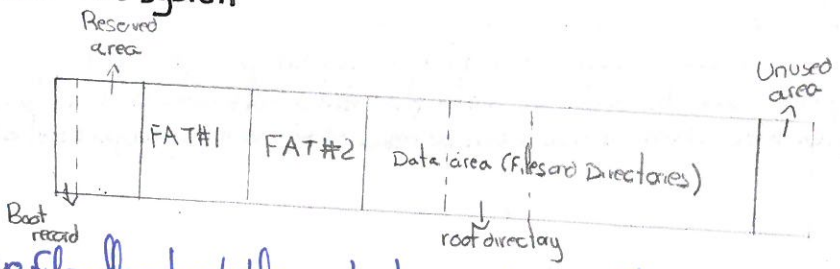
- Unix file system

Files are organized in a volume, a collection of addressable sectors in secondary memory that an OS or application can use for data storage. It contains the following elements:

- **Boot block**: Contains code required to boot the OS and load kernel into memory
- **Superblock**: Contains attributes and info about the file system, such as partition size, nº of free blocks
- **Inode table**: The collection of inodes for each file. An inode is a control structure that contains the key information of a file necessary for OS. Inode advantages:
  - Are fixed size and relatively small, to be stored in MM for long time
  - Small files can be accessed with little or no indexing, reducing processing and disk access time
  - The maximum size of a file is large enough to satisfy almost all applications
- **Data blocks**: Storage space avaible for data files and subdirectories

The files are assigned in blocks. The assigment is dynamic, that means under demand. May not be continuous.

- MS-DoS file system



Two file allocation table as a back-up mechanism. There's a secondary boot sector that allows access to the boot code in case of an active partition

FAT16 → Block disk of 16bit, 2^{16} possible addresses

FAT32 → Block disk of 32bit, 2^{32} possible addresses (4 G)

Each entry contains an address to the next block of a file and a mark

1) $P_1 = 10$
$P_2 = 4$
$P_3 = 5$
} Arrive at the same time.

FCFS, calculate the Tarand and Waiting Times

a) Arrivals: 1, 2, 3

Tarand P_1 = 10 - 0 = 10 Tarand P_2 : 14 - 0 = 14 Tarand P_3 = 19 - 0 = 19

WT P_1 : 10 - 10 = 0 WT P_2 : 14 - 4 = 10 WT P_3 : 19 - 5 = 14

P_1	P_1	P_1	P_4	P_1	P_1	P_1	P_1	P_1	P_1	P_2	P_2	P_2	P_2	P_3	P_3	P_3	P_3	P_3
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19

igual pero al revés

b) Arrivals: 3, 2, 1

Tarand P_3 = 5 - 0 = 5 Tarand P_2 : 9 - 0 = 9 Tarand P_1 = 19 - 0 = 19

WT P_3 : 5 - 5 = 0 WT P_2 : 9 - 4 = 5 WT P_1 = 19 - 10 = 9

2)

Process	Arrival Time	Run Time	Priority
A	1	8	2
B	2	2	4
C	3	1	3
D	4	2	4
E	5	5	1

Turnroad time?
Waiting time?
Response time?
} Average
CPU utilization?

