U2 - Procesos e hilos

Índice:

Procesos

Introducción

> **Definición**
>
> Un **proceso** es la ejecución de un programa junto con su estado. Es la entidad a la que el sistema operativo asigna tiempo de CPU y otros recursos. Cada proceso tiene su propio espacio de direcciones y contexto de ejecución, lo que permite que varios programas o procesos coexistan en el sistema sin interferir entre sí.

Los procesos se cargan en el **stack**

Como varios procesos se pueden ejecutar a la vez ocurre lo que se denomina **Concurrencia**. La **concurrencia** ocurre cuando varios procesos parecen estar ejecutándose al mismo tiempo. En un sistema concurrente, los procesos no necesariamente se ejecutan simultáneamente, pero el sistema operativo alterna entre ellos tan rápidamente que da la impresión de que están ejecutándose al mismo tiempo. Esta concurrencia se puede presentar de dos formas

- El **paralelismo real** se da en sistemas con múltiples procesadores o núcleos de CPU, donde varios procesos pueden ejecutarse verdaderamente al mismo tiempo, cada uno en su propio procesador o núcleo. Esto aumenta el rendimiento y la capacidad del sistema para ejecutar tareas simultáneamente.
- El **pseudoparalelismo** es cuando un sistema con un solo procesador da la apariencia de que varios procesos están ejecutándose al mismo tiempo. Sin embargo, en realidad, el procesador alterna entre diferentes procesos muy rápidamente, ejecutando pequeñas porciones de cada uno (conmutación de procesos).

Conmutación de procesos
La **conmutación de procesos** o **cambio de contexto** ocurre cuando el sistema operativo pausa un proceso en ejecución y guarda su estado (contexto) para ejecutar otro proceso. Este cambio es necesario para aprovechar al máximo la CPU, especialmente cuando un proceso está inactivo (esperando una operación de E/S).

Creación de procesos

La creación de procesos es esencial para la maquina, las principales formas en las que se crean son las siguientes:

Situación	Descripción
Arranque del SO (primer/segundo plano)	El sistema operativo crea procesos base al arrancar. Los de primer plano son interactivos por el usuario, y los de segundo plano (servicios) ejecutan tareas de fondo y no son interactivos para el usuario.
Un proceso hace una llamada al sistema	Un proceso en ejecución puede crear otros procesos a través de llamadas al sistema como `fork()` (Unix/Linux) o `CreateProcess()` (Windows).
Un usuario dispara un proceso	Un usuario puede iniciar un proceso manualmente, ya sea a través de la línea de comandos o haciendo clic en un ícono en una interfaz gráfica.
Inicio de trabajo por lotes (jobs)	Procesos automáticos que se inician a intervalos específicos o en ciertos eventos, como con cron o tareas programadas en Windows.

Cada una de estas situaciones aprovecha los recursos del sistema operativo para gestionar la creación y el control de nuevos procesos en el sistema.

