How do processes occupy the same memory, and yet run at same time (in debugger, when looking at assembly instructions memory address).

This is done with Virtual Addresses (Where virtual memory is used). Used with Virtual management unit (VMU).

EIP: Instruction Pointer – this is the address where the program is currently executing an instruction. (eg executing line 32 of code)

**Program Counter**, is another name for EIP (**NB Terminology)**

**Stack** is used for temporary variables, when functions are called (and all the temp variables they use).

**Heap**: Where non-temporary variables are stored

A process consists of multiple parts: (**NBNB)**

-Program counter: Processor registers, current activity.

-stack: Temp variables

-heap: dynamic variables

-data: global variables created at start

- text: Program Code

Only one process can be running at a time. Once a process goes from ready, then the EIP fetches each line of code from memory (one at a time) to execute, making the process status running. Then the process does each of those line code instructions (if possible, else interacts with kernal for eg File writing).

Eg Press e key on keyboard: Kernal raises interrupt and puts key in Buffer management. Pauses whatever is running, saves its state (**Context Switch**: **Terminology Alert NBNB**) by putting that process somewhere in memory, then gets that key process.

How does a process create an interrupt: A process must be running to create an interrupt. Therefore the key e process must be running to do an interrupt.

2 types of interrupts that can occur from process:

Software Interrupt: Where you create from a process a timer (after x min, do y)

Error Interupts: Division by 0

Maskable interrupts:

Non-Maskable Interrupts:

If theres 2 system hardware interrupt requests, CPU looks at the priority number each has, and handles lowest number first.

**In Linux: A system interrupt is a system call**

A process is **passive,** when its running its **active.**

**One program**, can be **multiple processes**.

**Process in Memory:**

-stack: Temp variables  
-heap: dynamic variables  
-data: global variables created at start  
- text: Program Code

**Process State (1st major topic)**

New: Process is being created

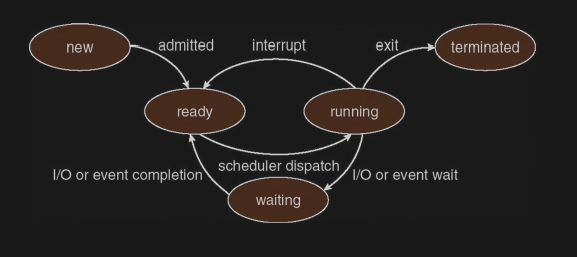
Running: Instructions being executed

Waiting: Waiting for some event to occur (eg File Writing)

Ready: Waiting to be assigned to a processor

Terminated: Finished execution

**Diagram of Process State**-Must know why a process moves from one state to another (**NBNBNB)**

****

**Process Control Block (PCB) (Or called Task Control Block)**

-Complex data type that stores a number of aspects for each process. (Can be though of as a struct, or class)

-Stores info associated with each process

-When a **content switch happens**, this is the stuff that gets saved (ie when system interrupt or system call happens) on the PCB.

Stores:  
Process State:  
Process Number:  
Process Counter

A context switch is **resource** expensive.

Any running process will have at least one thread (**NB Terminology?)**

**Not important**: Process Representation in Linux

Ready Queue: When process has ready status (from before)  
**-Ready vs waiting: NB**Waiting is equal to hardware (waiting to save/open file, network file to send/receive)  
Running, now you are interrupted, then go to ready queue (Not waiting for any hardware).

Waiting is a separate queue as hardware writing file is like “years” to the CPU.

Representation of Process Schedule: **Diagram of Process State is an easier way to explain this for a Question.**

**Schedulers (2nd main part of chp)**

**Short term (Or CPU scheduler):** To make sure one process only has a certain amount of time. (Invoked frequently, every few milliseconds). Or mouse cursor movement.

**Long term (Job Scheduler):** Looks at jobs, selects one and puts it in ready queue, being available to run for short term scheduler. **Medium Schedulers:** When ram is full, you can use virtual memory (on hard disc) to store suspended processes.

Disc thrashing: overproviding amount of processes that memory can handle, overusing hard disc virtual memory and back.’

Processors can be described:  
I/O-bound process: Spends more time doing I/O operations, many short CPU bursts – eg Word.  
CPU-bound process: Spends more time doing computations; few very long CPU bursts –eg neural networks.