a) Non-Preempive Priorities

Tarand time: Cuando acaba - Arrival time

TaA: 9 - 1 : 8 TaD: 19 - 4 = 15 Average = 11'8

TaB: 17 - 2 = 15 TaE: 14 - 5 = 9

TaC: 15 - 3 = 12

Waiting time = Tarand - Run time

WTA: 8 - 8 = 0 WTD: 15 - 2 = 13 Average: 8'2

WTB: 15 - 2 = 13 WTE: 9 - 5 = 4

WTC: 12 - 1 = 11

Response time = Cuando empieza - Arrival time

RTA: 1 - 1 : 0 RTD: 17 - 4 = 13 Average = 9'2

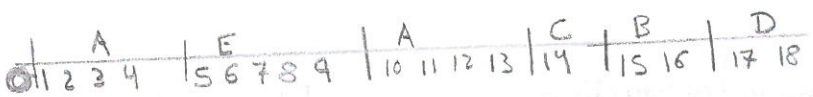
RTB: 15 - 3 = 12 RTE: 9 - 5 = 4

RTC: 14 - 3 = 11

CPU utilization: total - Donde empieza / total

CPUU A: 19 - 1 / 19

6) Preemptive Priority



Turnd = Cuando acaba - Arrival

TaA: 14-1 = 13
TaB: 17-2=15
TaC: 15-3=12
TaD: 19-4=15
TaE: 10-5=5

Average = 12

Waiting Time = Ta — Runtime

WTA: 13-8=5
WTB: 15-2=13
WTC: 12-1=11
WTD: 15-2=13
WTE: 5-5=0

Average = 8'4

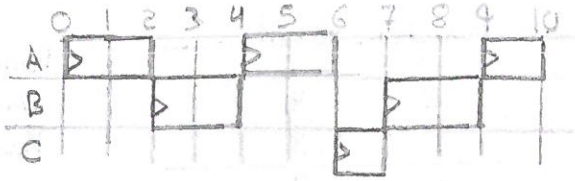
Response Time: Cuando empieza - Arrival

RtA: 1-1=0
RtB: 15-2=13
RtC: 14-3=11
RtD: 17-4=13
RtE: 5-5=0

Average = 8'2

3) Round-Robin. CPU utilization? Turnaround? Waiting time? q=2

Process	Arrival Time	Run Time
A	0	5
B	1	4
C	3	1



Turnaround = Cuando acaba - Arrival

TaA: 10-0=10
TaB: 9-1=8
TaC: 7-3=4

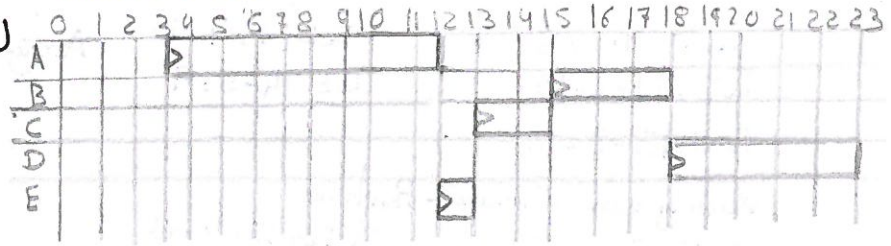
Waiting time = Ta - Runtime

WTA: 10-5=5
WTB: 8-4=4
WTC: 4-1=3

4)

Process	Arrival time	Run Time
A	3	9
B	5	3
C	7	2
D	8	5
E	11	1

a) SFJ



Tarand = Cuando acaba - Arrival

TaA: 12-3=9 TaB: 18-5=13 TaC: 15-7=8 TaD: 23-8=15 TaE: 13-11=2

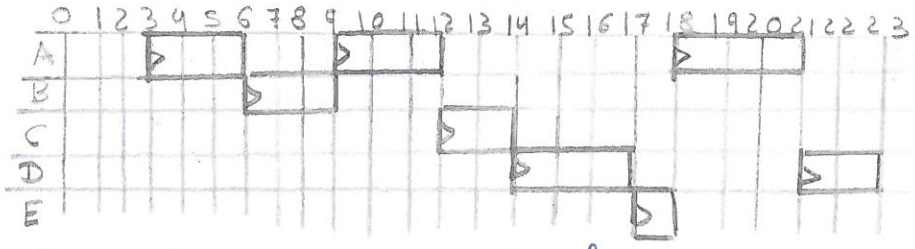
Waiting time: Ta - Runtime

WTA: 9-9=0 WTB: 13-3=10 WTC: 8-2=6 WTD: 15-5=10 WTE: 2-1=1

Response time = Cuando empieza - Arrival

RtA: 3-3=0 RTB: 15-5=10 RTC: 13-7=6 RTD: 18-8=10 RTE: 12-11=1

b) Round-Robin q=3



Response time: cuando empieza - Arrival

RtA: 3-3=0
RtB: 6-5=1
RtC: 12-7=5
RtD: 14-8=6
RtE: 17-11=6

Tarand: Cuando acaba - Arrival

TaA: 21-3=18
TaB: 9-5=4
TaC: 14-7=7
TaD: 23-8=15
TaE: 18-11=7

WT: Ta — Runtime

WTA: 18-9=9
WTB: 4-3=1
WTC: 7-2=5
WTD: 15-5=10
WTE: 7-1=6

(P0) Partitioning

355 i) addr ≤ LR
355 ≤ 105 ? X

(X,Y) es INVALID por el formato, que no es aceptado

102 i) addr ≤ LR
102 ≤ 105 ? ✓

ii) base + addr
27 + 102 = 129

(P0) Paging

305

i) 355 { 355/100 = 3 (page size)
355 % 100 = 55 } ⇒ (3, 55)