Terminación de procesos

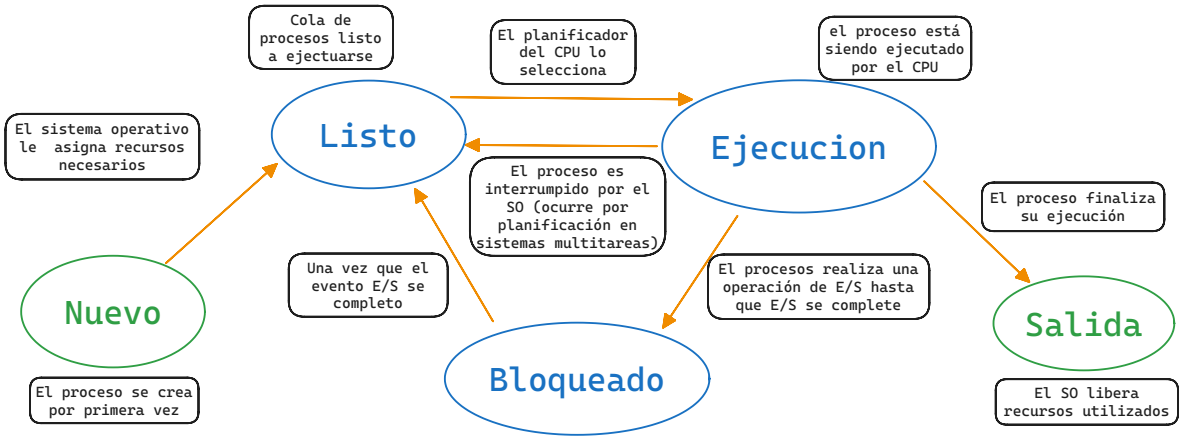
La **terminación de procesos** es la fase final en el ciclo de vida de un proceso. Puede ocurrir por diversas razones, tanto voluntarias como involuntarias. Los sistemas operativos gestionan la liberación de los recursos asignados al proceso y la limpieza de su estado cuando este finaliza.

Tipo de Terminación	Descripción
Normal (voluntario)	El proceso termina de forma voluntaria porque completó su tarea correctamente.
Error Programado (voluntario)	El proceso se termina voluntariamente al detectar un error que fue anticipado en su lógica de programación.
Error Fatal (involuntario)	El proceso es terminado de forma involuntaria por el sistema operativo debido a un error grave que no puede manejar.
Recibe una señal de kill (involuntario)	Otro proceso o el sistema operativo finaliza el proceso mediante una señal de terminación, sin que el proceso tenga control.

Estos mecanismos de terminación aseguran que los recursos sean liberados correctamente y que el sistema continúe funcionando sin problemas.

Estado de los procesos

El estado de los procesos es bastante importante, veamos el siguiente diagrama que nos ayudara a entenderlo



Jerarquía de procesos

La **jerarquía de procesos** es un concepto importante en los sistemas operativos, especialmente en aquellos basados en la norma POSIX (como Unix y Linux). La idea es que los procesos pueden tener relaciones de "paternidad" e "hijitud" que definen cómo son creados, gestionados, y eventualmente terminados

Cada proceso en un sistema operativo se identifica de manera única mediante un **PID** (**Process ID** o Identificador de Proceso).

Procesos padre e hijos

Un **proceso padre** es aquel que crea uno o más **procesos hijos** mediante llamadas al sistema, como `fork()` en sistemas POSIX. El padre y los hijos pueden compartir o no espacio de memoria, dependiendo del sistema y de cómo se haya implementado la creación del proceso.

Procesos Huérfanos

Un **proceso huérfano** es un proceso hijo cuyo proceso padre ha terminado o sido terminado antes que él. En sistemas POSIX, cuando un proceso padre termina, los procesos huérfanos son "adoptados" automáticamente por el proceso con **PID 1**, que normalmente es el proceso `init`

Procesos Zombies

Un **proceso zombie** es un proceso que ha terminado su ejecución, pero aún tiene una entrada en la tabla de procesos del sistema. Esto ocurre porque el proceso padre no ha "recogido" el estado de terminación del hijo.

Posible tabla de procesos (BCP) Block Control Process

Process management	Memory management	File management
Registers	Pointer to text segment	Root directory
Program counter	Pointer to data segment	Working directory
Program status word	Pointer to stack segment	File descriptors
Stack pointer		User ID
Process state		Group ID
Priority		
Scheduling parameters		
Process ID		
Parent process		
Process group		
Signals		
Time when process started		
CPU time used		
Children's CPU time		
Time of next alarm		

- PSW: el estado del proceso
- Identificador:
  - PID: Process ID.
  - PPID: identificador del padre del proceso.
  - UID: identificador del usuario que inició el proceso.
- IP/PC
- Registros del procesador
- Información de planificación de CPU (prioridad de ejecución de un proceso)
- Información de manejo de memoria
- Información de E/S
- Información contable (por ejemplo, tiempo en CPU)

