PROCESOS

DEPARTAMENTO DE CIENCIAS E INGENIERÍA
DE LA COMPUTACIÓN

UNIVERSIDAD NACIONAL DEL SUR

KMC © 2020

AGENDA

1. Concepto de Proceso
2. Planificación de Proceso
3. Operaciones sobre Procesos
4. Comunicaciones Interprocesos (IPC)
    1. Ejemplos de Sistemas de IPC
5. Pipe

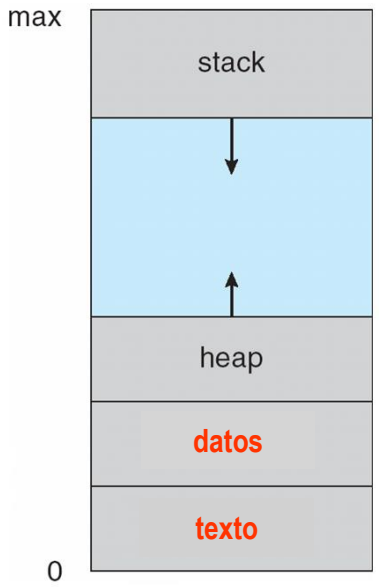
AGENDA

1. **Concepto de Proceso**
2. Planificación de Proceso
3. Operaciones sobre Procesos
4. Comunicaciones Interprocesos (IPC)
    1. Ejemplos de Sistemas de IPC
5. Pipe

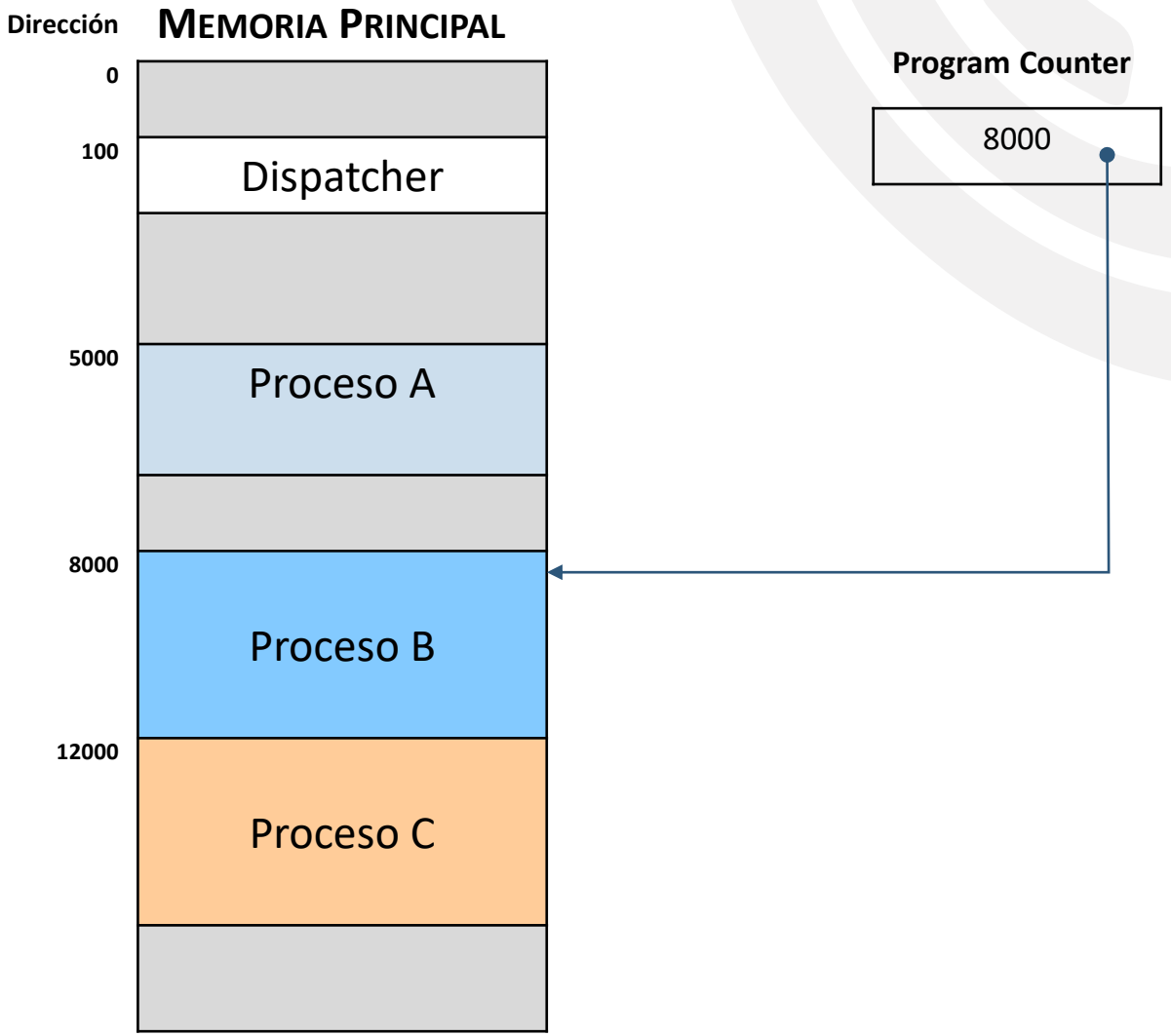
CONCEPTO DE PROCESO

- Un SO ejecuta una variedad de programas:
  - Sistema Batch – jobs
  - Sistemas de Tiempo Compartido – programas de usuario o tareas
- **Proceso** – un programa en ejecución.

