

Procesos

Programa Vs Proceso

- Un programa es un conjunto de sentencias escritas en un lenguaje de programación con un objetivo específico.
- Un proceso es un programa en ejecución mas su contexto.

Programa Vs Proceso

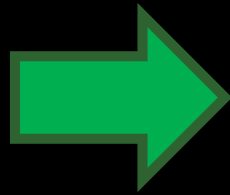
Programa

- Estático
- En disco
- No necesita otro recurso

Proceso

- Dinámico
- En memoria
- Utiliza CPU, E/S, memoria, archivos, etc.

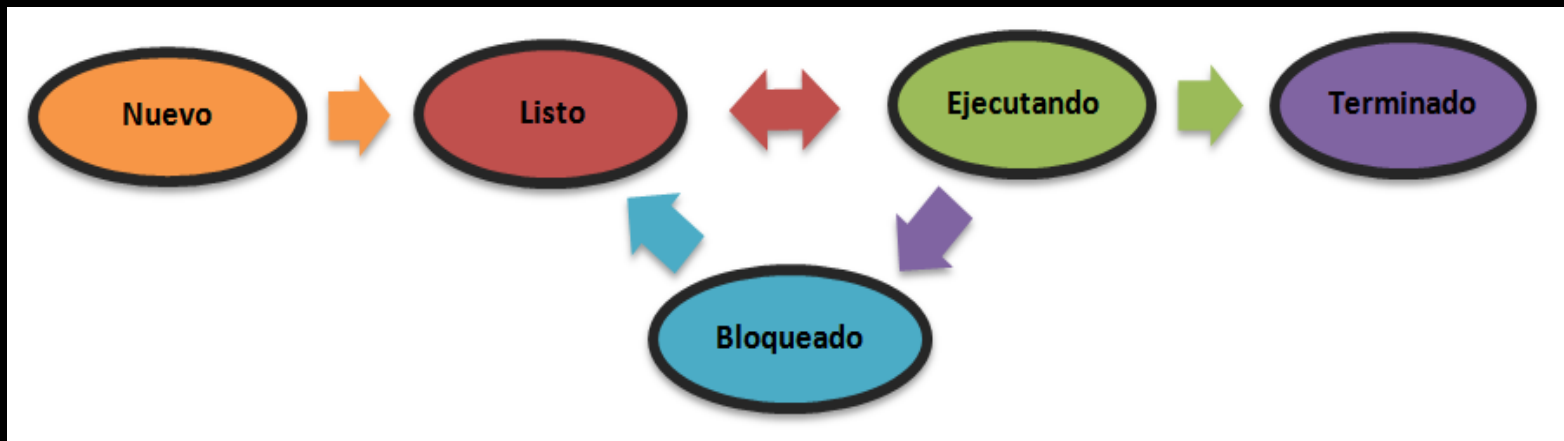
Programa Vs Proceso



Aplicaciones	Procesos	Servicios	Rendimiento	Funciones de red	Usuarios
Nombre de imagen	Nomb...	CPU	Espacio de tr...	Espacio máxi...	
AvastSvc.exe *32	SYSTEM	00	66.228 KB	213.072 KB	
AvastUI.exe *32	master	00	5.668 KB	9.652 KB	
brave.exe	master	00	93.232 KB	102.812 KB	
brave.exe	master	00	53.932 KB	54.088 KB	
brave.exe	master	00	43.492 KB	46.192 KB	
brave.exe	master	00	53.304 KB	53.448 KB	
brave.exe	master	00	89.636 KB	120.228 KB	
brave.exe	master	00	29.644 KB	93.268 KB	
brave.exe	master	00	60.756 KB	62.288 KB	
brave.exe	master	00	156.340 KB	226.968 KB	
brave.exe	master	00	25.488 KB	39.896 KB	
brave.exe	master	00	19.124 KB	19.188 KB	
brave.exe	master	00	33.456 KB	33.684 KB	
brave.exe	master	00	55.628 KB	57.000 KB	
brave.exe	master	00	199.236 KB	222.620 KB	

Ciclo de vida

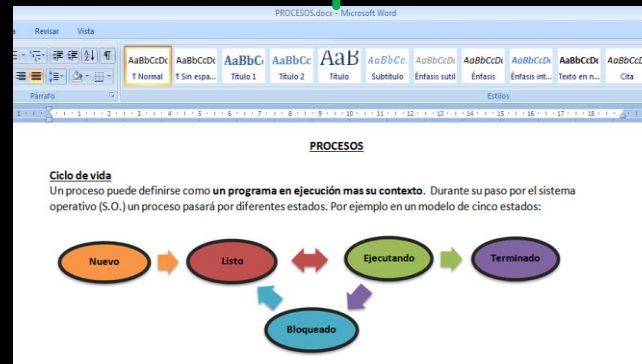
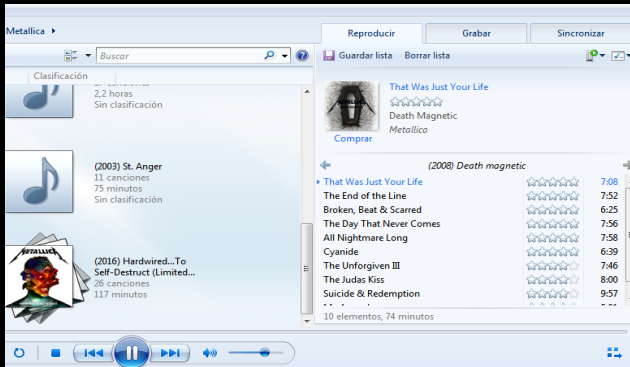
- Durante su paso por el sistema un proceso pasa por diferentes estados.



