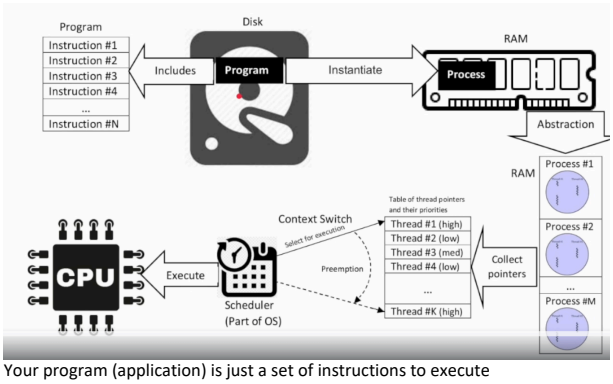
Process / Thread / CPU / RAM / Schedular

Sunday 26 January 2025 10:53

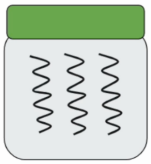
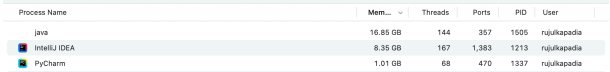
- Lets imagine that we have developed a simple Java application that gets packages as a JAR file.
- The JAR file will be on the disk somewhere.
- 

- Your program (application) is just a set of instructions to execute
- Now when I run the Java program, it will loaded into memory which would create a process.
- So a process is an instance of computer program that has its isolated space. It would include core, data, other resources allocated by OS like memory, socket, etc.
- A process is heavy weight (which will be discussed later). It lives in RAM. Just because it lives here doesn't mean it is executed.
- CPU is what executes the instructions

Process

- Process is an instance of computer program.
- It includes your code, other resources allocated by the OS like memory, socket, etc.
- A process is heavy weight
- Its expensive to create and destroy a process.

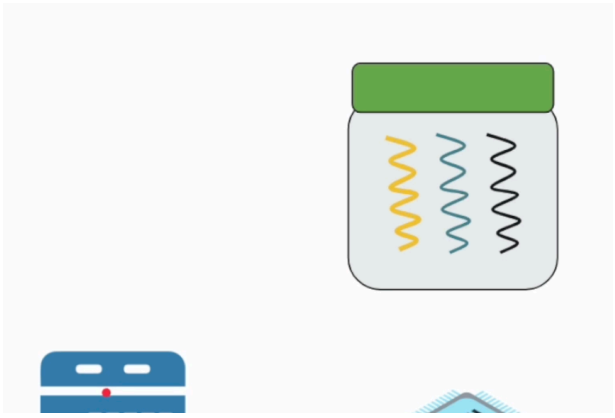
Thead

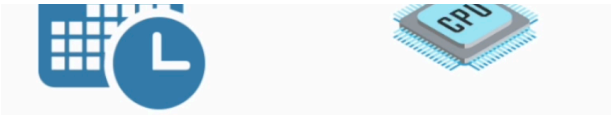
- A thread is part of a process
- 
- A process can contain one or more threads. A process will contain at least one thread.
- If we see the above picture as a process, it has 3 threads. We can imagine the green part as the allocated memory to the process. These threads can share the memory
- So we can see process as a unit of resources and thread as a unit of execution
- For e.g. see activity monitor
- 

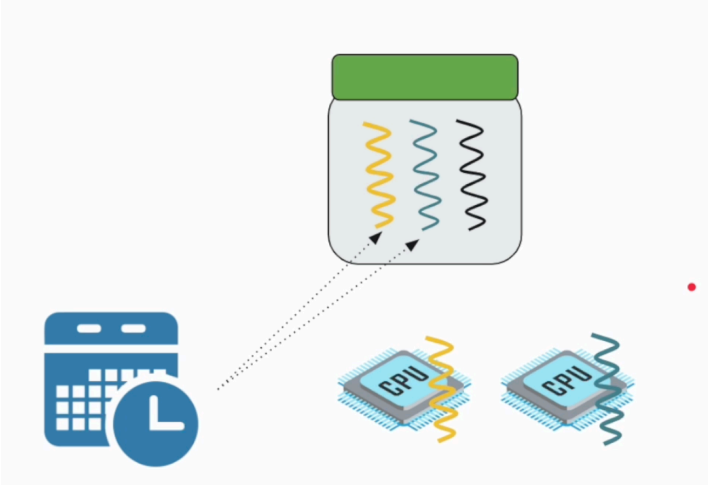
Process Name	Mem...	Threads	Ports	PID	User
java	16.85 GB	144	357	1505	rujulkapadia
IntelliJ IDEA	8.35 GB	167	1,383	1213	rujulkapadia
PyCharm	1.01 GB	68	470	1337	rujulkapadia

- It has the process name, the memory allocated to those processes and the no. of threads associated to those processes.