**Multitasking in Mobile System**: Not important

**Context switch**

Context switch time is**: Overheads –** the system does nothing useful during this time.  
-Remember that some processes are semi-complete, so you have to also store how complete % they are ect, which all takes computation time to do. (expensive)

**Operation on Processes (NB one of 3rd main part of chapter)**

You can only create a process (spawn), from another process. (**NB)  
-eg** Notepad is a child of Explorer.exe  
-You get a parent child.  
-With Windows, initial booting processes run then die.

Processes have a **process identifier (pid)**

Parents and children can share resources, or not. Or children are a part of the parents resources.

**Process termination:** Exiting processes return a status **value**, which the parent interprets in order to decide on what to do next (error code ect)

Cascading termination: all children in tree get terminated when parent terminated.

Zombie: when no parent waiting. (without invoking wait)

Orphan: if parent terminated without invoking wait.  
-Normally is given a new parent quickly.

**Interprocess Communication (NB 4th main part of chapter)**

**IPC: Terminology NBNB (Interprocess Communication)**

Processes are sandboxed, isolated from each other. (chrome tabs) – this works by them thinking they are the only processes running.

But how do they communicate? There are defined channels they can communicate through, set by the parent.

**Two models:**

**If a process just tries to communicate with each other without IPC, the OS will end the process.**

**Shared memory:** Process A writes to shared memory, process B reads from it.  
-Only works on 1 device, cannot extrapolate over network  
-need to know about other processes and the shared memory.  
  
**Memory passing:** Process A adds a message to message queue (with FIFO), with Process B reading from the queue.  
-Works over network, so not only 1 device.  
-Only need to know about message box/queue.

Eg of process communication: Copy Paste.

**Cooperating Processes**

**Independent Process:** Cannot effect or be effected by execution of another process.

**Cooperating Process:** Can effect or be effected by execution of another process.  
-Advantage:  
-Info sharing  
-Computation Speed up  
-Modularity  
-Convenience

**Uni stands for one, like mono (NBNB)**

**Producer-Consumer Problem (NBNB Terminology)**

This process is how data flows for each message.

Describes two entities( producer + Consumer): eg A function takes in parameters (consumer), and returns something (producer).

-**Unbounded-buffer:** places no practical limit on size of the buffer  
**-Bounded-buffer**: Assumes that there is a fixed buffer size. (More realistic)

In Bounded:  
Buffer size which can only use 1 less than total  
In: in code  
Out: out code

**The code for** producer consumer don’t need to know. **(Bounded Buffer)**  
-What you need to know, producers cant add anything if buffer is full (must wait), and consumer cant consume anything if buffer is empty (must wait)

**IPC – Shared Memory**

Where you create a memory bound outside the process, allowing another process to use that shared memory.  
Advantage: Memory is mainly created and managed by process itselves. OS doesn’t do much.

Synchronization challenges: Data can be overwritten and data might not be read by consumer. So consumer must immediately read any changes to memory.

**IPC – Message Passing**

Basic system: Primitive send and receive message processes.  
-message size is fixed or variable

**Message passing**

The implementation of the communication link:

Physical:

Logical:

**Direct Communication:**

-Process must name each other explicitly.  
--Challenge: Each process must have a pragmatically coded unique identifier  
Properties:  
-Only 2 can communicate on a link  
-A link is associated with that one pair

**Indirect Communication**

Instead of directly communicating with eachother, a process can send a message to a “mailbox” (called port) with a unique ID. The other process can then get the message from that port.

**NBNB Terminology**

**Blocking:** Generate an instruction, then wait for that instruction to finish before continue.

**Non-Blocking:** Generate an instruction, then don’t wait for that instruction to finish ,immediately continue.

**Buffering**

Zero capacity: no messages queued  
Bounded capacity: messages queued to size of bounded  
Unbounded: infinite size.

Between **these** 2, not **important**

**IPC Systems – Mach (Micro Kernal operating design)  
-NBNB might be in ST1 about mach**

Between **these** 2, not **important**

**Communication in client-server systems**

**Sockets:** An endpoint for communication

Socket 0.0.0.0:0000 refers to port 0000 on host 0.0.0.0

**Remote Procedure Calls**

Big-Endian: Which of the 4 bytes stroes  
Little-Endian:

**Pipes:**

**Named pipes:** Don’t have to have parent child relationship… more powerful.

**From sockets onwards, didn’t write down much but NBNB**