ii) offset < page size ? ✓
55 < 100

iii) page ≤ PTLR ?
3 ≤ 3 ? ✓

iv) page ⇒ frame
3 0

v) frame * page size + offset:
= 0 * 100 + 55 = 55

Tema 5

Turnaround = Cuando acaba - Arrival time
Waiting Time = Turnaround - Runtime
Response Time: Cuando empieza - Arrival time

Formas de ordenar
↳ SFJ: El más pequeño Run Time va primero
↳ Round-Robin: With q=x. Se ejecuta x y va por orden de llegada, cuando acaba va al final

Tema 6

Partitioning

1) Logical address ≤ Limit [301] (3,01) No vale
2) Base + direcaón

Paging

Ejemplo ⇝ 355 (Page size) page size = 100

i) 355 // 100 = 3 } (3;55)
   355 % 100 = 55

Page

ii) offset < page size? ✓
   55 < 100

iii) Page + RBTP? ✓
   3 3

iv) Page => frame
   3 ⇒ 0

v) frame * page size + offset

Segmentation

(3,01) [301] no vale

(2, 125)
↳ 2 + RBTS
offset ≤ lenght
offset + base

5)

	Partitioning		Segmentation	
	RBase	RLimit	RBTS	RLTS
P_0	4	96	3	2
P_1	1300	302	0	2

	Logic. address	Partitioning	Paging	Segmentation
P_0	101	Error	701	Invalid
	323	Error	1023	Invalid
	(0,30)	Invalid	130	440
	(2,125)	Invalid	Error	Error

	Logic address	Partitioning	Paging	Segmentation
P_1	101	1401	1	Invalid
	323	ERROR	523	Invalid
	(0,30)	Invalid	330	233
	(2,125)	Invalid	Error	825

(P_0)

Partitioning

101 i) Logic address ≤ Limit ?
101 ≤ 96 x

323 i) Logic address ≤ Limit?
323 ≤ 96 x

Paging

101 i) 101 /100 = 1
101 % 100 = 1
(1, 01)

ii) Offset < page size?
1 < 100 ✓

iii) Page + RBTP
①

iv) Page ⇒ Frame
1 ⇒ 7

v) frame * page size + offset
7 * 100 + 1 = 701

323 i) 323 / 100 = 3
323 % 100 = 23
(3,23)

ii) offset < page size?
23 < 100 ✓

iii) Page + RBTP
③

iv) Page ⇒ Frame
3 ⇒ 10

v) frame * page size + offset
10 * 100 + 23 = 1023

(0,30) ii) Offset < page size
30 < 100

iii) Page + RBTP
⓪

iv) Page ⇒ frame
0 ⇒ 1

v) frame * page size + offset
1 * 100 + 30 = 130

(2,125) ii) Offset < page size?
125 < 100 X

Segmentation

(0,30)
i) Offset ≤ Lenght?
30 ≤ 31 ✓

ii) 0 + RBTS = 3

iii) Offset + base =
= 30 + 410 = 940

(2,125)
i) 2 + RBTS
2 + 3 = 5

ii) Offset ≤ Lenght?
125 ≤ 95 X

(P_1) Partitioning

101 i) Logic.address ≤ Limit?
101 ≤ 302 ✓

ii) Base + address
101 + 1300 = 1401

323 i) Logic.address ≤ Limit?
323 ≤ 302 X

Paging

101 i) 101 / 100 = 1
101 % 100 = 1
(1,01)

ii) Offset < page size?
1 < 100 ✓

iii) Page + RBTP
①

iv) Page ⇒ Frame
1 ⇒ 0

v) frame * page size + offset
0 * 100 + 1

323 i) 323 / 100 = 3
323 % 100 = 23
(3,23)

ii) offset < page size?
23 < 100 ✓

iii) Page + RBTP
③

iv) Page ⇒ frame
3 ⇒ 5

v) frame * page size + offset
5 * 100 + 23 = 523

1)

