🟣

Clase 4

Materia	ISO Teoría
Estado	Done
Fecha	@August 30, 2022

Procesos

> 📌 Un proceso es un **programa en ejecución.** Un programa una entidad abstracta (…9 y lo que está en disco es el código del programa.

Proceso tiene varios sinónimos, entre ellos tarea y job.

- **Si está en disco** → Programa muerto.
- **Si se ejecuta** → Es un proceso, está vivo y tiene un ciclo de nacer, transitar y morir (🙂)

Diferencia entre un programa y un proceso:

El **programa** es estático, no tiene un *program counter* (posición de la instrucción que se ejecuta) y existe desde que se edita hasta que se borra.

El **proceso** es dinámico, tiene un *program counter* y su ciclo de vida comprende desde que se lo dispara hasta que termina.

El modelo del proceso

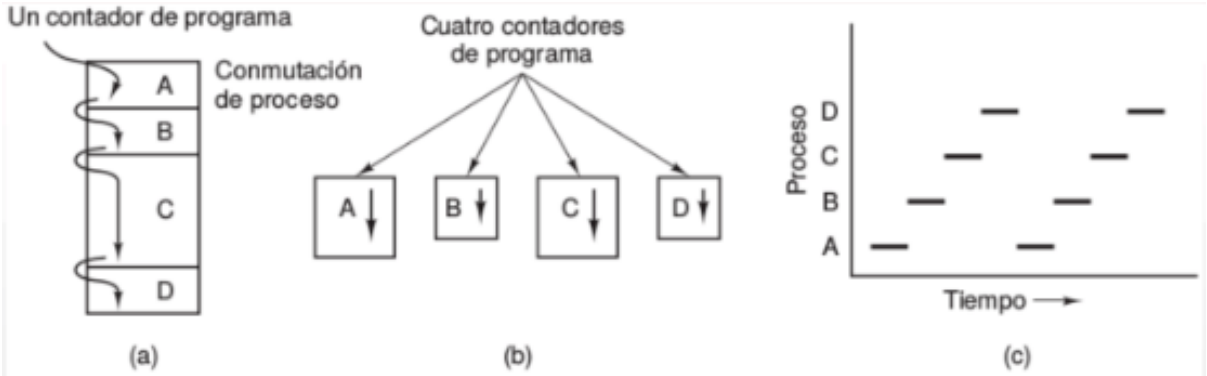


Figura 2-1. (a) Multiprogramación de cuatro programas. (b) Modelo conceptual de cuatro procesos secuenciales independientes. (c) Sólo hay un programa activo a la vez.

Es un modelo conceptual de 4 procesos secuenciales e independientes entre sí. **Sólo un proceso está activo en cualquier instante (si tenemos una sola CPU).**

Componentes del proceso

Código,

El proceso es una abstracción 🌈 (?) Para ejecutarse dicho proceso, debe incluir:

- Sección de código (texto)
- Sección de datos (variables globales)
- **Pilas,** dónde están los datos temporales como parámetros, variables temporales y dirección de return.
    - Tiene que tener, como mínimo, dos pilas. Una de modo usuario y otra más para el modo kernel.
    - Estas pilas se crean automáticamente y su tamaño se ajusta run-time
    - Está formado por stack frames, que son empujados (`pushed-` al llamar a una rutina) y se retiran (`popped-` cuándo se retorna de la rutina)
    - El stack frame tiene los parámetros de la rutina (variables locales) y datos necesarios para recuperar el stack frame anterior (PC, y el valor del stack pointer en el momento del llamado)