Ciclo de vida

Nuevo	El proceso está siendo creado.
Listo	El proceso se encuentra en condiciones de competir por el procesador.
Ejecutando	El proceso está utilizando el procesador.
Bloqueado	El proceso espera la ocurrencia de algún evento.
Terminado	El proceso finalizó su ejecución, solo queda que el S.O. libere sus recursos.

Concurrencia



Concurrencia

- Debido a la diferencia de tiempos que existe entre una computadora y el ser humano, ejecutar procesos de manera concurrente sobre un único procesador, genera la **sensación** de paralelismo.

Cambio de proceso

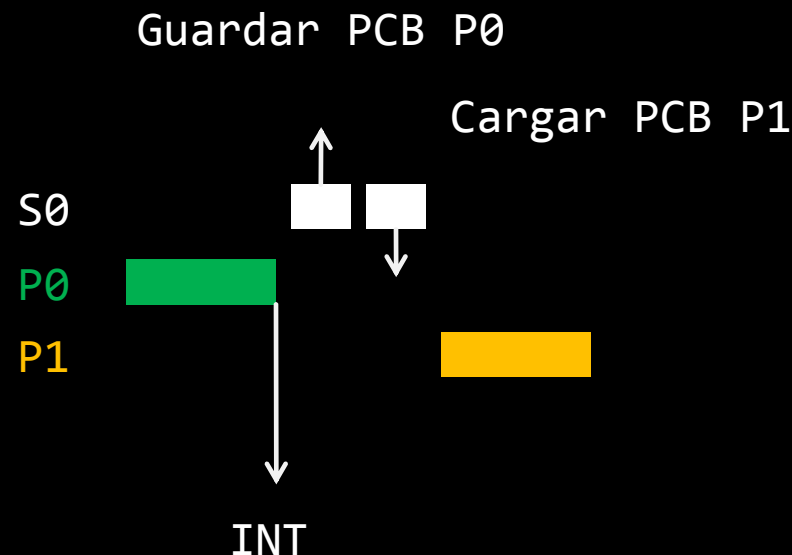
- Para poder ejecutar procesos de manera concurrente, es necesario realizar un **cambio de proceso**, es decir intercalar procesos en el tiempo.

Cambio de contexto

- Al llevar a cabo un cambio de proceso, es fundamental guardar el contexto del proceso en ejecución y cargar el contexto del proceso a ejecutar.
- Es lo que conocemos como cambio de contexto (context switch)

Process Control Block (PCB)

- Estructura que almacena el contexto de ejecución de cada proceso en el sistema.



Información del PCB

■ En líneas generales almacena:

- Identificador del proceso (PID).
- PIDs de procesos emparentados, padre, hijos.
- Estado (Nuevo, Listo, etc.).
- Contador de programa.
- Registros del procesador.
- Información para la planificación (prioridad por ej.).
- Información para la administración de memoria (MPT, SMT, etc.)
- Información de Entrada/Salida (archivos por ejemplo).
- Información para las estadísticas (Uso de CPU por ejemplo).