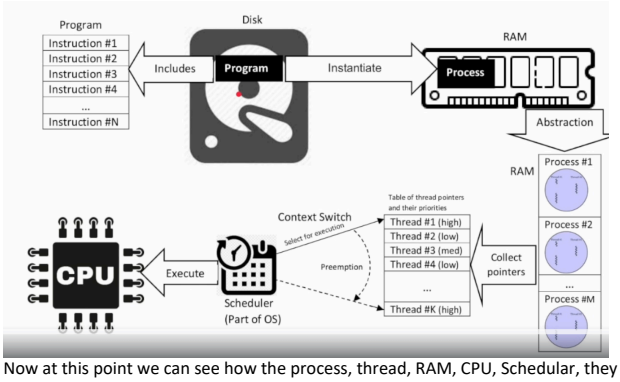
Schedular / CPU / Thread

- Lets consider this
- 



- We can consider the green box as a process that has 3 threads.
- Then we have a CPU or a processor.
- Our OS has something called schedular. Its task is to assign the thread to the CPU for execution.
- The schedular will determine how long the thread can execute.
- If the process will have only 1 thread, the schedular will assign that thread to CPU for execution for some time.
- If the process has more thread, it will assign thread one by one to the CPU.
- So essentially, when you have one process and only one processor, and when you have multiple threads, it will switch among those threads.
- 

- If you have 2 processor, it will try to assign one thread to each processor.
- CPU and processor are interchangeable terms.
- Modern CPUs come with multiple cores and here each core can be seen as a different processor.



- Now at this point we can see how the process, thread, RAM, CPU, Schedular, they are all connected.
- When we have a single processor like the above and we have multiple programs running like Chrome, IntelliJ Idea, and Java application, when they are all up and running I.e. when you have multiple processes, each and every process, they will have threads, and all these threads will be competing to be executed by the CPU.
- So the OS Schedular will keep on switching among threads for the execution. This is called Context Switch.
- So when it switches from one thread to another thread, the current thread, i.e. the execution point, the state, it has be stored so that it ca be resumed later from the point where it was stopped.
- These kind of threads are called Kernel threads or OS threads.
- This is how things are getting executed behind the scenes.

Java (Platform) Thread

- The original Java Thread was introduces 25 years ago
- The Java thread is simple a wrapper around the OS thread. So 1 Java thread = 1 OS thread.
- Why the Java thread has to be a wrapper around OS thread? This is because OS thread is the unit of scheduling, the OS thread is what gets executed by the processor. Only via OS thread, you can schedule something to execute.

Our Application Code

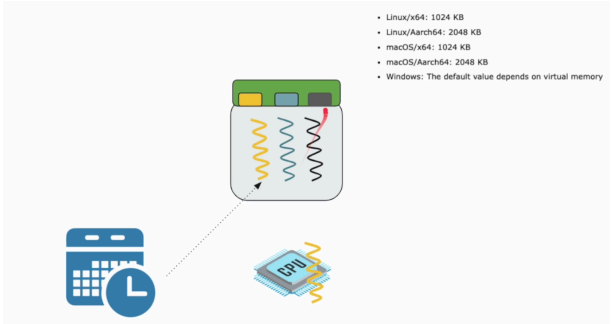
```
void method1(){
  int a=5;
  method2();
  ...
  ...
}

void method2(){
  int b=3;
  method3();
  ...
  ...
}

void method3(){
  ...
}
```

- So we have method 1 that calls method 2, method 2 calls method 3 and so on.
- When a Java thread is executing these methods, the OS schedular will keep switching threads.
- So the OS has to store the methods, local variables and these function call information somewhere. The place where this all is stored is called stack memory.
- So we have heap memory and stack memory

Heap vs Stack Memory

- Heap memory is where we store objects we dynamically create like ArrayList, Hashmap, etc.
- Where the stack memory will contain local variable, object references, function call information, etc
- 

- Each and every thread will have its own stack memory like the small box above the thread (in the picture).
- This size of the stack memory is determined when the process starts or a thread is created. This size has to be given upfront. It cannot be modified once the thread is created.
- By default, Java will assign 1 MB for every single thread. It can vary based on the CPU architectue and the OS for e.g. for the MacOS, it will be 2 MB.
- Whether the thread will do some work or not, we have to set aside some memory for every thread.

What is the **Problem**?

- CPU is very expensive. It costs a lot to have more CPU resources.
  - So we try to make use of CPU as much as we can!!
- Often times, in a microservices type architecture, we have too many network calls. But the network calls are slow.
- During this time, the threads will be idle. The CPU will not be utilized properly.
- So people create too many threads to make use of the CPU.
- But the problem is that thread is an expensive resource i.e. it is heavy and consumes a lot of memory.
- So we need a mechanism to make these network calls more efficient without wasting System resources.