Atributos del proceso

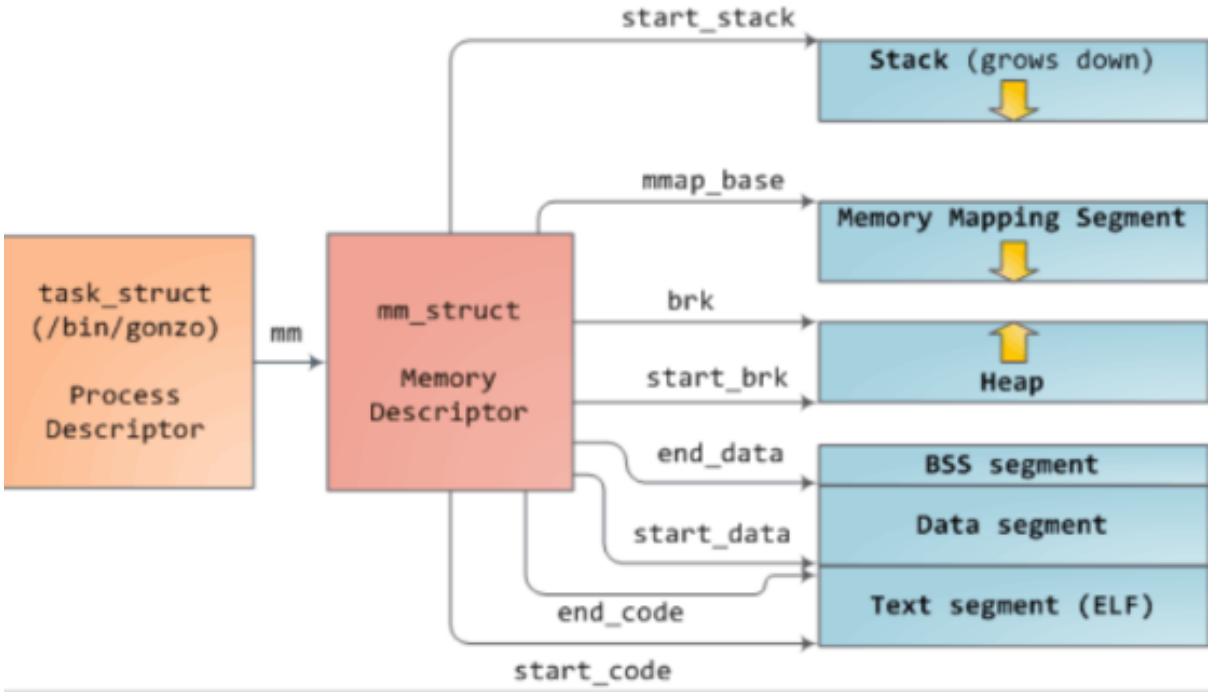
Los **atributos del proceso** son los identificadores del mismo y de su proceso padre, id de quién lo disparo, si hay una estructura de grupos o grupo que lo disparó y desde qué terminal y quién lo ejecutó.

PCB — *Process Control Block*

Y también el **PCB** (*process control block*) que es una estructura de datos asociada al proceso, existe una PCB por cada uno de ellos.

- El PCB es lo primero que se crea y lo último que se borra en cada proceso.
- El PCB lo podemos pensar como un gran registro en el que se guardan los atributos anteriormente mencionados, también guarda punteros y direcciones de memoria relacionadas al proceso.
- Tiene la información asociada a cada proceso:
  - PID, PPID, etc.
  - Valores de los registros de la CPU (PC, AC, etc).
  - Planificación (estado, prioridad y tiempo consumido del proceso, etc).
  - Ubicación (donde está el proceso) en memoria.
  - Accounting (cantidades, cuanta memoria ocupó, cuanta entrada salida ocupó).
  - Entrada / salida (estado, pendientes, etc).

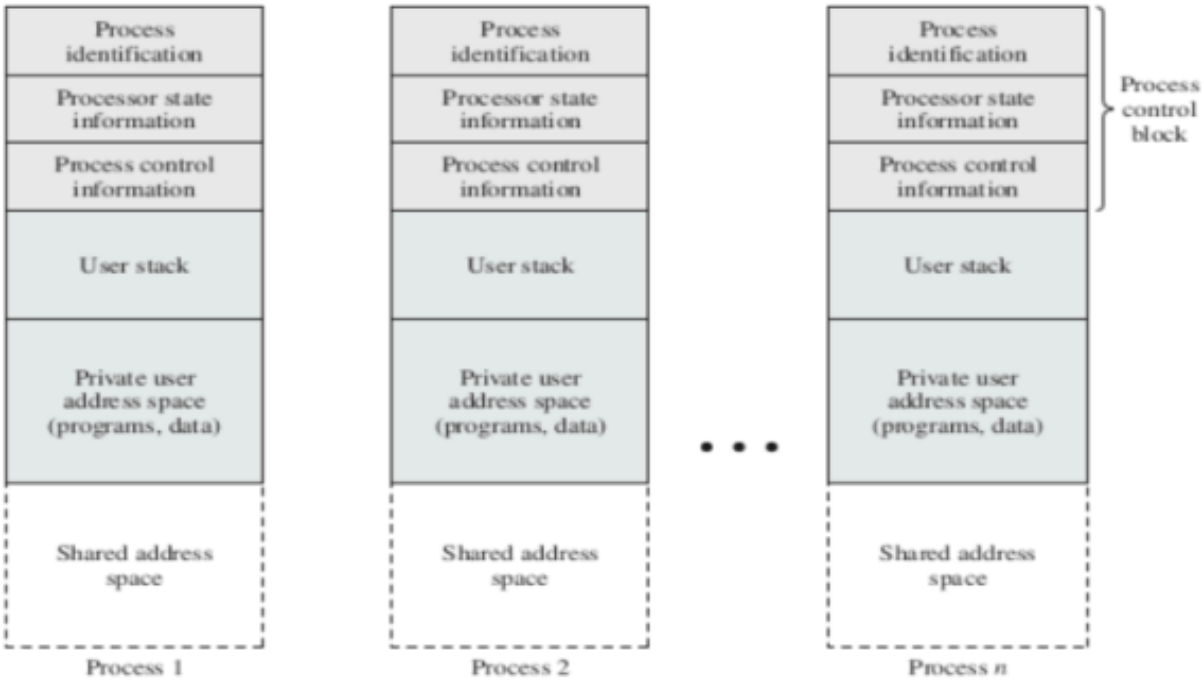
Administración de procesos	**Administración de memoria**	**Administración de archivos**
Registros	Apuntador a la información del segmento de texto	Directorio raíz
Contador del programa	Apuntador a la información del segmento de datos	Directorio de trabajo
Palabra de estado del programa	Apuntador a la información del segmento de pila	Descripciones de archivos
Apuntador de la pila		ID de usuario
Estado del proceso		ID de grupo
Prioridad		
Parámetros de planificación		
ID del proceso		
Proceso padre		
Grupo de procesos		
Señales		
Tiempo de inicio del proceso		
Tiempo utilizado de la CPU		
Tiempo de la CPU utilizado por el hijo		
Hora de la siguiente alarma		



