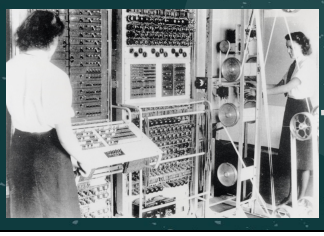
Sistemas Operativos I

Introducción

Objetivo de la clase de hoy: revisitar la historia y presentar la materia.

Historia Sistemas Operativos

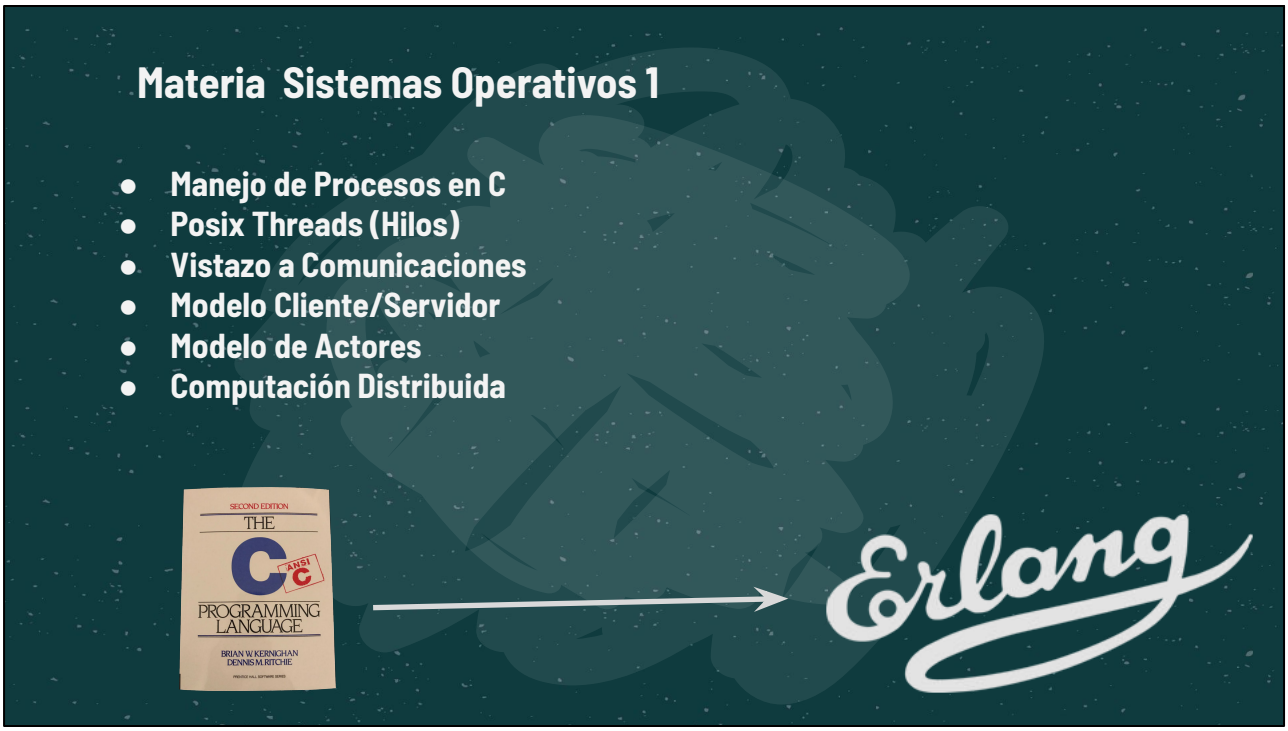
- 1962 : Canales
- A finales de los 60': Procesadores con múltiple unidades de procesamiento.
- 1960-1970: Región Crítica + Sincronización
- Finales 1970 y Ppio de 1980: Comunicaciones y redes
- 1980 - 1990 : Computación Distribuida
- Finales del siglo pasado: grandes cantidades unidades de procesamiento, modelo cliente/servidor, Internet y WWW!!!!



Veamos **una** historia de cómo presentar la programación concurrente y paralela.

La programación concurrente nace en 1962 con la invención de los denominados *canales*. Controladores independientes que permitían al CPU ejecutar un programa mientras que al mismo tiempo podía interpretar operaciones de comunicación con el mundo exterior (IO) dedicados a la espera de otro **programa suspendido**. Es decir, se podía tener la idea de una ejecución *al mismo tiempo* de dos procesos. Es decir, esto nace de la interacción del CPU con el mundo exterior, donde pasamos de tener procesos totalmente cerrados a poder interactuar con el CPU.
Ya que el Sistema operativo es el programa encargado de establecer una interfaz entre el hardware y el usuario, la programación concurrente se presenta como una problematica de los sistemas operativos. Aunque como ya verán en otras materias (PSSS: EDyAII) hoy en día ya es un campo en sí mismo.
A finales de 1960 ya se producen los primeros procesadores con múltiples unidades de procesamiento cumpliendo por fin la promesa de tener varios procesos ejecutándose al mismo tiempo, dando lugar a lo que se conoce como computación paralela.
El primer problema que se encuentra entonces es lo que se conoce como *el problema de la región crítica*. Ahora se tienen varios procesos compartiendo su tiempo de vida, y nace la pregunta de cómo compartir los recursos disponibles entre ellos? Por ejemplo, si hay una impresora, quién y cómo se tiene acceso a ella? Todos los procesos tienen el mismo nivel de precedencia?
Por ejemplo: imaginemos una computadora con una sóla unidad de procesamiento, por lo que todos los procesos comparten dicha unidad. Ahora es posible que