- Un proceso incluye:
  - contador de programa
  - stack
  - sección de datos



EJECUCIÓN DE UN PROCESO



EJECUCIÓN DE UN PROCESO - TRAZAS

Traza Proceso A	Traza Proceso B	Traza Proceso C
5000	8000	12000
5001	8001	12001
5002	8002	12002
5003	8003	12003
5004		12004
5005		12005
5006		12006
5007		12007
5008		12008
5009		12009
5010		12010
5011		12011

5000 – Dirección inicio del programa del Proceso A
8000 - Dirección inicio del programa del Proceso B
12000 - Dirección inicio del programa del Proceso C

EJECUCIÓN DE UN PROCESO - TRAZAS

100 – Dirección inicio del dispatcher

1	5000	
2	5001	
3	5002	
4	5003	
5	5004	
6	5005	
----------	----------	timeout
7	100	
8	101	
9	102	
10	103	
11	104	
12	105	
13	8000	
14	8001	
15	8002	
16	8003	
----------	----------	I/O Req.
17	100	
18	101	
19	102	
20	103	
21	104	
22	105	
23	12000	
24	12001	
25	12002	
26	12003	
27	12004	
28	12005	
----------	----------	Timeout
29	100	
30	101	
31	102	
32	103	
33	104	
34	105	
35	5006	
36	5007	
37	5008	
38	5009	
39	5010	
40	5011	
----------	----------	Timeout
41	100	
42	101	
43	102	
44	103	
45	104	
46	105	
47	12006	
48	12007	
49	12008	
50	12009	
51	12010	
52	12011	
----------	----------	Timeout

ESTADO DE LOS PROCESOS

Un proceso cambia de ***estado*** durante su ejecución.

- nuevo: el proceso es creado.
- corriendo (ejecutando): las instrucciones están siendo ejecutadas.
- espera: el proceso está esperando que ocurra algún evento.
- listo: el proceso está esperando ser asignado a la CPU.
- terminado: el proceso ha finalizado su ejecución.

DIAGRAMA DE ESTADOS DE UN PROCESO – 3 ESTADOS

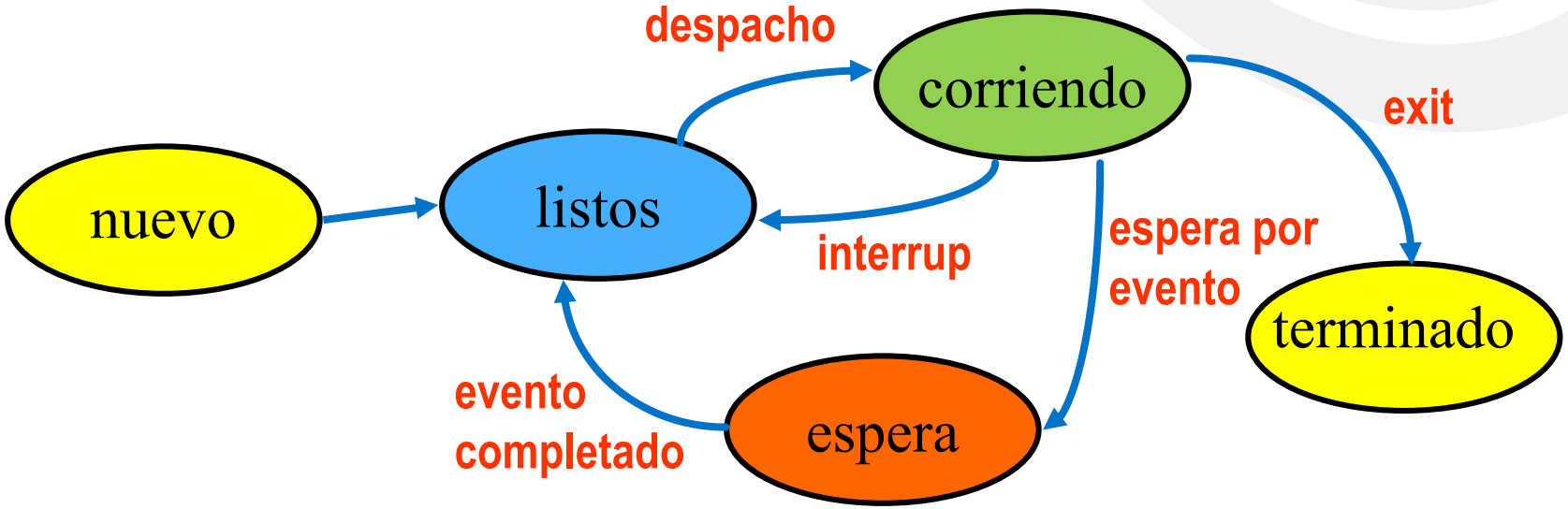
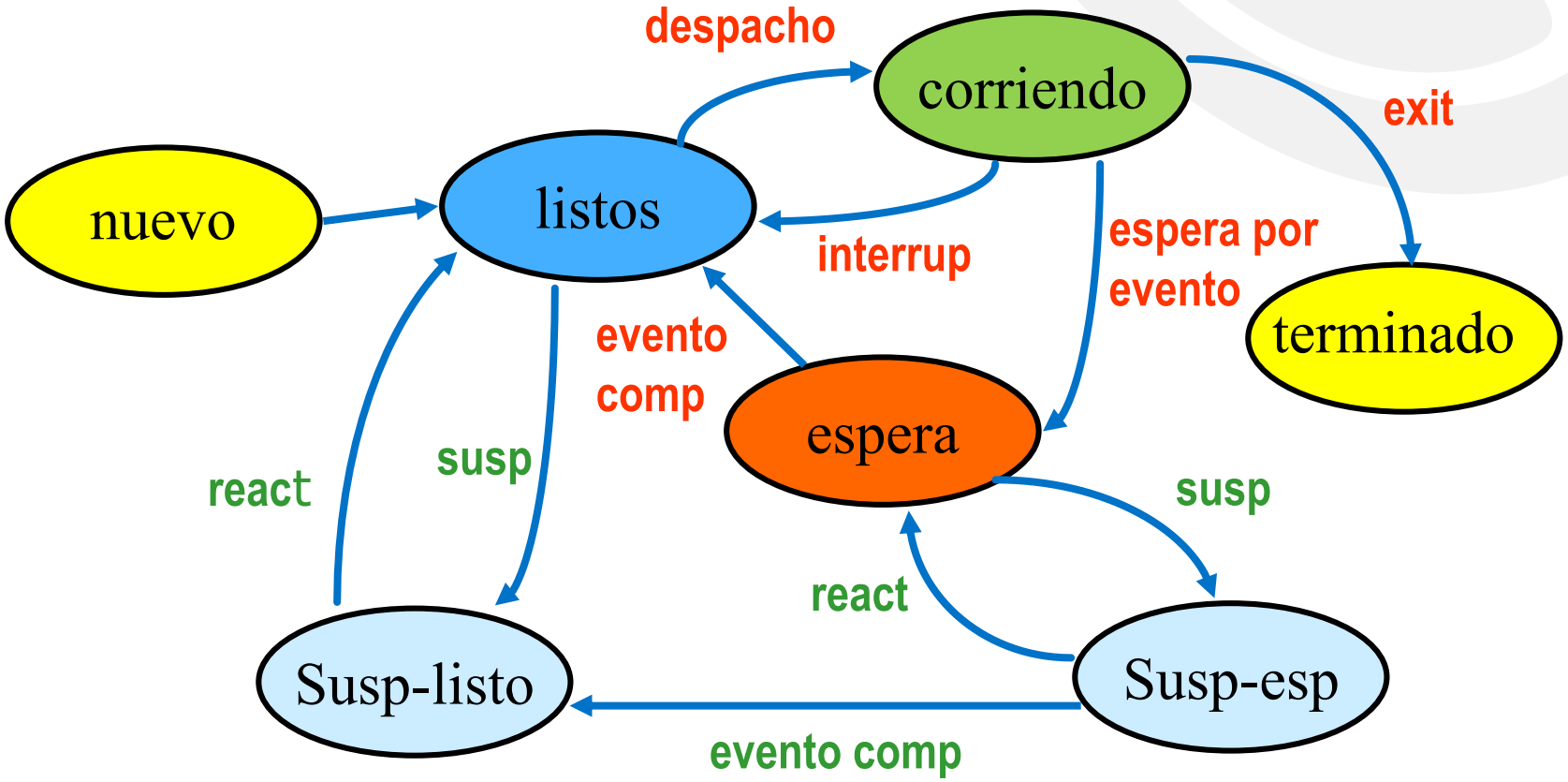