Planificadores

Categorías de Planificadores Según el Tipo de Proceso

Categoría de Planificadores	Características	Uso Típico	Objetivo del Planificador
Procesamiento por Lotes	- Ejecución en masa - Planificación por tiempo - No requiere interacción constante	- Procesamiento de datos - Cálculos extensivos - Generación de informes	- Maximizar el throughput - Minimizar el tiempo de espera
Interactivo	- Interacción continua - Planificación basada en prioridad - Requiere respuesta inmediata	- Aplicaciones de escritorio - Juegos - Sistemas de navegación	- Minimizar el tiempo de respuesta - Garantizar la interactividad
Tiempo Real	- Cumplimiento de plazos - Determinismo - Alta predictibilidad	- Sistemas embebidos - Control industrial - Telecomunicaciones	- Garantizar el cumplimiento de plazos - Minimizar la variabilidad en el tiempo de respuesta

Tipos de Planificadores

Sin Expulsión (No Apropiativo)

1. **Primero en Entrar, Primero en Servir (FCFS)**
   - **Descripción:** Los procesos se ejecutan en el orden en que llegan a la cola de planificación. El primer proceso en entrar es el primero en ser ejecutado.
   - **Ventajas:** Simplicidad y facilidad de implementación.
   - **Desventajas:** Puede causar el efecto de "convoy", donde los procesos largos bloquean la ejecución de procesos cortos.
2. **El Trabajo Más Corto Primero (SJF)**
   - **Descripción:** Los procesos con el tiempo de ejecución más corto se ejecutan primero. No interrumpe los procesos en ejecución.
   - **Ventajas:** Minimiza el tiempo promedio de espera.
   - **Desventajas:** Puede llevar a la inanición de procesos largos (que nunca sean ejecutados) y puede ser difícil predecir el tiempo de ejecución exacto.

3. **El Menor Tiempo Restante a Continuación (SRTN)**
   - **Descripción:** Variación de SJF que permite la preempción. Los procesos con el menor tiempo de ejecución restante se ejecutan primero.
   - **Ventajas:** Reduce el tiempo promedio de espera y es más eficiente en términos de respuesta.
   - **Desventajas:** Puede llevar a la inanición de procesos con tiempos de ejecución largos.

Con Expulsión (Apropiativo)

1. **Round Robin (Turno Circular)**
   - **Descripción:** Cada proceso recibe un quantum de tiempo fijo para su ejecución. Si el proceso no termina dentro del quantum, es interrumpido y colocado al final de la cola.
   - **Ventajas:** Equidad en la asignación de tiempo de CPU y simplicidad.
   - **Desventajas:** Puede causar un alto overhead debido al cambio frecuente de contexto y no siempre es eficiente para todos los tipos de procesos.
2. **Prioridades**
   - **Descripción:** Los procesos se ejecutan en función de su prioridad. Los procesos con mayor prioridad son atendidos antes que los de menor prioridad.
   - **Tipos de Prioridades:**
     - **Fijas:** Las prioridades no cambian durante la ejecución del proceso.
     - **Variables:** Las prioridades pueden ajustarse dinámicamente durante la ejecución, basadas en factores como el tiempo de espera.
   - **Ventajas:** Permite priorizar procesos importantes y adaptarse a condiciones cambiantes.
   - **Desventajas:** Puede causar inanición de procesos con baja prioridad y la gestión de prioridades puede ser compleja.