	First - Fit	Best - Fit	Worst - Fit	Next - Fit
3k				
20k	9k		12k	9k
11k		10k	4k	
18k	12k		10k	12k
5k	4k	4k		4k
12k	10k	12k		10k
13k			9k	
9k		9k		

4) Translate logical addresses into physical addresses

	Partitioning	Paging	Segmentation
713	1513	201	Invalid
2550	Error	Error	Invalid
(2,50)	Invalid	562	3265
(0,80)	Invalid	2128	592

Partitioning

i) Logical address ≤ Limit? ✓
713 ≤ 1524

i) Logical address ≤ Limit? ✗
2550 ≤ 1524

ii) Base + address
800 + 713

Paging

i) 713 / 512 = 1 (1,201)
713 % 512 = 201

ii) offset < Page limit? ✓
201 < 512

iii) Page + RBTP
①

iv) Page ⇒ Frame
1 ⇒ 0

v) Frame * page size + Offset
0 * 512 + 201 = 201

i) 2550 / 512 = 4
2550 % 512 = 502 (4,502)

ii) Offset < Page size
4 < ????

(2,50)
ii) offset < Page size
50 < 512

iii) Page + RBTP
②

iv) Page ⇒ Frame
2 ⇒ 1

v) frame * page size + offset
1 * 512 + 50 = 562

(0,80)
ii) offset < Page Size ✓
80 < 512

iii) Page + RBTP
0

iv) Page ⇒ Frame
0 ⇒ 4

v) Frame * page size + offset
4 * 512 + 80 = 2128

Segmentation

(2,50)
i) offset ≤ lenght? ✓
50 ≤ 100

ii) offset + base
50 + 3215 = 3265

(0,80)
i) offset ≤ lenght?
80 ≤ 200

ii) offset + base
80 + 512 = 592

Log address	Partitioning	Paging	Segmentation
355	905	755	Invalid
102	652	902	Invalid
(0,50)	Invalid	850	Error
(2,33)	Invalid	133	53

(P1)

Partitioning

i) Logic. address ≤ Limit?
355 ≤ 1401 ✓

ii) Address + Base
550 + 355

i) Logic. address ≤ Limit?
102 ≤ 1401 ✓

ii) address + Base
102 + 550

Paging

i) 355 / 100 = 3
355 % 100 = 55

ii) offset < page size
55 < 100 ✓

iii) Page ⇒ frame
7 ⇒ 7

iii) Page + RBTP
3 + 4 = 7

v) frame * page size + offset
7 * 100 + 55 = 755

i) 102 / 100 = 1
102 % 100 = 02

ii) Offset < page size?
02 < 100

iii) Page + RBTP
1 + 4 = 5

iv) Page ⇒ frame
5 ⇒ 9

v) frame * page size + offset
9 * 100 + 2 = 902

(0,50)

ii) Offset < page size?
50 < 100 ✓

iii) Page + RBTP
0 + 4 = 4

iv) Page ⇒ Frame
4 ⇒ 8

v) frame * page size + offset
8 * 100 + 50 = 850

(2,33)

ii) Offset < page size?
33 < 100 ✓

iii) Page + RBTP
2 + 4 = 6

iv) Page ⇒ frame
6 ⇒ 1

v) frame * page size + offset
1 * 100 + 33 = 133

Segmentation

i) Page + RBTS
(0)

ii) offset ≤ Lenght?
50 ≤ 40 ✗

i) Page + RBTS
(2)

ii) offset ≤ Lenght?
33 ≤ 103 ✓

iii) Offset + base
33 + 20 = 53

P1 Paging

(0,30) ii) Offset < page size? iv) Page ⇒ Frame
30 < 100 ✓ 0 3
iii) Page + RBTP v) Frame * page size + offset
0 3 * 100 + 30 = 330

(2,125) ii) Offset < page size?
125 < 100 ✗

Segmentation)

(0,30) i) 0 + RBTS = 0 iii) Offset + base
ii) Offset < Lenght? ✓ 30 + 203 = 233
30 < 125