DIAGRAMA DE ESTADOS DE UN PROCESO – 5 ESTADOS

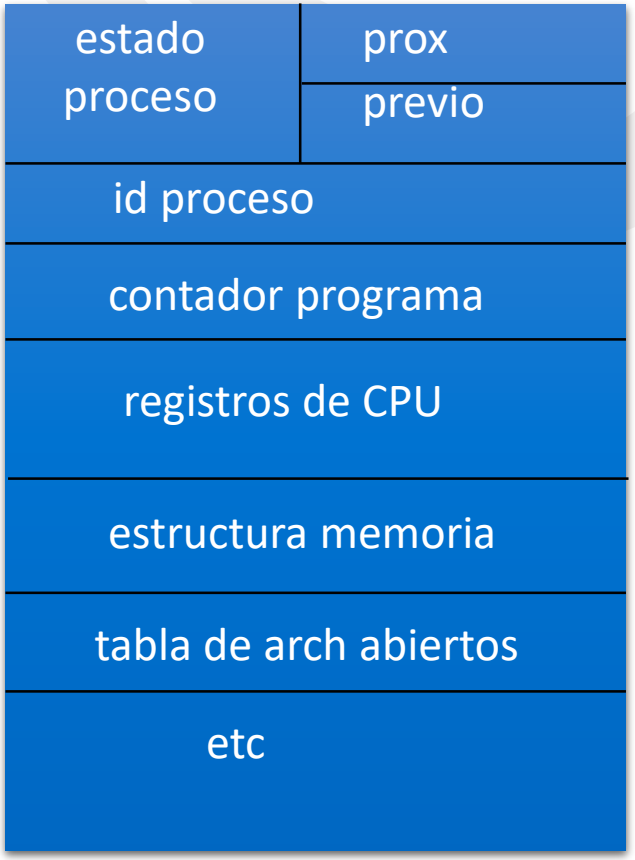


BLOQUE DE CONTROL DE PROCESOS (PCB)

Es una estructura de dato que contiene información asociada con cada proceso.

- Estado de Proceso
- Contador de Programa
- Registros de CPU
- Información de planificación de CPU
- Información de administración de memoria
- Información contable
- Información de estado E/S