Resumen y Comparación

Planificador	Con Expulsión	Sin Expulsión	Características	Ventajas	Desventajas
Primero en Entrar, Primero en Servir (FCFS)	No	Sí	Ejecución en el orden de llegada	Simplicidad, fácil de implementar	Puede causar efectos de convoy
El Trabajo Más Corto Primero (SJF)	No	Sí	Ejecución prioritaria para procesos con tiempos cortos	Minimiza tiempo promedio de espera	Inanición de procesos largos
El Menor Tiempo Restante a Continuación (SRTN)	Sí	Sí	Ejecución prioritaria para procesos con menor tiempo restante	Reduce tiempo promedio de espera	Inanición de procesos largos
Round Robin (Turno Circular)	Sí	-	Quantum de tiempo fijo para cada proceso	Equidad en la asignación de CPU	Alto overhead debido a cambios de contexto
Prioridades	Sí	No	Ejecución basada en prioridad asignada	Permite priorización flexible	Inanición de procesos con baja prioridad

Despachador

El **modulo del despachador** le da el control de la CPU al proceso seleccionado por el planificador de corto plazo esto abarca:

- Cambio de contexto
- Cambio a modo usuario
- Hacer el salto a la dirección en el programa de usuario para reiniciar ese programa

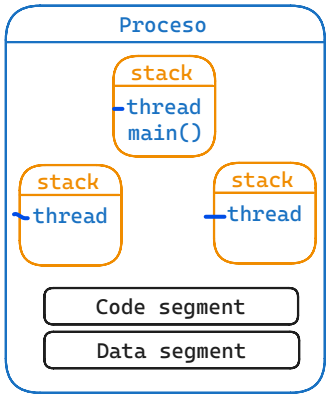
Hilos

Los hilos, también conocidos como "threads" en inglés, son las unidades más pequeñas de ejecución dentro de un proceso en un sistema operativo. Permiten que un proceso realice múltiples tareas simultáneamente, mejorando la eficiencia y el rendimiento de las aplicaciones. Aquí tienes una explicación más detallada:

> **Definicion**
>
> Un **hilo** es una secuencia de ejecución dentro de un proceso. Cada **hilo** tiene su propio conjunto de registros, contador de programa y pila, pero comparte el mismo espacio de direcciones y recursos del proceso principal, como memoria y archivos.

Relación entre Hilos y procesos

Un **proceso** inicia ejecutando su punto de entrada como un hilo, los **hilos** pueden crear otros hilos dentro del proceso, cada hilo tiene su propio **stack** y todos los hilos dentro del proceso comparten código y segmentos de datos



Multihilo vs Monohilo

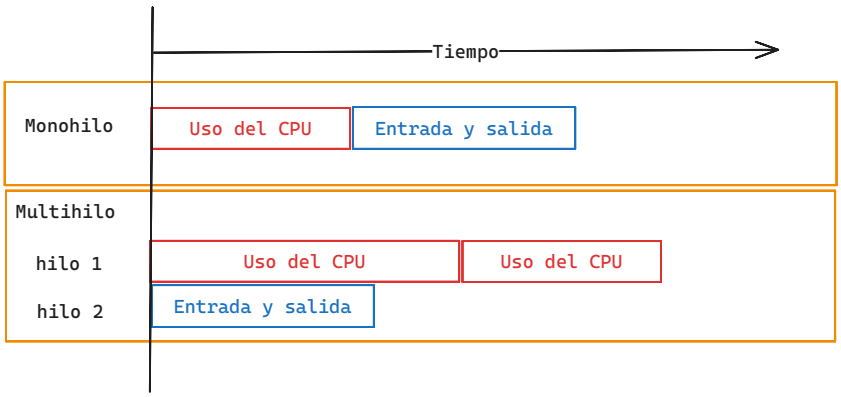
Monohilo

Un único hilo realiza todas las tareas secuencialmente. Más simple en términos de programación, pero puede no aprovechar al máximo los recursos del sistema y puede experimentar bloqueos prolongados.

Multihilo

Múltiples hilos pueden ejecutar tareas simultáneamente, lo que mejora la eficiencia y la capacidad de respuesta. Sin embargo, requiere una gestión cuidadosa de la sincronización y puede ser más complejo

de implementar.

