# 1. Concurrencia

Concurrencia cuando hay una existencia simultánea de múltiples procesos/hilos en ejecución.

## 1.1. La Abstracción

Un thread es una secuencia de ejecución atómica que representa una tarea planificable de ejecución.

- **Secuencia de ejecución atómica:** Cada thread ejecuta una secuencia de instrucciones como lo hace un bloque de código en el modelo de programación secuencial.
- **Tarea planificable de ejecución:** El sistema operativo tiene injerencia sobre el mismo en cualquier momento y puede ejecutarlo, suspenderlo y continuarlo cuando él desee.

### Threads vs procesos

**Proceso:** un programa en ejecución con derechos restringidos.
**thread:** una secuencia independiente de instrucciones ejecutándose dentro de un programa.

### Thread Scheduler

El Thread Scheduler (planificador de hilos) es una parte fundamental del sistema operativo encargada de administrar y controlar la ejecución de los hilos de ejecución en un entorno multitarea. Su función principal es asignar el tiempo de CPU disponible a los diferentes hilos de manera equitativa y eficiente, maximizando el rendimiento del sistema.

El Thread Scheduler toma decisiones sobre qué hilo debe ejecutarse en un momento dado, determinando el orden y la duración de la ejecución de los hilos en función de ciertas políticas de planificación.
En la actualidad hay dos formas de que los threads se relacionen entre sí:

- **Multi-threading Cooperativo:** Los threads se comunican entre sí y cooperan para realizar una tarea. No hay interrupción a menos que se solicite.
- **Multi-threading Preemptivo:** Consiste en que un threads en estado de running puede ser movido en cualquier momento.

## 1.2. Estructura y Ciclo de Vida de un Thread

El S.O. provee la ilusión de que cada uno de estos threads se ejecutan en su propio procesador, haciendo de forma transparente que se ejecuten o paren su ejecución.

Para que la ilusión sea creíble, el sistema operativo debe guardar y cargar el **estado** de cada thread. Como cualquier thread puede correr en el procesador o en el kernel, también debe haber estados compartidos, que no deberían cambiar entre los modos.

Para poder entender la abstracción hay que comprender que existen dos estados:

- El estado per thread.
- El estado compartido entre varios threads.

### 1.2.1. El Estado Per-thread y Threads Control Block (TCB)

Thread Control Block (TCB) estructura que representa el estado de un thread. La TCB almacena el estado per-thread de un thread:
Para poder crear múltiples threads y pararlos y rearrancarlos, el S.O. debe poder almacenar en la TCB el estado actual del bloque de ejecución:

- El puntero al stack del thread.
- Una copia de sus registros en el procesador.

### 1.2.2. Metadata referente al thread que es utilizada para su administración

## 1.3. Sincronización

La sincronización se refiere a la coordinación y control de la ejecución de múltiples hilos (Thread) para garantizar que se realicen correctamente las operaciones compartidas y evitar problemas como las condiciones de carrera y la inconsistencia de los datos.

### 1.3.1. Race Conditions

Una race condition se da cuando el resultado de un programa depende en como se intercalaron las operaciones de los threads que se ejecutan dentro de ese proceso.

Una Race Condition (condición de carrera) es un fenómeno indeseable que ocurre cuando dos o más hilos o procesos acceden y manipulan simultáneamente un recurso compartido sin una sincronización adecuada. El resultado de una Race Condition es impredecible y puede llevar a resultados incorrectos o inesperados en el programa.

Las Race Conditions ocurren cuando la ejecución de múltiples hilos o procesos no está coordinada correctamente y depende del orden o tiempo exacto

en el que se realicen las operaciones. Esto puede suceder cuando los hilos comparten datos y realizan operaciones de lectura y escritura en esos datos sin una protección adecuada.

Por ejemplo, considera dos hilos que comparten una variable numérica y realizan la siguiente secuencia de operaciones:

- Hilo 1: Lee el valor actual de la variable.
- Hilo 2: Lee el valor actual de la variable.
- Hilo 1: Incrementa el valor leído en 1.
- Hilo 2: Incrementa el valor leído en 1.
- Hilo 1: Escribe el nuevo valor en la variable.
- Hilo 2: Escribe el nuevo valor en la variable.

Si ambos hilos ejecutan estas operaciones de manera simultánea y sin sincronización, puede ocurrir una Race Condition. Dependiendo del orden de ejecución de los hilos y de las operaciones individuales, el resultado final puede ser inconsistente y no determinista. Por ejemplo, el resultado podría ser que ambos hilos incrementen el valor en 1 y sobrescriban los cambios del otro, o podrían basarse en valores desactualizados, entre otras posibilidades.

Las Race Conditions pueden ser difíciles de detectar y depurar, ya que su comportamiento puede ser intermitente y depender de factores como la velocidad relativa de los hilos, la planificación del sistema operativo y otros eventos concurrentes. Para evitar las Race Conditions, se utilizan mecanismos de sincronización, como bloqueos, semáforos, mutex, entre otros, que aseguran el acceso exclusivo a los recursos compartidos y garantizan la consistencia y la integridad de los datos.

## 1.4. Operaciones Atómicas

No pueden dividirse en otras y se garantiza la ejecución de la misma sin tener que intercalar ejecución.

El "Problema de la Heladera Llena" es un escenario abstracto que ilustra un desafío común en la programación concurrente, que consiste en coordinar el acceso a un recurso compartido entre múltiples hilos o procesos.

**Heisenbug** es un error no determinístico, el nombre se puso en honor al físico Heisenberg. Este tipo de error desaparece cuando uno quiere debuggearlo ya que depende de condiciones como el del intercalado del scheduler.

El término **interleaves** se utiliza en el contexto de la programación concurrente para referirse a la forma en que las instrucciones o acciones de múltiples hilos o procesos se entrelazan o se ejecutan en un orden no determinista.

## 1.5. Una Mejor Solución Locks

Un forma menos compleja de alcanzar una solución para el problema de la heladera es mediante la utilización de locks. Un **lock** es una variable que permite la sincronización mediante la exclusión mutua, cuando un thread tiene el candado o lock ningún otro puede tenerlo.

La idea principal es que un proceso asocia un lock a determinados estados o partes de código y requiere que el thread posea el lock para entrar en ese estado. Con esto se logra que sólo un thread acceda a un recurso compartido a la vez.

Esto permite la exclusión mutua, todo lo que se ejecuta en la región de código en la cual un thread tiene un lock, garantiza la atomicidad de las operaciones.

## 1.6. Errores comunes de concurrencia

En concurrencia el concepto de **dead lock** aparece cuando entre dos o más threads uno obtiene el lock y por algún motivo nunca libera el mismo haciendo que sus compañeros se bloqueen. Según Dahlin un deadlock es un ciclo de espera a través de un conjunto de threads, en el cual cada thread espera que algún otro thread en el ciclo tome alguna acción.