PCB: Process Control Block

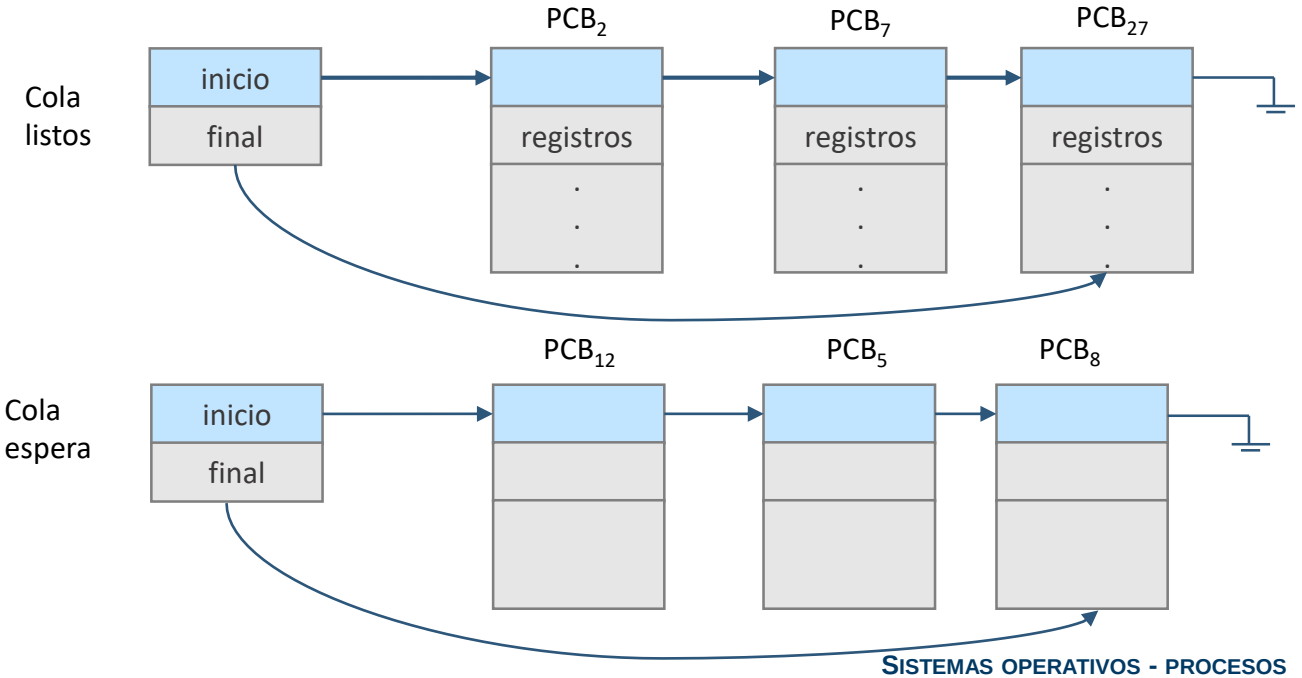


AGENDA

1. Concepto de Proceso
2. **Planificación de Proceso**
3. Operaciones sobre Procesos
4. Comunicaciones Interprocesos (IPC)
    1. Ejemplos de Sistemas de IPC
5. Pipe

COLAS DE PLANIFICACIÓN DE PROCESOS

- Cola de Job (o tareas) – conjunto de todos los procesos en el sistema.
- Cola de listos – conjunto de todos los procesos residentes en memoria principal, listos y esperando para ejecutar.
- Colas de dispositivos – conjunto de procesos esperando por una E/S en un dispositivo de E/S.
- Migración de procesos entre las colas.