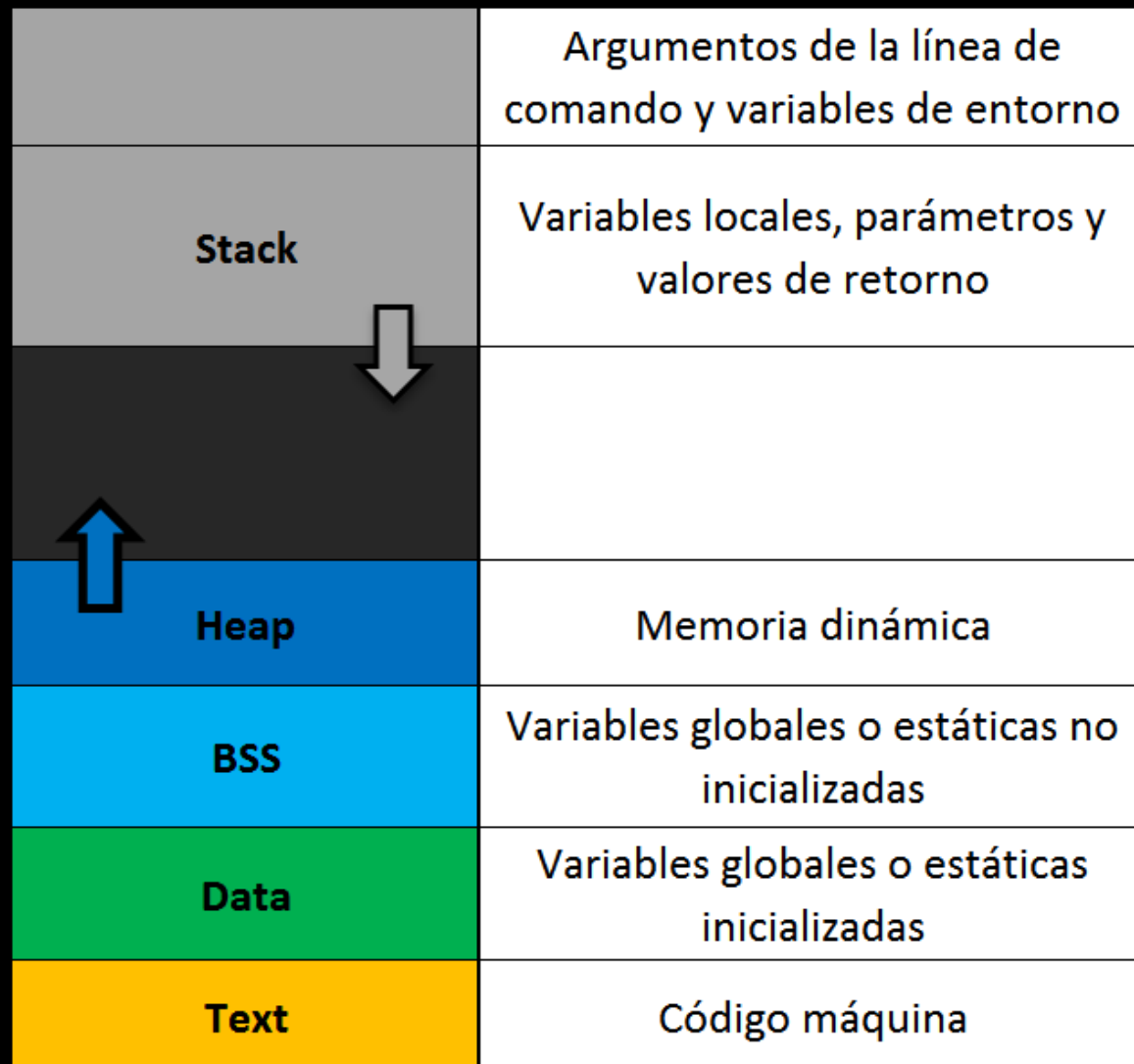
PCB en linux

- Estructura task_struct contenida en sched.h

```
1  #define TASK_RUNNING 0
2  #define TASK_INTERRUPTIBLE 1
3  #define TASK_UNINTERRUPTIBLE 2
4  #define TASK_STOPPED 4
5  #define TASK_TRACED 8
6
7  /*in tsk->exit_state */
8  #define EXIT_DEAD 16
9  #define EXIT_ZOMBIE 32
10 #define EXIT_TRACE ( EXIT_ZOMBIE | EXIT_DEAD )
```

<https://elixir.bootlin.com/linux/latest/source/include/linux/sched.h>

Memoria de un proceso



Memoria de un proceso

TEXT Es el binario ejecutable. Contiene las instrucciones del programa.

DATA Variables globales y estáticas inicializadas con valor distinto de cero.

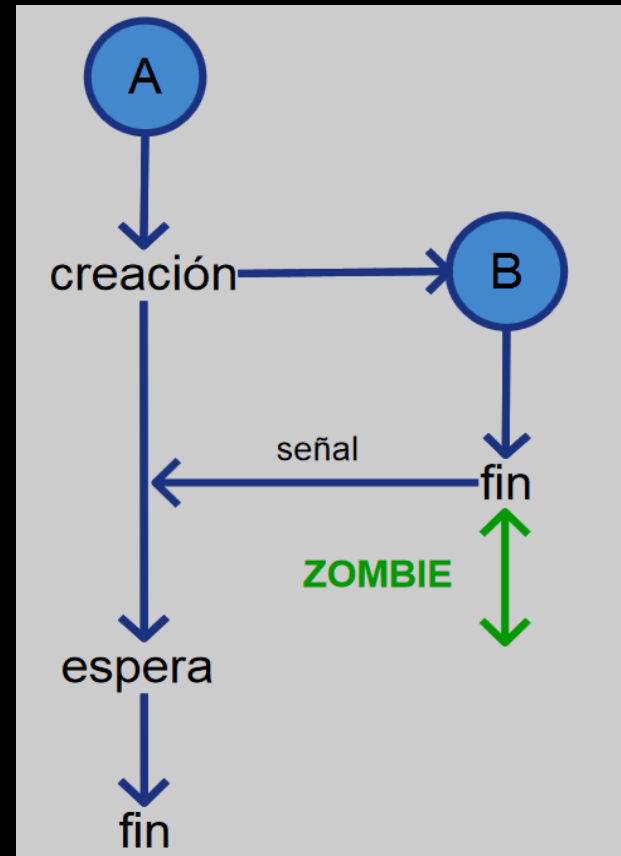
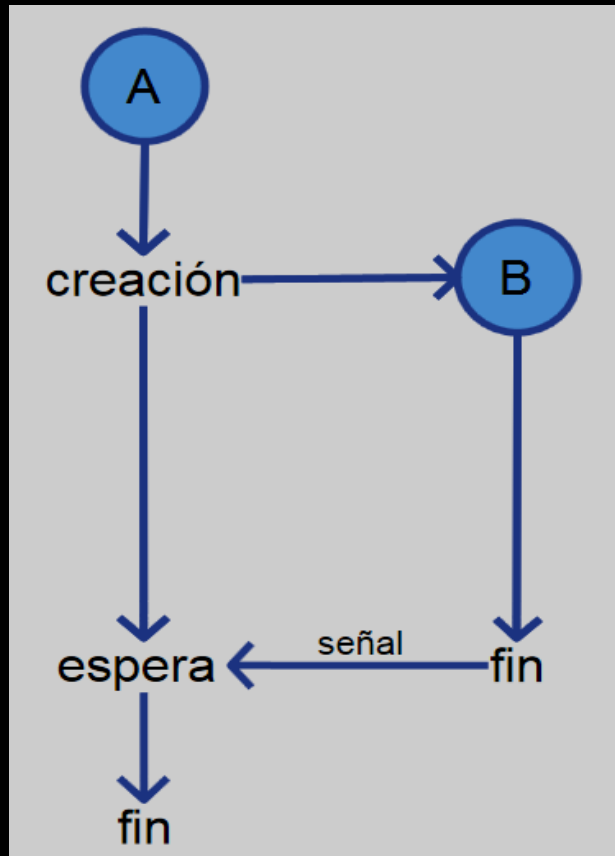
BSS (Block Started by Symbol)

