

## INTRODUCCIÓN A LOS SISTEMAS OPERATIVOS – TEMA 2: PROCESOS

### Creación de procesos:

- Un proceso se crea por otro proceso.
- Un proceso padre tiene uno o más procesos hijos.
- Se forma un árbol de procesos.
- El proceso padre nace ya creado (por el S.O, éste carga una imagen de un proceso en memoria).

### Actividades en la creación:

- Crear la PCB.
- Asignar PID al proceso creado (Process Identification) → id único que lo identifica.
- Asignarle memoria para regiones:
  - Stack, text (código), datos.
- Crear estructuras de datos asociadas al mismo:
  - Fork (copiar contexto, regiones de datos, text y stack).

### Relación entre procesos padre e hijo:

Con respecto a la ejecución:

- El padre puede continuar ejecutándose concurrentemente con su hijo.
- El padre puede esperar a que el proceso hijo (o los hijos), terminan para continuar la ejecución (no utilizado frecuentemente, a menos que la espera sea voluntaria).
- Tanto padre como hijo compiten por igual por la CPU.

Con respecto al espacio de direcciones:

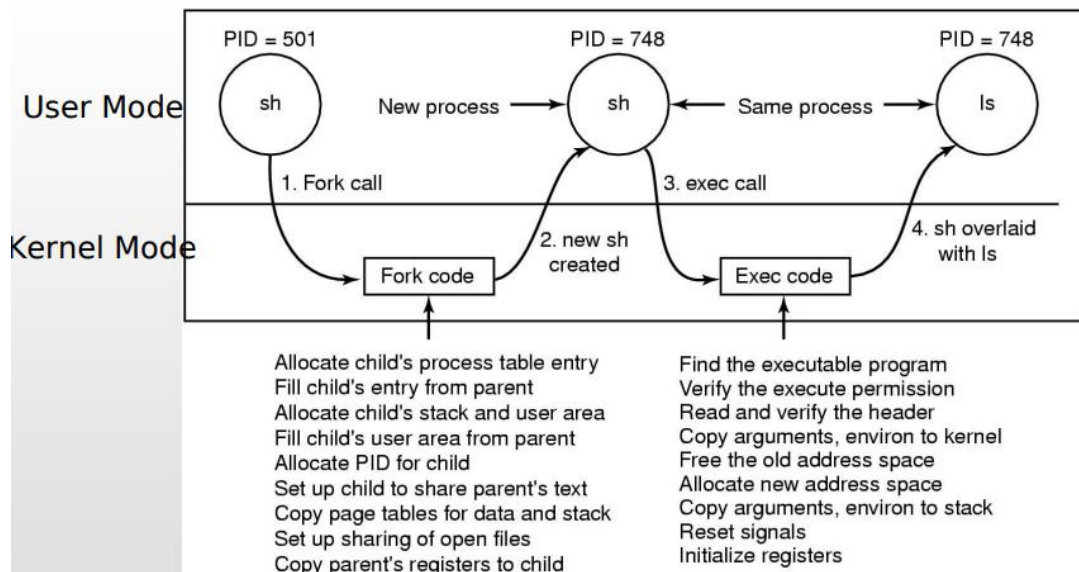
- El hijo es un duplicado del proceso padre (caso UNIX).
- Se crea el proceso y se le carga adentro el programa (caso WINDOWS, arranca de 0, vacío).

### Creación de procesos:

- En UNIX:
  - System call `fork()` crea nuevo proceso.
    - Retorna  $> 0$  al padre (PID del hijo)
    - Retorna  $< 0$  ERROR.
    - Retorna 0 al hijo.
  - System call `execve()`, generalmente usado después del `fork`, carga un nuevo programa en el espacio de direcciones. Todo lo que estaba antes ya no está. Básicamente carga el contenido del programa en el proceso que ya tengo (hijo).

## INTRODUCCIÓN A LOS SISTEMAS OPERATIVOS – TEMA 2: PROCESOS

- En WINDOWS:
  - System call CreateProcess() crea un nuevo proceso y carga el programa para ejecución.



### Terminación de procesos:

- Ante un (exit) se retorna el control al sistema operativo.
  - El proceso padre puede esperar recibir un código de retorno (via wait). Generalmente se lo usa cuando se requiere que el padre espere a los hijos.
- Proceso padre puede terminar la ejecución de sus hijos (kill):
  - La tarea asignada al hijo es finalizada.
  - Cuando el padre termina su ejecución:
    - No se permite a los hijos continuar.
    - Terminación en cascada.

### Procesos cooperativos e independientes:

- Independiente: el proceso no afecta ni puede ser afectado por la ejecución de otros procesos. No comparte ningún tipo de dato, el problema a resolver sólo implica ese proceso.
  - Cooperativo: afecta o es afectado por la ejecución de otros procesos en el sistema, se resuelve un problema entre varios procesos.
- **¿Para qué nos sirven los procesos cooperativos?**
- Para compartir información → un archivo por ej.
  - Para acelerar el cómputo → separar una tarea en sub-tareas que cooperan ejecutándose en paralelo.
  - Para planificar tareas de manera tal que haya ejecución paralela.