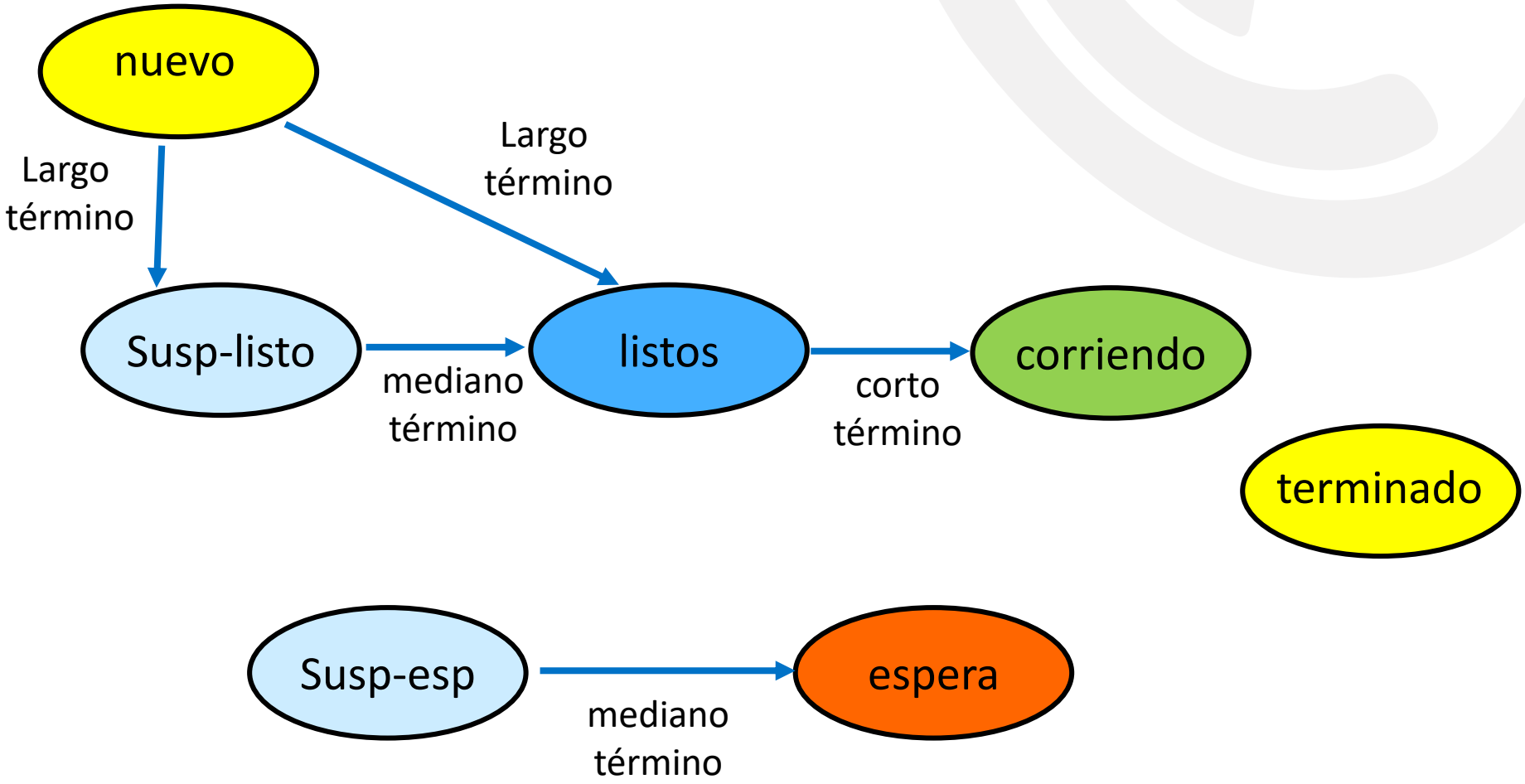
PLANIFICADORES DE PROCESOS

- Planificador de **largo término** (o planificador de jobs) – selecciona que procesos deberían ser puestos en la cola de listos.
- Planificador de **corto término** (o planificador de CPU) – selecciona que procesos deberían ser proximamente ejecutados y colocados en la CPU.
- Planificador de **mediano término**

PLANIFICADORES DE PROCESOS

- El planificador de corto término es invocado muy frecuentemente (milisegundos) $\Rightarrow$ (debe ser rápido).
- El planificador de largo término es invocado poco frecuentemente (segundos, minutos) $\Rightarrow$ (puede ser muy lento).
- El planificador de largo término controla el *grado de multiprogramación*.
- Los procesos pueden ser descriptos como:
  - ***Procesos limitados por E/S*** – pasa más tiempo haciendo E/S que computaciones, ráfagas (burst) de CPU muy cortas.
  - ***Procesos limitados por CPU*** – pasa más tiempo haciendo computaciones que E/S, ráfagas (burst) de CPU muy largas.

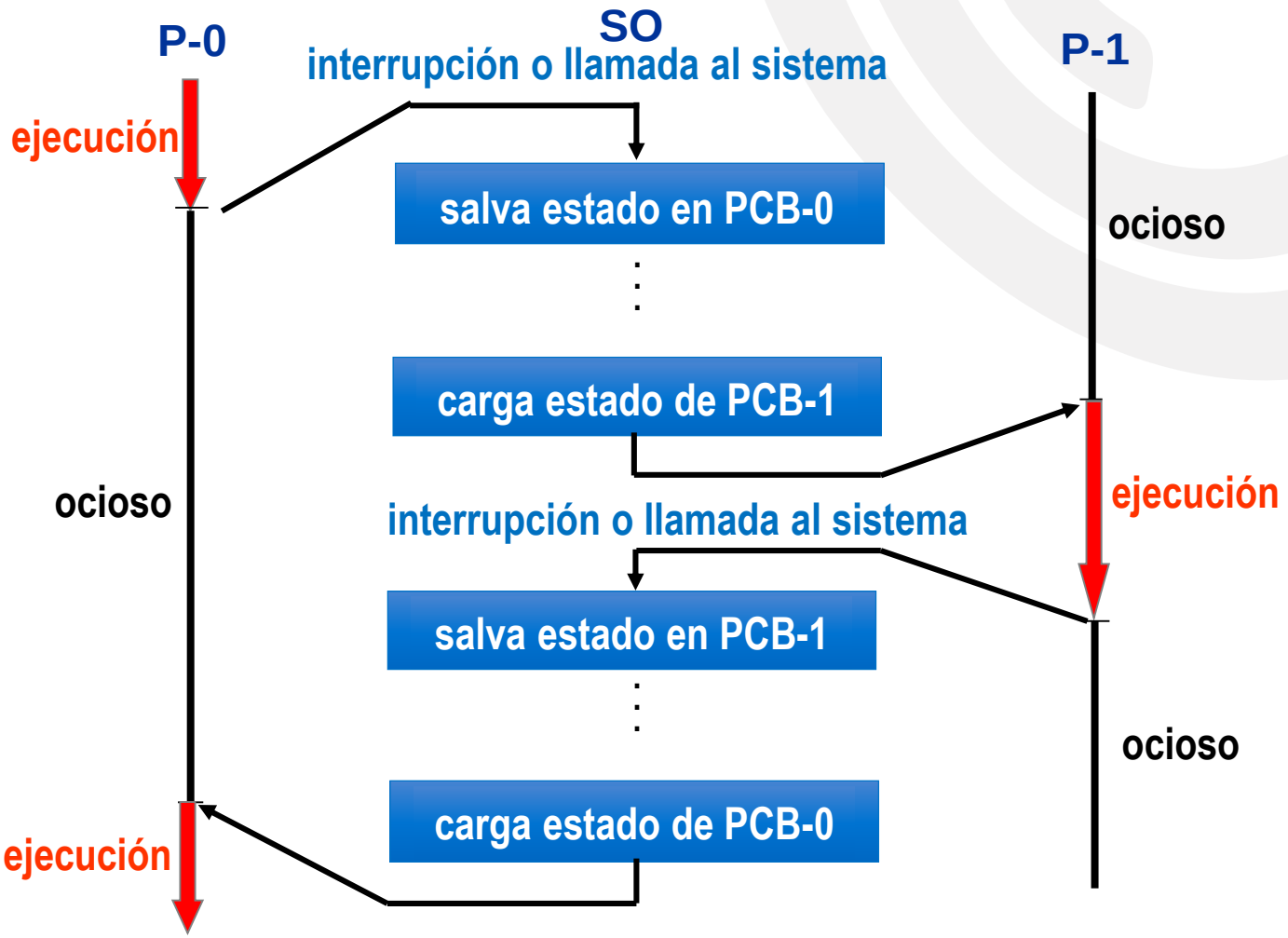
PLANIFICACIÓN Y TRANSICIÓN DE ESTADOS DE UN PROCESO



CAMBIO DE CONTEXTO

- Cuando la CPU conmuta a otro proceso, el sistema debe salvar el estado del viejo proceso y cargar el estado para el nuevo proceso vía un cambio de contexto.
- El contexto de un proceso está representado en el PCB
- El tiempo que lleva el cambio de contexto es sobrecarga; el sistema no hace trabajo útil mientras está conmutando.
- El tiempo depende del soporte de hardware.