Variables globales y estáticas no inicializadas ó inicializadas a cero.

STACK Por cada llamada a función se crea un bloque en la pila.

HEAP Memoria dinámica creada por el procesos malloc(), calloc(), etc.

Creación de procesos



fork()

- Duplica al proceso llamador (Padre) creando un nuevo proceso (hijo).

```
#include <unistd.h>  
pid_t fork(void);
```

fork()

■ Retorno

- ⦿ En caso de error de creación: -1
- ⦿ En caso de creación exitosa:
 - ⦿ Al padre: el PID del hijo.
 - ⦿ Al hijo: 0

fork()

- ⦿ El hijo hereda el contexto de ejecución del padre.
- ⦿ Cada proceso tiene su propia área de memoria.

Familia `wait()` – `sys/wait.h`

- `wait()`: Bloquea al llamador hasta recibir la señal de finalización de cualquier hijo.
- `waitpid()`: Bloquea hasta recibir la señal de finalización de un hijo en particular.
- `waitid()`: Similar a `waitpid()`, pero se puede especificar el tipo de ID (PID o ID de grupo) por el cual esperar.

wait()

● `pid_t wait( int *status )`

En `status` se almacena el estado de finalización del proceso hijo. Sobre esa variable se pueden ejecutar algunas macros, como por ejemplo:

`WIFEXITED` Retorna True si el hijo finalizó normalmente.

`WIFSIGNALED` Retorna True si el hijo finalizó a causa de una señal.

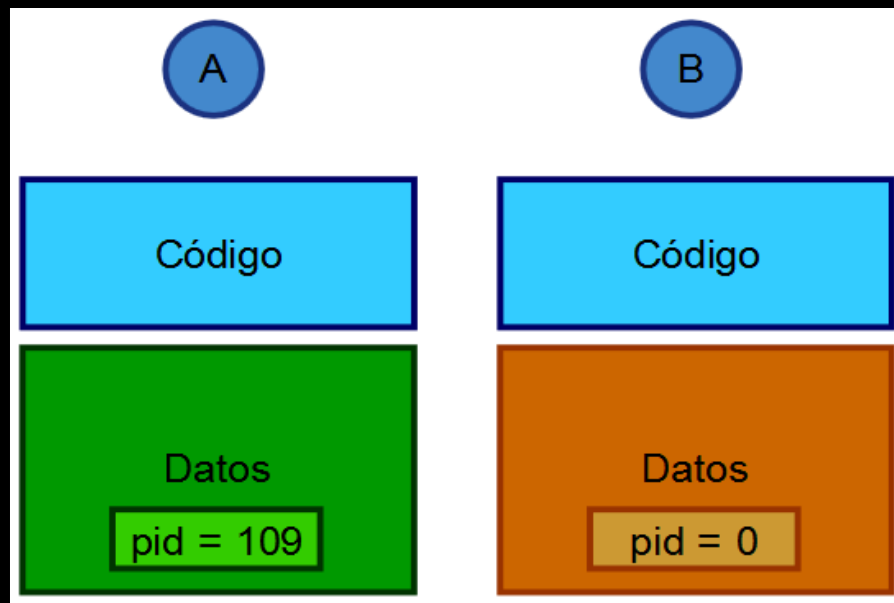
`WTERMSIG` Retorna el número de señal que causó la finalización del proceso hijo.

`WCOREDUMP` Retorna True si el hijo produjo un core dump.

Retorna el PID ó -1 en caso de error.