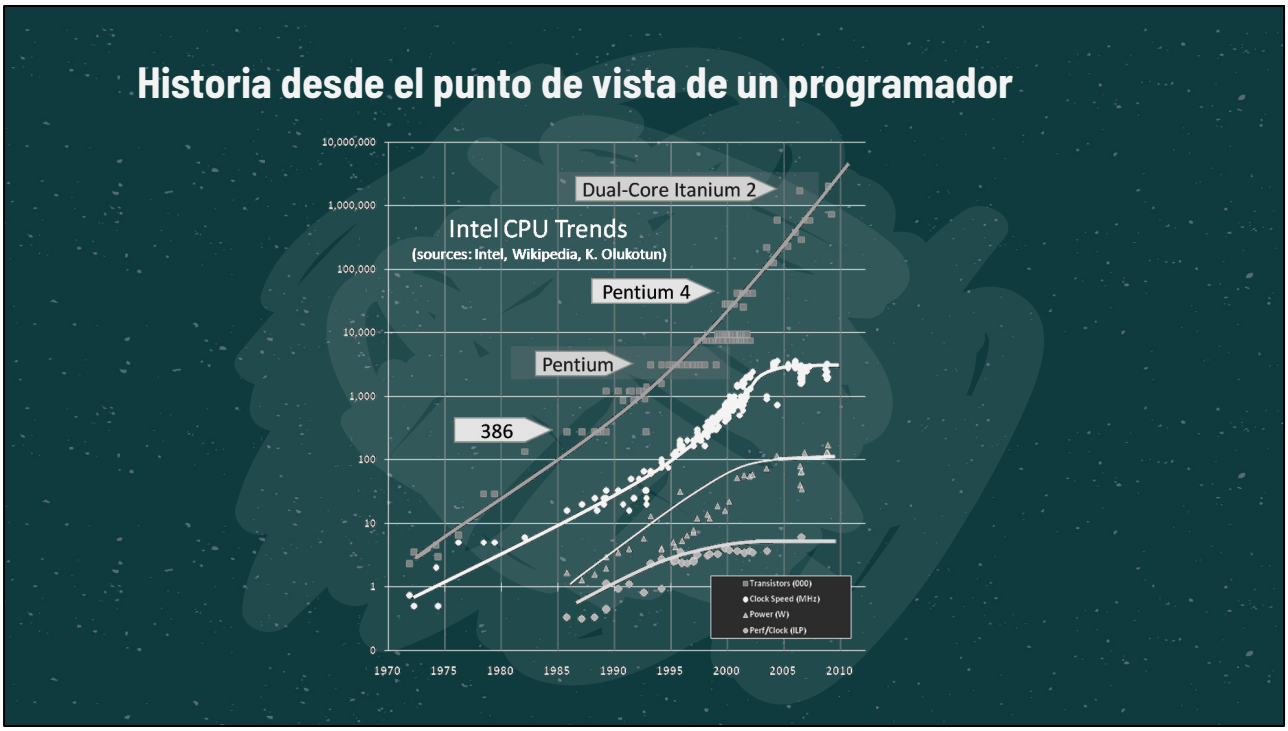
queramos que los procesos dedicados al sistema operativos tengan mayor privilegio. Se introduce entonces el concepto de sincronización: podemos organizar los procesos de manera tal que todos accedan a los recursos de una manera *eficiente?*. Y para esto se diseñaron distintos tipos de mecanismos como ser: semáforos, barreras, monitores, etc.
A finales de 1970 y principio de 1980 el foco se mueve al mundo de las comunicaciones y las redes, con proyectos como **Arpanet** y **Ethernet.** Tener varias computadoras conectadas en red da lugar a una nueva forma de computación denominada *computación distribuida*, donde obliga a repensar el modelo de computación como diferentes actores enviando mensajes en vez de leer y escribir en segmentos de memoria compartida (como lo era hasta ese momento).
Entre 1980 y 1990 el foco de estudio se centró en el estudio de sistemas distribuidos. Llegando entonces a finales del siglo pasado pasado donde se tienen ya grandes cantidades de unidades de procesamiento dedicadas a resolver problemas concretos, **INTERNET, WWW** y el modelo cliente/servidor. Como además el ingreso de las computadoras a la vida cotidiana de las personas.



