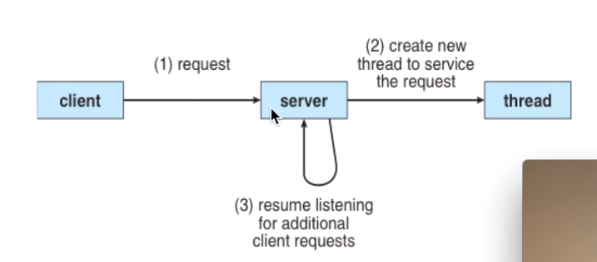
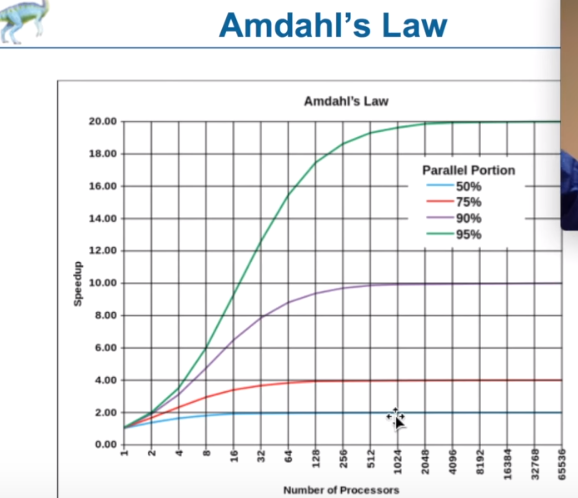
Mesela web sunucusunu dusunelim. Herkes ayni server’a baglaniyor ve server hepsine ayni anda cevap vermek zorundadir. Iste burda her baglanan kisi icin bir thread olusturursak ayni anda herkes o islemi gormus olur. Burda durup baktigimizda web server bizim processimizdir ve bunun threadleri olmustur. Yani thread olmasi icin process olmak zorundadir.

-

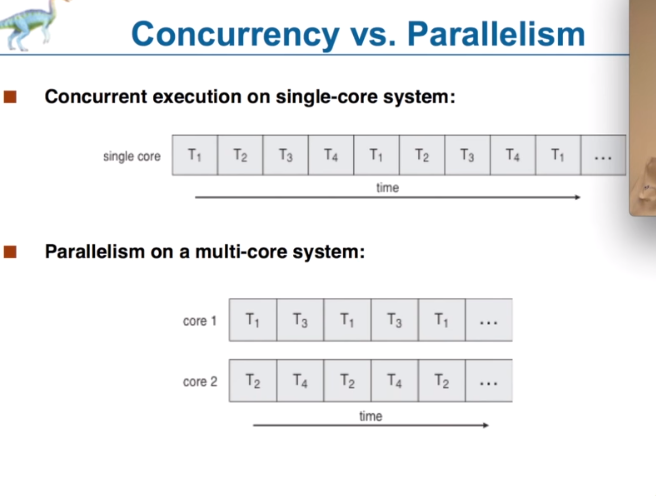


Process – Kol, Thread – Parmak

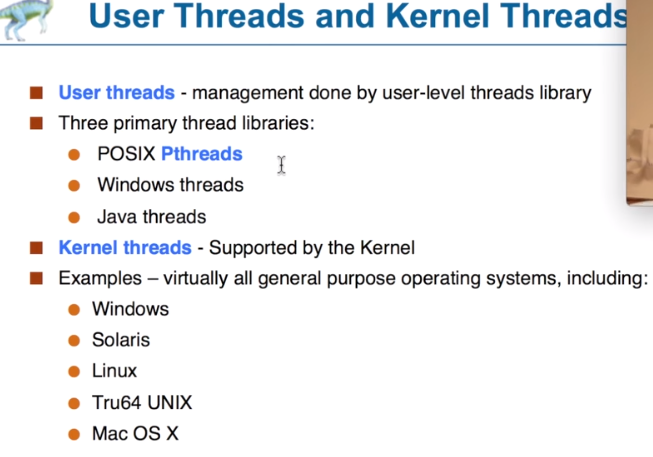
* MS-DOS = Tek process, tek thread
* Linux = Cok process, cok thread
* Java VM = Tek process, cok thread
* Processler sadece kendi uzaylarinda calisirlar. Sadece kendilerinin calistigini zannederler. Sanki tum RAM o processe ayrilmis gibi anlar fakat onun haricinde calisan processler de vardir ve hepsi kendini tek zanneder. Thread de ise processin isim uzayini kullanir ve bu isim uzayi paylasilir dolayisiyla bir thread in adresi degisirse hepsinin adresi degisir.
* Ekonomik olarak thread daha mantiklidir.



**Multicore, Multiprocessor** = Amac tamamen yapilacak isin parallel olarak baska yerlerde yapilmasi ve sonuca daha hizli bir sekilde ulasilmasi. Bir islem bittiginde oteki bitmediyse otekini beklemek zorundadir ve bu kotu bir durumdur bunu cozup paralelligi iyi bir sekilde saglamamiz gereklidir.



User Threads and Kernel Threads. Iki tip vardir.



Java – write once run everywhere.

Kernel threadler ve user threadler arasinda many-to-one, one-to-one, many-to-many gibi iliskiler vardir.