CONMUTACIÓN DE CPU DE PROCESO A PROCESO



AGENDA

1. Concepto de Proceso
2. Planificación de Proceso
3. **Operaciones sobre Procesos**
4. Comunicaciones Interprocesos (IPC)
    1. Ejemplos de Sistemas de IPC
5. Pipe

CREACIÓN DE PROCESOS

Actividades

1. Asignar un identificador de proceso único al proceso.
2. Reservar espacio para proceso.
3. Inicializar el PCB.
4. Establecer los enlaces apropiados.
5. Crear o expandir otras estructuras de datos.

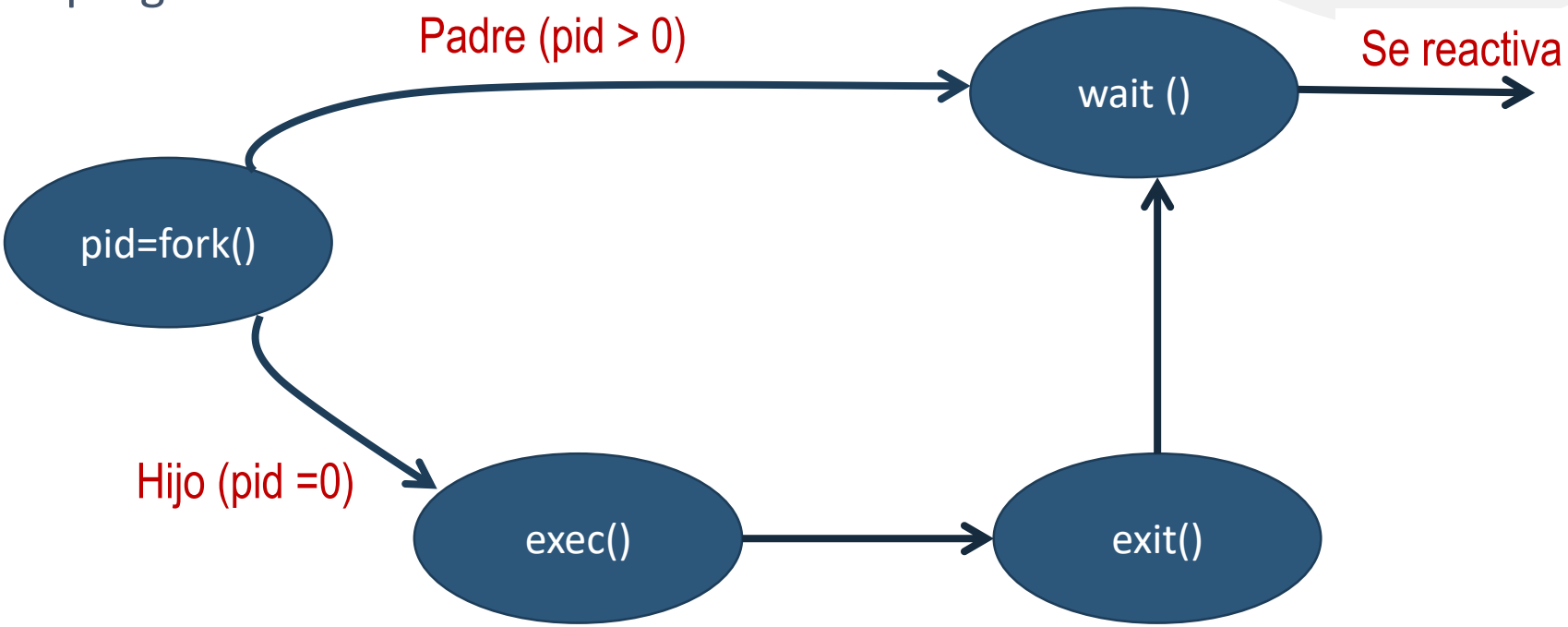
CREACIÓN DE PROCESOS - POLÍTICAS

- Espacio de direcciones
  - El hijo duplica el del padre.
  - El hijo tiene un programa cargado en él.
- Procesos padres crean procesos hijos, lo cuales, a su vez crean otros procesos, formando un árbol de procesos.
- Recursos compartidos
  - Padres e hijos comparten todos los recursos.
  - Hijo comparte un subconjunto de los recursos del padre.
  - Padre e hijo no comparten ningún recurso.
- Ejecución
  - Padres e hijos ejecutan concurrentemente.
  - Padres esperan hasta que los hijos terminan.