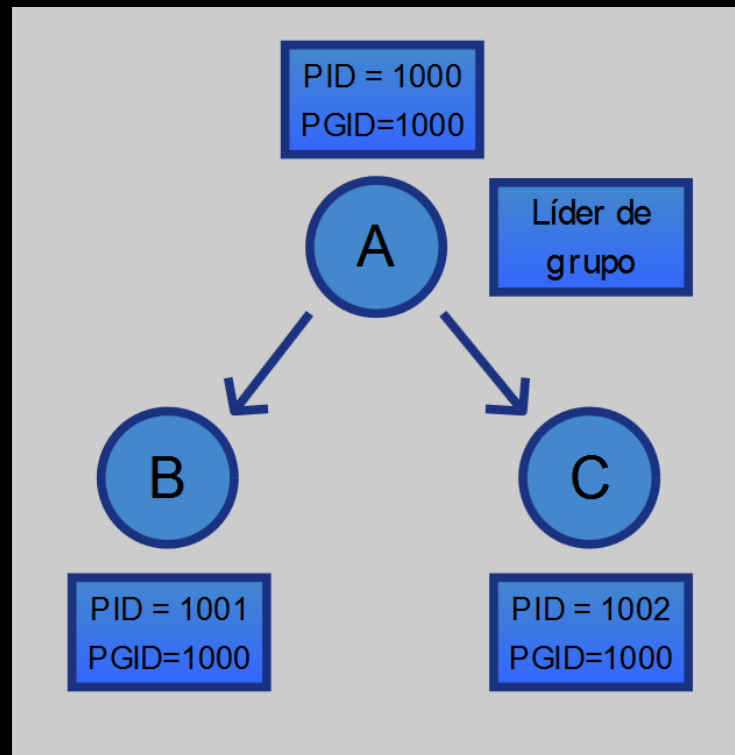
Memoria

⦿ Luego del fork()



PGID (Process Group ID)

- Un grupo de procesos comparte el mismo PGID. Por ejemplo:



Familia exec()

- Reemplaza la imagen (Text, Data, BSS y Stack) del proceso llamador con la imagen de otro proceso.

```
#include <unistd.h>
int execve(
    const char *file,
    char *const args[],
    char *const envp []
);
```

Métodos en Python

☉ Métodos relacionados:

```
os.fork( )  
os.wait( ) -> Retorna ( PID, estado )  
os.waitpid( PID, opciones )  
os._exit( estado )  
os.execv( ruta, argumentos )
```

☉ Links de referencia:

<https://docs.python.org/es/3.10/library/os.html>

<https://docs.python.org/3/library/exceptions.html>

Creación de procesos en JAVA

◎ `ProcessBuilder`

Establece los atributos del nuevo proceso a crear.

◎ `ProcessBuilder.start()`

Lleva a cabo la creación del nuevo proceso.

◎ `Process.waitFor()`

El proceso llamador, queda a la espera de la finalización del proceso sobre el que se ejecuta la llamada.

Creación de procesos en JAVA

◎ Links de referencia:

<https://docs.oracle.com/javase/7/docs/api/java/lang/ProcessBuilder.html>

<https://docs.oracle.com/javase/7/docs/api/java/lang/Process.html>

<https://docs.oracle.com/javase/7/docs/api/java/lang/Runtime.html>

<https://docs.oracle.com/javase/9/docs/api/java/lang/ProcessHandle.html>

Hilos (Threads)

Características

- Puede definirse como un proceso liviano.
- Comparte el código, la memoria, archivos, señales y semáforos con todos los hilos del mismo proceso al cual está asociado.
- Maneja su propio contador de programa y stack, además de un TCB (Thread Control Block).