(2,125) i) 2 + RBTS = 8
ii) Offset ≤ Lenght?
125 ≤ 400 ✓ iii) offset + base
125 + 700 = 825

6) P0

Logic. address	Partitioning	Paging	Segmentation
355	ERROR	55	Invalid
102	129	202	Invalid
(0,50)	Invalid	550	350
(2,33)	Invalid	633	158

Partitioning

355 i) Log address ≤ Limit?
355 ≤ 105 ✗

102 i) Log address ≤ Limit?
102 ≤ 105 ✓
ii) Base + Log address
102 + 27 = 129

Paging

i) 355/100 = 3
355 % 100 = 55
ii) Offset < Page size?
55 < 100 ✓
iii) Page + RBTP
3 + 0 = 3
iv) Page ⇒ frame
3 ⇒ 0
v) frame * page size + Offset
0 * 100 + 55 = 55

i) 102/100 = 1
102 % 100 = 02
ii) Offset < Page size?
2 < 100 ✓
iii) Page + RBTP
1 + 0 = 1
iv) Page ⇒ frame
1 = 2
v) frame * pagesize + offset
2 * 100 + 02 = 202

ii) offset < page size?
50 < 100 ✓
iii) Page + RBTP
0 + 0 = 0
iv) Page ⇒ frame
0 ⇒ 5
v) frame * page size + offset
5 * 100 + 50 = 550

ii) Offset < page size?
33 < 100
iii) Page + RBTP
2 + 0 = 0
iv) frame ⇒ page
2 ⇒ 6
v) frame * page size + offset
6 * 100 + 33 = 633

Segmentation

i) 0 + RBTS
0 + 4 = 4
ii) offset ≤ Lenght?
50 ≤ 90 ✓
iii) offset + base
50 + 300 = 350

i) 2 + RBTS
2 + 4 = 6
ii) Offset ≤ Lenght?
33 ≤ 250 ✓
iii) Offset + base
33 + 125 = 158

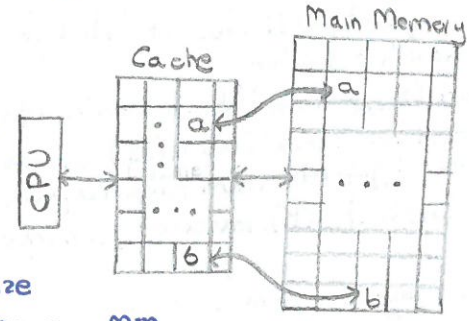
- Cache basics

The cache memory (CM) is the level of the memory hierarchy which is closest to the processor

Usually it is made of SRAM in order to transfer data to/from the processor at high speed

Its size is usually small, so that its cost is affordable and its access time is kept low

In the memory hierarchy, cache is located over Main memory (MM), interactin with it



Main memory is divided in to blocks of words of identical size

Cache is also divided into blocks of words of the same size as in MM

When the CPU request to access a speafic word, the cache controller selects the block in MM which contains the requested address, and then brings that block to cache

When the CPU generates another request, it can get the word from cache if the address is inside the same block

A miss in cache implies:

- The pipelined datapath of the CPU is stopped (stall) as many cicles as necessary
- Hardware stall: the opeartwe system does not manage it
- Once the required block has been brought to cache from the lower level:
  - The IF stage continues if the miss was due to an instruction search
  - The MEM stage is completed if the miss was due to a failed access to data in memory.
- The impact of the misses in cache can be quantified:

$$Time_{CPU} = NI \cdot \left(CPI_{exec} + \frac{Cycles\ of\ memory\ stalls}{NI}\right) \cdot cycle$$

$$Cycles\ of\ memory\ stalls = Number\ of\ accesses \cdot m \cdot Cycles_{mp}$$

- Different policies regarding these questions:
  ↳ Placement schemes, replacement schemes, policies to update main memory

