

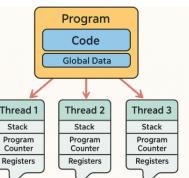
# Apuntes

## Concurrencia e Hilos

### Qué es un hilo?

Un **hilo** se puede ver como un camino de ejecución dentro de un programa. Un mismo programa puede tener varios hilos activos al mismo tiempo, y todos comparten su memoria principal (como si trabajaran en la misma oficina). Sin embargo, cada hilo tiene su propio espacio de trabajo privado:

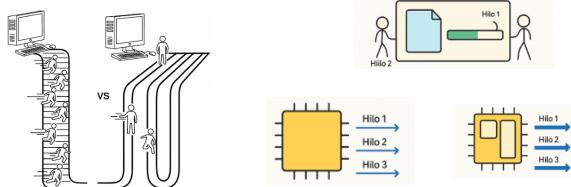
- su propia oficina (pila o stack)
- su propias actividades (contador de instrucciones)
- sus propios registros



### Función de los hilos

Usar varios hilos permite que un programa haga muchas cosas "al mismo tiempo". Por ejemplo:

- Mientras un hilo se encarga de descargar un archivo, otro hilo muestra el progreso de descarga en pantalla.
- Mientras un procesador solo tiene un núcleo, los hilos se ejecutan uno tras otro muy rápido (concurrencia), dando la sensación de simultaneidad.
- si el procesador es multinúcleo, los hilos sí pueden ejecutarse exactamente al mismo tiempo (parallelismo)



### Concurrencia vs Paralelismo

#### Concurrencia:

Un solo núcleo atiende varias tareas intercalándolas rápidamente. Es como un chef que prepara varias recetas a la vez, moviéndose rápido de una a otra.

#### Paralelismo:

Varias tareas realmente se ejecutan a la vez, porque hay más de un núcleo trabajando. Es como tener varios chefs, cada uno preparando una receta diferente al mismo tiempo



### Problemas al usar hilos

Cuando varios hilos modifican la misma información a la vez, pueden surgir errores impredecibles. Esto se llama **condición de carrera**, porque el resultado depende de cuál hilo "gane la carrera" y escriba primero.

**Sección crítica:** es la parte del código donde se accede a datos compartidos. Para evitar errores, se necesita que solo un hilo entre a la vez.



### Variables de condición

Es un mecanismo de sincronización que permite que un hilo se bloquee (espere) hasta que otra condición específica ocurra y otro hilo lo notifique para que continúe.

#### Por qué se usan

si tengo un hilo **Productor** que coloca datos en un buffer y un hilo **Consumidor** que saca datos de ese buffer.

si el consumidor intenta sacar un dato y el buffer está vacío, no puede continuar, entonces:

1. El consumidor **espera (wait)** en la variable de condición.
2. Cuando el productor coloca un nuevo dato hace un **signal()** para despertar al consumidor y que continúe su tarea.



### Locks

Un lock funciona como un candado. Si un hilo obtiene el lock, entra a la sección crítica y los demás deben esperar.

```

// Función para inicializar el Lock
void initLock() {
    pthread_mutex_init(&lock, NULL);
}

// Función para obtener el Lock (acquire)
void obtenerLock() {
    pthread_mutex_lock(&lock);
}

// Función para soltar el Lock (release/unlock)
void soltarLock() {
    pthread_mutex_unlock(&lock);
}
  
```



### Semáforos

Funciona como un contador que dice cuántos hilos pueden usar un recurso al mismo tiempo.

Dos funciones básicas:

**wait()** -> baja el contador en 1  
Si el contador era 0, el hilo espera.

**signal()** -> sube el contador en 1  
Si había hilos esperando, despierta uno

#### Tipos de semáforos:

- **Binario (0 o 1)**  
solo deja entrar a un hilo a la vez
- **Contador general (>1)**

Permite que entren varios hilos al mismo tiempo, hasta el número del contador.  
Ejemplo: si el contador es 3, solo 3 hilos pueden entrar



### Problemas Clásicos

#### • Proveedor- Consumidor

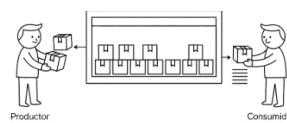
Un productor crea ítems y los pone en un buffer; un consumidor los retira. Necesitan sincronizarse para no sacar o poner ítems cuando el buffer está lleno o vacío.

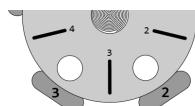
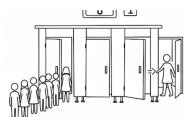
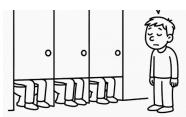
#### • Lector - Escritores

Lectores pueden leer al mismo tiempo, pero cuando un escritor quiera modificar, debe estar solo.

#### • Filósofos comensales

Filosofos comparten palillos para comer. Si todos toman un palillo al mismo tiempo, pueden quedar esperando eternamente (deadlock)





JONATHAN ANDRÉS GRANDA ORREGO  
[jonathan.granda@udea.edu.co](mailto:jonathan.granda@udea.edu.co)

JHON ALEXANDER BOTERO GÓMEZ  
[jhon.botero1@udea.edu.co](mailto:jhon.botero1@udea.edu.co)