Ciclo de vida

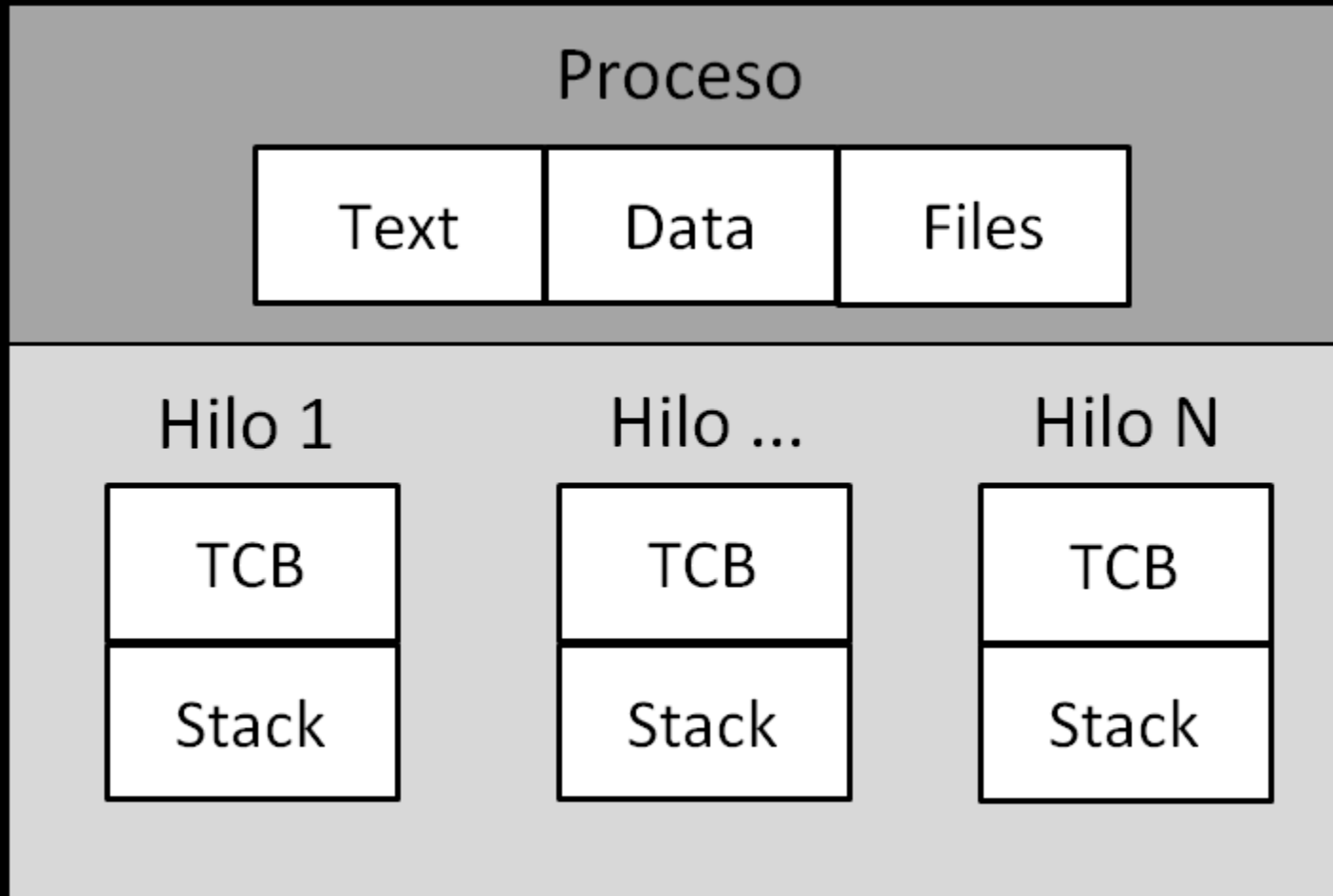
- En general se pueden definir los siguientes estados:



Thread Control Block (TCB)

- Identificador (TID).
- Stack Pointer: puntero al área de stack.
- Estado (Listo, bloqueado, ejecutando, terminado).
- Contador de programa.
- Registros del procesador.
- Puntero al PCB del proceso pesado al cual pertenece.

Memoria de los hilos



Ventajas sobre los procesos

- Menor tiempo de creación y finalización.
- El cambio de contexto entre hilos del mismo proceso es más rápido:
 - Menor información en el TCB.
 - Espacio de memoria compartido.

Implementación

- **ULT – User Level Thread**

La gestión de los hilos es realizada el propio **programa**. El kernel no interviene en la gestión de los hilos.

- **KLT – Kernel Level Thread**

La gestión es realizada por el **kernel**

ULT vs KLT

ULT	KLT
Implementados por el usuario.	Implementados por el Sistema Operativo. Creación, planificación, sincronización y gestión de bloqueos.
El S.O. no conoce su existencia. (Solo conoce al proceso pesado)	El S.O. conoce su existencia.
Menor tiempo en el cambio de contexto. (No involucra al S.O.)	Mayor tiempo en el cambio de contexto.
Ante un syscall bloqueante, se bloquea todo el proceso al cual pertenece el hilo.	Ante un syscall bloqueante, otro hilo puede pasar a ejecución.
Desaprovecha el multiprocesamiento.	Aprovecha el multiprocesamiento.
Multiplataforma.	Específico para un sistema operativo.