Espacio de direcciones de un proceso

Es el **conjunto de direcciones de memoria que ocupa un proceso** (Stack, código, datos) **no incluye** al PCB o las tablas asociadas.

- Un proceso en **modo usuario sólo puede acceder a su espacio de direcciones,** el cuál es limitado.
- En **modo kernel se puede acceder a estructuras internas** (PCB del proceso) **o a espacios de direcciones de otros procesos** (rompe el límite).



Contexto del proceso

El **contexto incluye toda la información que el SO necesita mantener para administrar el proceso**, y en consecuente, el CPU pueda ejecutarlo correctamente. Son parte del contexto: los registros de CPU, el contador de programa (PC), prioridades del proceso, si tiene E/S pendientes, etc.

Cambio de contexto (Context Switch):

- Se produce **cuando la CPU cambia de un proceso a otro**.
- Se **DEBE resguardar el contexto del proceso anterior**, que pasa a esperar para volver a ejecutarse después en la CPU.
- Se debe **cargar el contexto del nuevo proceso y comenzar desde la instrucción siguiente a la última ejecutada en dicho contexto**.
- Es **tiempo NO productivo de CPU** (la CPU debe estar optimizada para no sufrir tanto consumo de ciclos). El tiempo que consume dicho cambio depende del soporte del Hardware.

Kernel, otra vez

Se ejecuta en el procesador como cualquier otro proceso **aunque no sea un proceso**, tiene dos enfoques de diseño:

➡️ El **Kernel independiente:** es el Kernel que se ejecuta **fuera** de todo proceso.

- Diseño usado por los primeros SO
- Tiene su propio región de memoria y su propio stack.
- Cuando un proceso es interrumpido o realiza una System Call, el contexto del proceso se salva y el control se pasa al kernel del sistema.
- Una vez finalizada su actividad, devuelve el control al proceso (o a otro proceso diferente)
- **Ventaja:** Es un modelo sencillo.
- **Desventaja:** Genera una carga de tiempo no productivo en el CPU.

➡️ El **Kernel "dentro" de todos los procesos**: el código del Kernel se encuentra DENTRO del espacio de direcciones de CADA proceso. Ejecutándose en el **mismo contexto que algún proceso de usuario. No se repite en muchos espacios de memoria, ya que en realidad los procesos conoce dónde se encuentra el Kernel.**

- Usa una pila para el usuario y otra para el modo kernel.
- Dentro de un proceso se encuentra el código del programa (usuario) y el código de los módulos del kernel (privilegio)
- El código del kernel se comparte por todos los procesos, como una librería.
- El kernel se puede ver como un conjunto de rutinas que el proceso utiliza
- Cada interrupción (incluyendo llamadas al sistema), son atendidas en el contexto del proceso que se encuentra en ejecución.
- **Desventaja**: Constante cambio entre modo usuario y modo kernel
- **Ventaja**: No genera cambio de contexto (ya que se ejecuta en el mismo contexto que el proceso de usuario), por lo que es más rápido y eficiente.