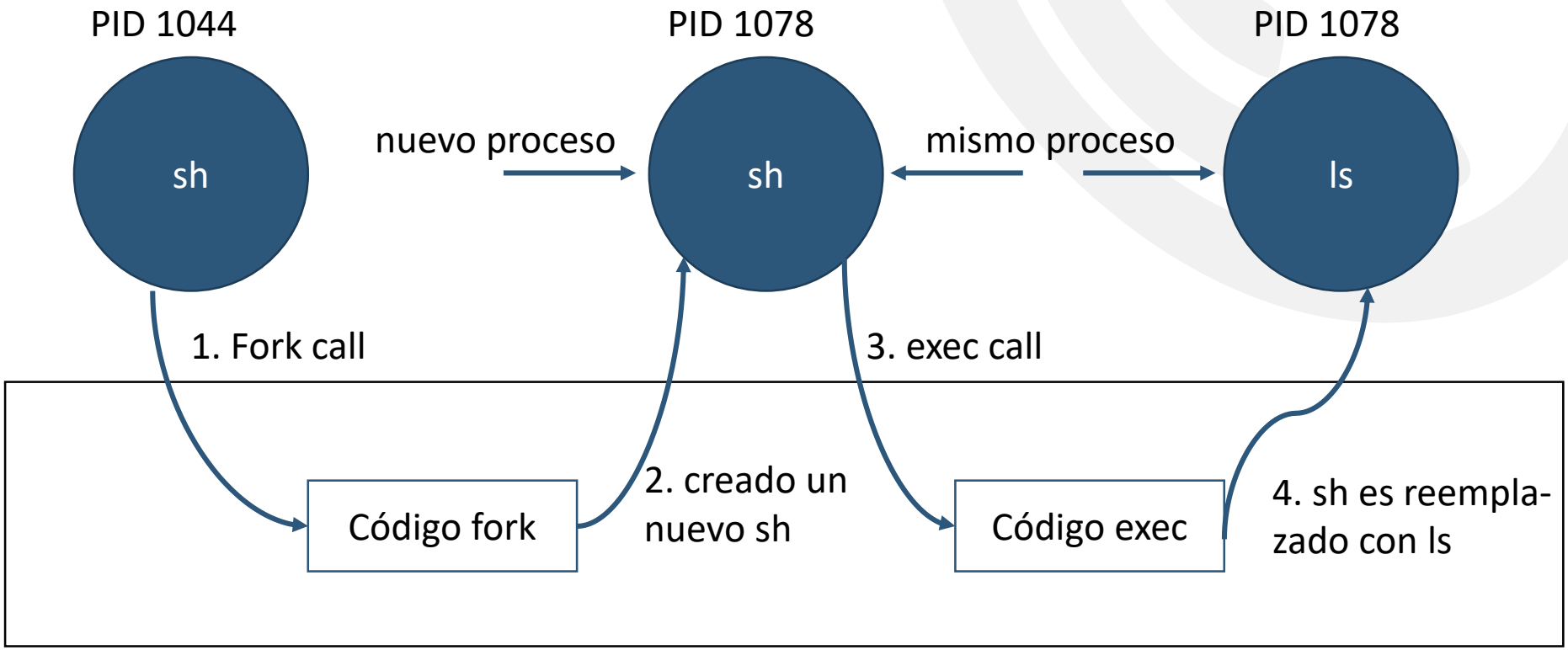
CREACIÓN DE PROCESOS

Ejemplos de UNIX

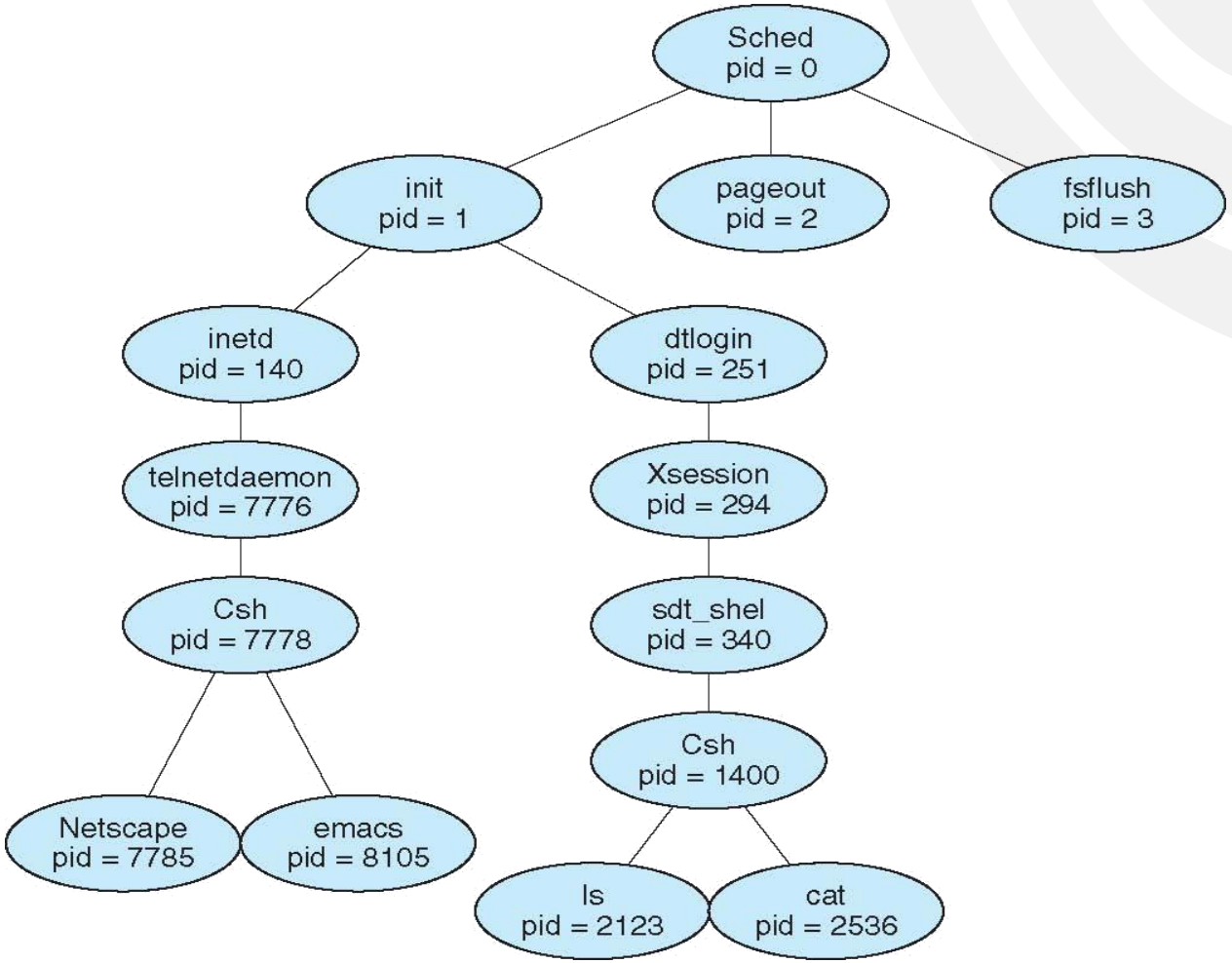
- La llamada a sistema **fork** crea un nuevo proceso
- La llamada a sistema **exec** es usada después del **fork** para reemplazar el espacio de memoria del proceso con un nuevo programa.