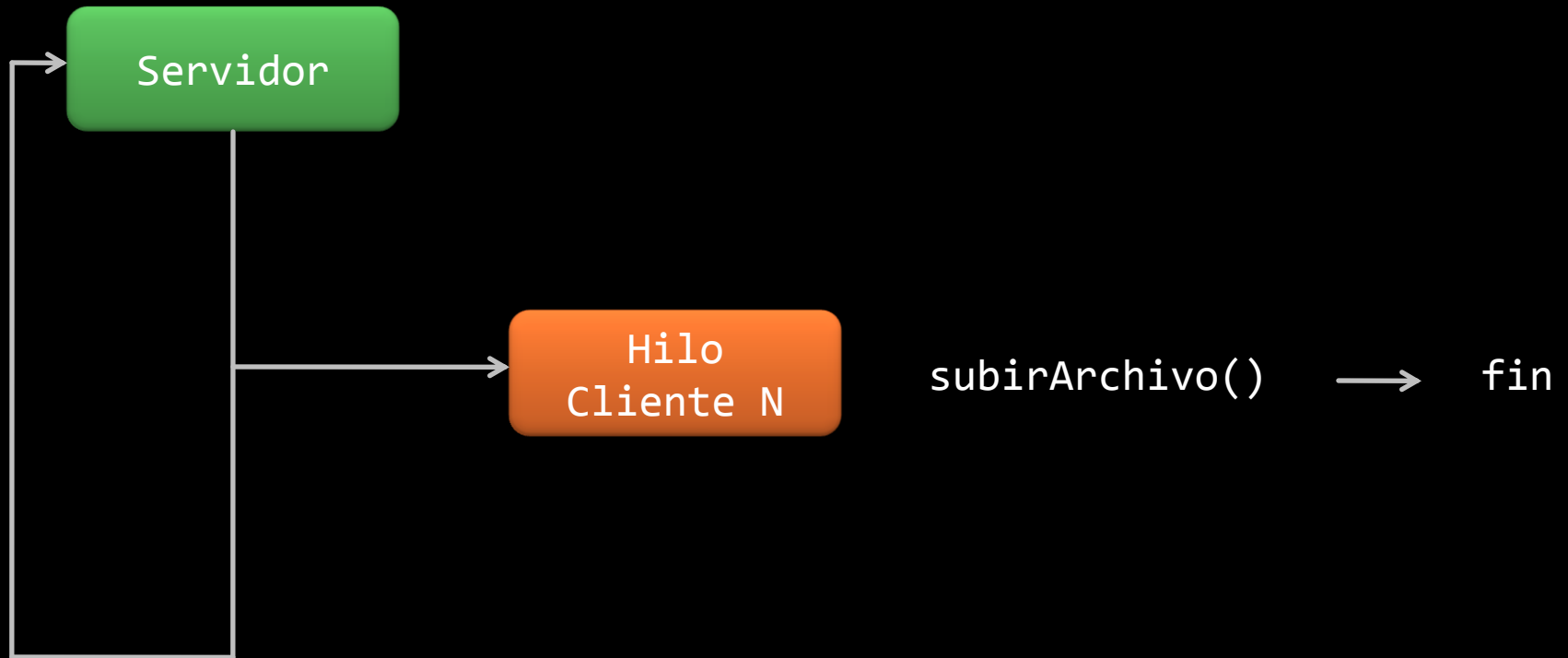
Patrones de trabajo

■ Jefe/Trabajador

- Un hilo (jefe) recopila o genera tareas a realizar y las delega en otros hilos (trabajadores).
- Las tareas a realizar **pueden ser distintas**.
- La cantidad de hilos puede ser variable en el tiempo.
- Los trabajadores pueden finalizar sin notificar al jefe (Independientes o detached).