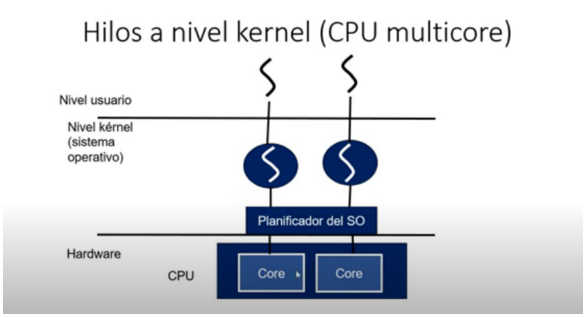


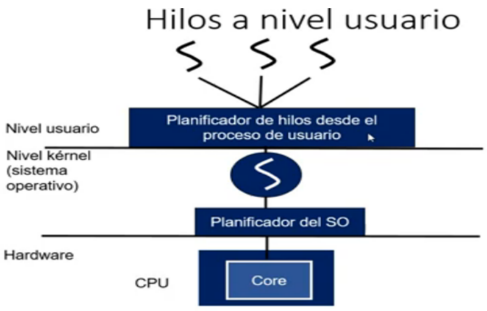
Aspecto	Monohilo (Single-Threaded)	Multihilo (Multi-Threaded)
Ejecución de Tareas	Un único hilo ejecuta todas las tareas secuencialmente.	Múltiples hilos pueden ejecutar tareas simultáneamente.
Concurrencia	No permite concurrencia real; las tareas se ejecutan una tras otra.	Permite concurrencia; múltiples hilos pueden trabajar en paralelo.
Complejidad del Código	Más simple; no se necesita gestionar la sincronización entre hilos.	Más complejo; requiere gestionar sincronización y evitar condiciones de carrera.
Utilización de Recursos	Puede no aprovechar al máximo los recursos del sistema (un solo núcleo).	Mejora la utilización de recursos al aprovechar múltiples núcleos de procesamiento.
Gestión de Bloqueos	Las operaciones bloqueantes pueden detener el hilo principal.	Otros hilos pueden seguir ejecutándose mientras uno está bloqueado.
Desempeño	Eficiente en tareas simples; menos eficiente en tareas concurrentes.	Mejora el desempeño en tareas que se benefician de la paralelización.
Ejemplos de Uso	Aplicaciones simples como editores de texto.	Aplicaciones complejas como servidores web y juegos.

Cuando tenemos una memoria que tiene diferentes procesos ejecutando, cada proceso tiene su dirección, sus datos y si pila, entonces al **paralelizar los procesos** puede suceder que se comuniquen entre ellos y eso no lo queremos sin que lo solicitemos entonces de eso se encarga el sistema operativo, de funcionar como Arbitro de la comunicación de procesos, entonces se requiere del sistema operativo para controlar eso, pero si **paralelizamos los hilos** no hay problema con que se comuniquen entre ellos ya que están dentro del mismo proceso, por lo que no requerimos que el sistema operativo lo controle y eso nos libera un poco el sistema operativo.

Hilos a nivel kernel vs Hilos a nivel usuario

- **Hilos a Nivel Usuario**: Son gestionados por bibliotecas de usuario sin la intervención directa del sistema operativo. Son rápidos de crear y destruir, pero tienen limitaciones en términos de sincronización y manejo de bloqueos, ya que el sistema operativo no puede diferenciar entre hilos de un mismo proceso.
- **Hilos a Nivel Kernel**: Son gestionados directamente por el sistema operativo, lo que permite una mejor sincronización y manejo de recursos. Aunque la creación y destrucción de hilos es más costosa, ofrecen una mayor flexibilidad y capacidad para manejar la concurrencia de manera más efectiva.


Hilos a nivel kernel (CPU multicore)
Nivel usuario
Nivel kérnel (sistema operativo)
Planificador del SO
Hardware
CPU
Core
Core


Hilos a nivel usuario
Nivel usuario
Planificador de hilos desde el proceso de usuario
Nivel kérnel (sistema operativo)
Planificador del SO
Hardware
CPU
Core