- Cache design parameters

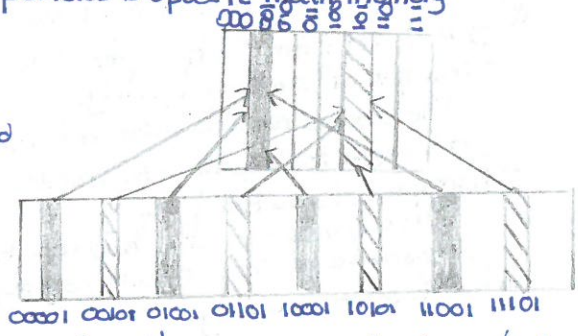
  ✍ Placement schemes

  - Direct mapping + Block identification

No options: each block of MM can be placed only at a specific block of cache

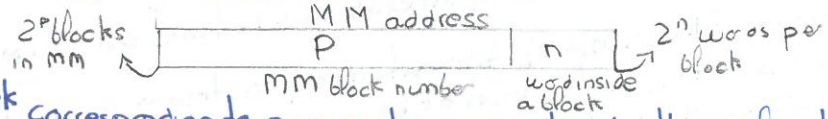
Specifically, block number i of MM must be placed at the cache block.

If the number of blocks is a power of two (as usual), its computed from the least-significants bits



Given a MM address in binary code, the block number can be obtained by discarting the n least-significant bits, being $n = \log_2(block\_size)$

The remaining p bits indicate the block number in MM

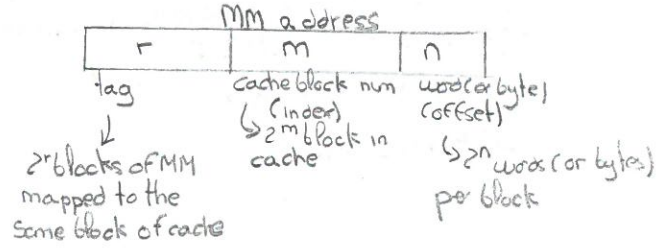


The block corresponding to p in cache is indicated by the m least-significant bits from p, being $m = \log_2(Number\_of\_blocks\_in\_cache)$. The remaining bits ($r = p - m$) from the MM block number distinguish this block from others. These bits are known as tag

Offset: Speafic word (or byte) referenced inside a block

Index: Block of cache where the MM blocks must be placed

Tag: To distinguish this block of MM from other MM blocks that could be placed at the same cache block



Unit 3: Cache memory

- Introduction

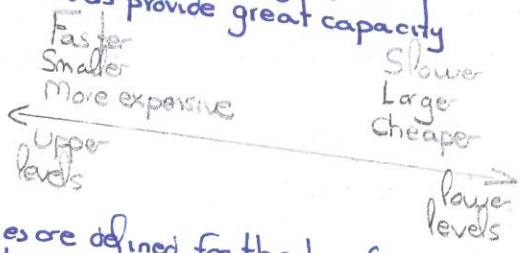
Storage capacity → Numbers of bits, bytes or words that the memory can store
Cost → Cost/bit factor
Access time (t_a) → Time taken to read/write a word, from the moment it is referenced
↳ Access speed (words/sec): inverse of t_a
Cycle time (t_c) → Minimum time required between two consecutive accesses to memory
↳ Bandwith (B): inverse of t_c: number of accesses per time unit

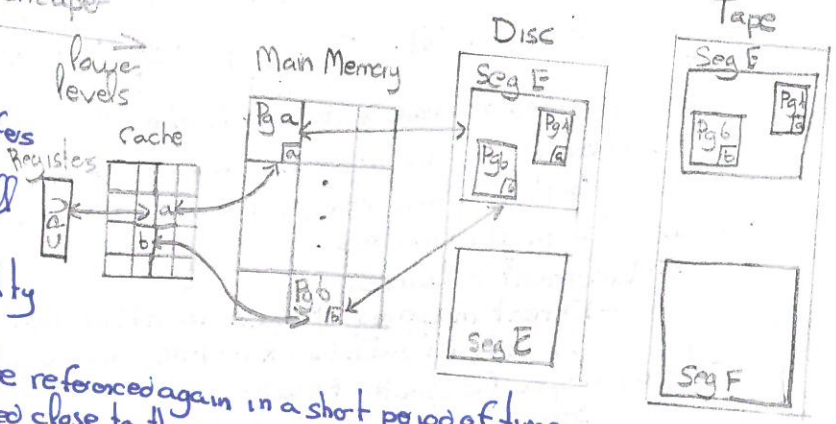
The CPU may access memory by:
- Byte level: Mem. address indicates the position of a byte in the memory
- Word level: Mem. address indicates the position of a word in the memory