Patrones de trabajo

■ Jefe/Trabajador

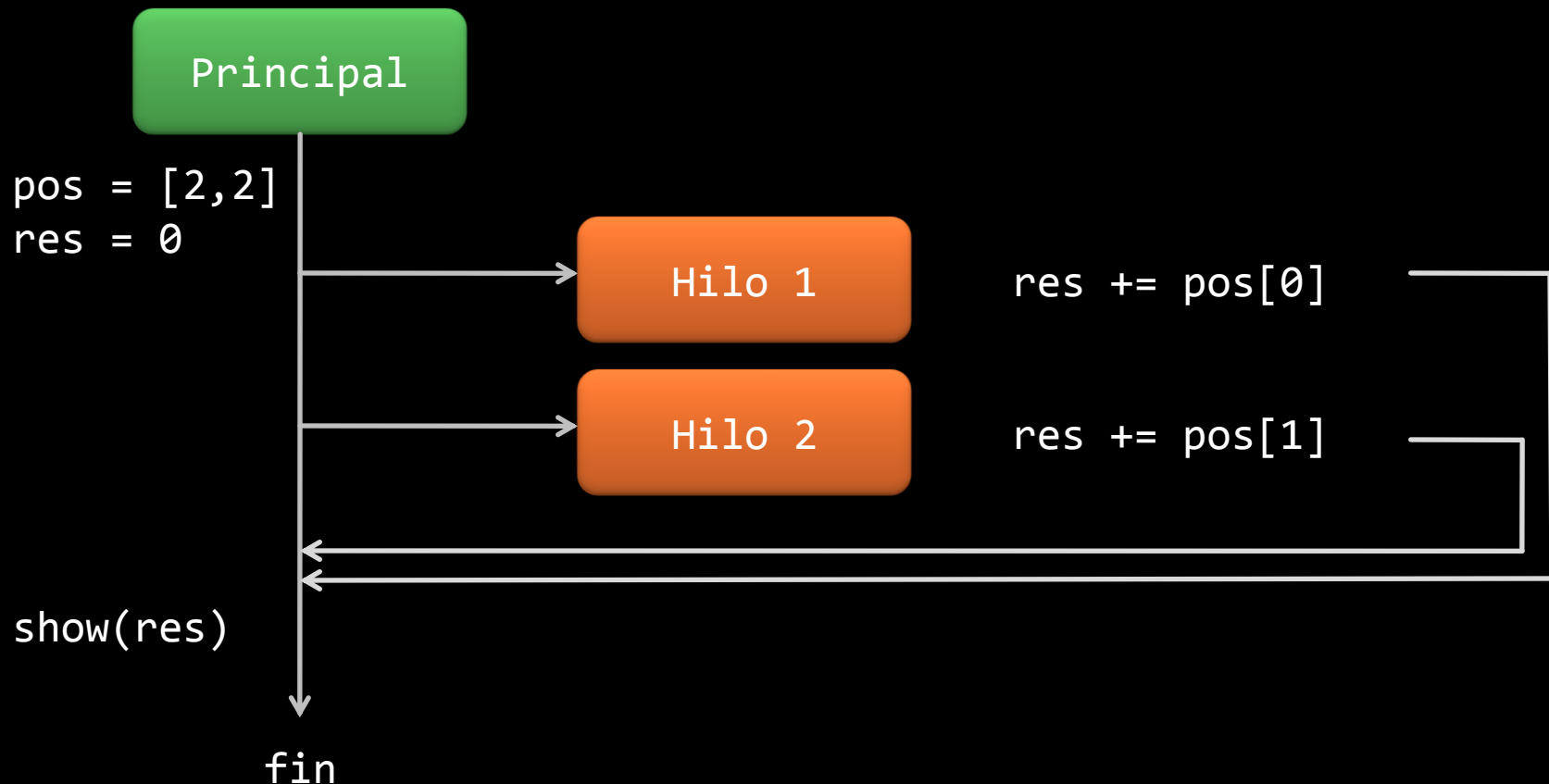


Patrones de trabajo

- Equipo de trabajo
- Se crean varios hilos que realizarán **el mismo trabajo** sobre distintos datos.
- La cantidad de hilos es fija una vez iniciado el procesamiento.
- Hay una espera de los hilos por parte del proceso principal, con lo cual dichos hilos no finalizan de inmediato (dependiente o joinable).

Patrones de trabajo

■ Equipo de trabajo



Creación de hilos

- Hilo dependiente (Joinable)
 - Necesita ser “joinado” por el hilo creador para poder liberar sus recursos.
- Hilo independiente (Detached)
 - NO necesita ser “joinado” por el hilo creador para poder liberar sus recursos.

Creación de hilos en C++

- Clase `std::thread` <thread> C++11
- *Constructor*
`thread(función, arg1, arg2, ..., argN )`
- Crea un objeto hilo joinable.
- Recibe la función que contiene el código del hilo y los argumentos de la misma si corresponde.

<https://cplusplus.com/reference/thread/thread/>

Creación de hilos en C++

- Clase `std::jthread` `<thread>` C++20
- Mismo comportamiento que `thread`, solo que al finalizar, el hilo se joina automáticamente.

<https://en.cppreference.com/w/cpp/thread/jthread>

Creación de hilos en C++

- Métodos

- *void detach()*

Convierte al objeto llamador en un hilo independiente (detached).

- *bool joinable()*

Retorna true si el objeto llamador es un hilo joinable.

Creación de hilos en C++

- Métodos

- *void join()*

Bloquea la ejecución del hilo llamador hasta que el hilo sobre el cual se ejecuta finalice.

Una vez realizado el join, el hilo se vuelve independiente.

Creación de hilos en Python

- Módulo `threading`
- `threading.Thread(`
 - `target` = `None`,
 - `name` = `None`,
 - `args` = `()`,
 - `kwargs` = `{}`,
 - `daemon` = `None``)`

<https://docs.python.org/es/3.8/library/threading.html>

Creación de hilos en Python

- **Argumentos**
- *Target*: La función que ejecutará el hilo.
- *Name*: Nombre del hilo, por defecto Thread-N.
- *Args*: Tupla con los argumentos para el target.
- *Kwargs*: Diccionario de argumentos.
- *Daemon*: si es joinable o detached.

Creación de hilos en Python

- Métodos

- *start()*

Invoca el método `run`, iniciando la actividad del hilo.

- *bool is_alive()*

Retorna `true` si el hilo fue iniciado y aún no finalizó.

- *Thread current_thread()*

Retorna el objeto hilo actual.

Creación de hilos en Python

- Métodos

- *int get_native_id()*

Retorna el TID asignado por el kernel.

- *int get_ident()*

Retorna un identificador único asignado por el intérprete. No tiene un significado como el TID por ejemplo.

Creación de hilos en Python

- Métodos

- *None join(timeout=None)*

El llamador se bloquea hasta que el hilo en cuestión finalice ó hasta que se cumpla el timeout (valor decimal que representan seg) en caso de no ser None.

En caso de join con timeout, seguido al join hay que llamar a `is_alive()` para saber si el hilo finalizó

Creación de hilos en Python

- `RuntimeError`

- *start*

Si se invoca mas de una vez.

- *join*

Si se intenta joinear el hilo actual.

Si se intenta joinear un hilo no iniciado.

- *Al setear daemon*

Si se realiza después del start.

Creación de hilos en Java

- **Class Thread**

- *Forma 1*

Heredar de la clase Thread y redefinir el método run()

- *Forma 2*

Implementar la interface runnable y el método run()

<https://docs.oracle.com/javase/8/docs/api/java/lang/Thread.html>

Creación de hilos en Java

- Algunos constructores
- `public Thread()`
- `public Thread(Runnable target)`
- `public Thread(String name)`
- `public Thread(Runnable target, String name)`
- ...

Creación de hilos en Java

- Algunos métodos

- `public static Thread currentThread()`
- `public long getId()`
- `public final String getName()`
- `public void start()`
- `public final boolean isAlive()`
- `public final void setDaemon(boolean)`
- `public final boolean isDaemon()`
- `public final void join()`

Creación de hilos en Java

● Links de referencia:

- <https://docs.oracle.com/javase/7/docs/api/java/lang/Thread.html>
- <https://docs.oracle.com/javase/7/docs/api/java/lang/Runnable.html>