Desde la materia seguiremos el mismo recorrido que la historia nos marcó.
Veremos (al menos) dos lenguajes de programación, partiendo la materia la mitad.
Primero utilizaremos el lenguaje de programación C donde partiremos desde lo más cerca posible del Sistema Operativo: llamada a sistema y creación de procesos. Y su vez comenzaremos a estudiar los problemas de región crítica (como memoria compartida) y sincronización.
Luego pasaremos a estudiar el concepto de Hilos utilizando el estándar de Posix Threads como alternativa a la creación de procesos.
Donde concluimos la etapa de C estudiando comunicación entre procesos, visitando un poco el área de comunicaciones y presentaremos el modelo cliente servidor.

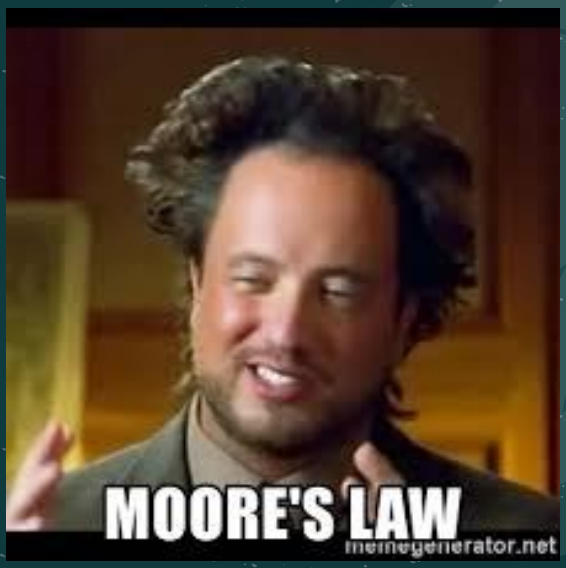
Luego pasaremos a estudiar otro lenguajes, probablemente Erlang, que se basa en la idea que la computación se produce mediante la comunicación entre los diferentes procesos. Esto nos lleva a repensar la forma en la estructuramos los programas, y nos vamos a divertir con eso.

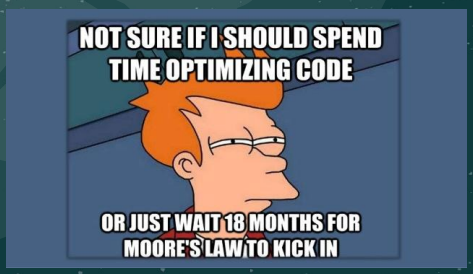
En otras palabras, partiremos desde los conceptos más crudos e iremos creciendo en abstracción.



La realidad desde el punto de vista del programador la historia fue distinta.
Recién los procesadores de múltiples cores llegaron a principios de los 2000.

Historia desde el punto de vista de un programador



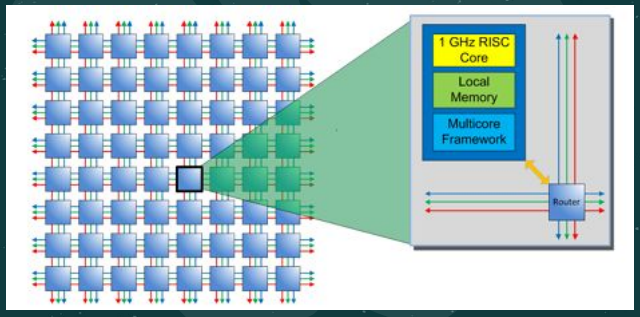


Durante todo ese periodo estaba instaurada lo que se conoce como Ley de Moore que dice a grandes rasgos que el procesamiento y memoria ram se duplica a razón de dos años.
Moore se basó en una técnica presentada por Dennard en 1974, que describe básicamente como duplicar la cantidad de transistores de un procesor a costa de aumentar un poco el consumo energetico.
Entonces para que voy a adaptar mi código usando programación concurrente/paralela/distribuida si simplemente mi código mágicamente va a andar más rápido, **exactamente el mismo código**, cuando salgan la nueva generación de CPUs.

Historia desde el punto de vista de un programador

Bueno lamentablemente esa historia terminó trágicamente, no es gratis duplicar la cantidad de transistores en (casi) el mismo espacio físico. Y aquí es donde viene la física a darnos una lección: ningún proceso es perfecto.
Siempre hay algo que energía que se pierde, y en éste caso, en forma de **calor**.
A finales del 2005, Intel ya abandona la producción de procesadores de un sólo núcleo.

Procesadores con varios núcleos



La solución del hardware es entonces pasar a tener múltiples unidades de procesamiento en el mismo procesador, permitiendo ejecutar múltiples procesos en exactamente el mismo tiempo. **Obligando** entonces a los programadores y la industria del software a repensar el software de forma Paralela. Esto es porque dos núcleos de 3Ghz no es lo mismo que uno sólo de 6Ghz.
La programación paralela comparte algunas problemáticas de la programación concurrente pero no es exactamente lo mismo.
Dentro de la materia no vamos a explorar extensamente el desarrollo de programas paralelos, eso lo verán con más detalle en la materia de EDyA2.