A byte address requires longer addresses

✦ Necessity of a memory hierarchy

What do we expect? → Great capacity, high speed, low cost
They seem incompatible with each other, but memory hierarchy is the solution
↳ Small levels provide high speed
↳ Slow levels provide great capacity





Different sizes are defined for the transfers of information between different levels
When a word is requested, this word and all the words in the same groups are transferred

Why is it efficient? → Principles of locality
- Principle of temporal locality:
  The address recently referenced will be referenced again in a short period of time
  Advisable to keep the info recently used close to the processor
- Principle of spatial locality
  The address near to those recently referenced will be referenced immediately
  Advisable to keep the info recently used close to the processor

What favours locality?

Depends on the type of programm
- Favour → Short loops over a stack and operations with matrices
- Not favour → Frequent calls to subrutines and access to scattered data

t_a Average memory access time (AMAT): Average time required to provide a word to the processor
Hit: The request word is found in the top level of the hierarchy
p Hit rate: Fraction of the total number of memory references which result in hits
t_a Hit (access) time: Time required to access the top level (includes time to determine if it is a hit or not)
Miss: The requested word must be brought from lower levels
m Miss rate: Fraction of the total number of memory references which result in misses
t_{mp} Miss penalty: Time overhead introduced by a miss (mainly the time the block to the upper level)

$m = 1 - p$

To calculate the AMAT:

$t_a = p\,T_a + m\,T_M =$

$= T_a + (1-p)\,T_{MP}$

To calculate the AMAT:
P_i: prob. of finding the data in the "i" level
T_i: Total time to access level "i" from the CPU

$$t_a = \sum_{i=1}^{n} P_i\,T_i$$

Main mem. 32 bit addressed and 1 byte word width

Direct mapped cache with 64 blocks, each one of 16 bytes

If the word at address 1200 (decimal) were referenced, which block of cache would this reference be looked in?

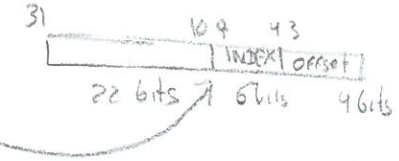
Block in MM where that address is: 1200/16 = 75

Block of cache corresponding to that MM block: 75 MOD 64 = 11

Another way to calculate

1200 decimal: ...0000 100 1011 0000

$n \log_2 16$



PPT 91

- 90% of all mem references are hits p=0.9 m=0.1
- Block size is two words
- The CPU requests 10^7 references (words) per second to mem
- 25% ref are writes
- Assume cache and MM allows to read or write 10^7 words per sec
- A single word is read/write at once
- Assume that 30% of cache blocks has been modified
- The policy to handle write misses is write-allocate

How much bandwidth is currently consumed by traffic between cache and MM write the fraction of bus bandwidth consumed in these cases:

- Write-through is used to handle write this.

We will have traffic:

1- Read or write misses:

The block is brought from MM to cache

2- Writes

The word is written in MM

Traffic 1: References frequency × Miss rate × block size:

$= 10^7 \text{ refs} \times \text{sec} \cdot 0.1 \text{ miss/ref} \cdot 2 \text{ words} = 2 \cdot 10^6 \text{ words/sec}$

Traffic 2: " " × write rate × 1 word =

$= 10^7 \text{ refs} \times \text{sec} \cdot 0.25 \text{ write/ref} \cdot 1 \text{ word} = 2.5 \cdot 10^6 \text{ words/sec}$

Total traffic: $4.5 \cdot 10^6$ words/sec

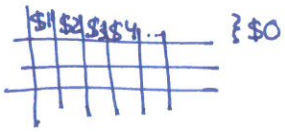
Bus bandwidth: $\frac{4.5 \cdot 10^6 \cdot 100}{10 \cdot 10^6} = 45\%$

-v var1:val1
-v var2:val2

awk → fuente
→ separador
→ variables iniciales
→ códigos

/^r*/
↳ empeza por r

$0 = linea completa



usuarios, intepetres... de /etc/passwd

CSV
ficheros buscar en casa

\ → olvidar el uso del siguiente caracter

/^\/bin\/bash$