CREACIÓN DE PROCESOS EN UNIX - EJEMPLO



ÁRBOL DE PROCESOS EN UNIX



TERMINACIÓN DE PROCESOS

- El proceso ejecuta la última sentencia y espera que el SO haga algo (**exit**).
  - Los datos de salida del hijo se pasan al padre (vía **wait**).
  - Los recursos de los procesos son liberados por el SO.
- El padre puede terminar la ejecución del proceso hijo (**abort**).
  - El hijo ha excedido los recursos alocados.
  - La tarea asignada al hijo no es mas requerida.
  - El padre está terminando.
    - El SO no permite a los hijos continuar si su padre termina.
    - Terminación en cascada.

PROCESOS COOPERATIVOS

- Un proceso ***independiente*** no puede afectar ni ser afectado por la ejecución de otro proceso.
- Un proceso ***cooperativo*** puede afectar o ser afectado por la ejecución de otro proceso.
- Ventajas de los procesos cooperativos
  - Información compartida
  - Aceleración de la computación
  - Modularidad

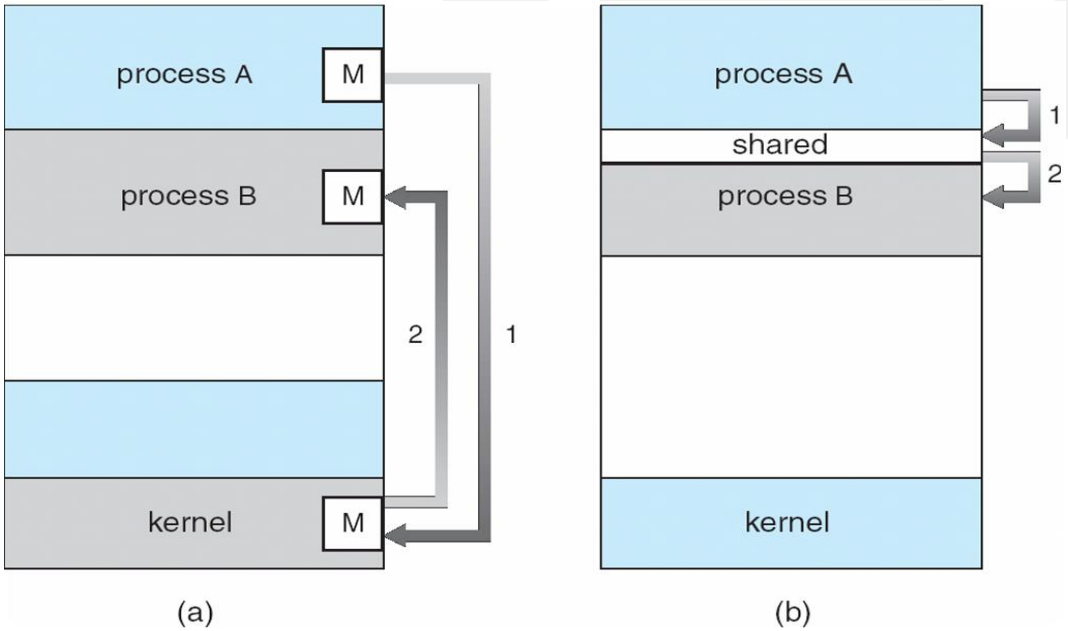
AGENDA

1. Concepto de Proceso
2. Planificación de Proceso
3. Operaciones sobre Procesos
4. **Comunicaciones Interprocesos (IPC)**
    1. **Ejemplos de Sistemas de IPC**
5. Pipe

COMUNICACIÓN INTERPROCESOS

- Los procesos cooperativos necesitan comunicación interprocesos (**IPC**)

- Dos modelos de IPC
  - Memoria compartida
  - Pasaje de Mensajes



Pasaje de Mensaje

Memoria Compartida

PROBLEMA DEL PRODUCTOR-CONSUMIDOR

- Paradigma procesos cooperativos, el proceso ***productor*** produce información que es consumida por un proceso ***consumidor***.
  - buffer ilimitado - no tiene límites prácticos en el tamaño del buffer.
  - buffer limitado supone que hay un tamaño fijo de buffer.

MODELOS DE COMUNICACIÓN – MEMORIA COMPARTIDA

Ejemplo de procesos cooperativos: Productor-Consumidor

- Implementarse con buffer no limitado o limitado.

```
While (true) {  //produce un ítem en nuevo_item
        While (((in + 1) % BUFFER_SIZE) == out); // espera
        buffer[in] = nuevo_ítem;
        in = (in + 1) % BUFFER_SIZE;
}
```

```
#define BUFFER_SIZE 10
Typedef struct {
        …
} item;
item  buffer[BUFFER_SIZE]
int in = 0;
int out = 0
```

```
While (true) {  //produce un ítem en nuevo_item
        While (in == out); // espera
        ítem_consumido = buffer[out];
        out = (out + 1) % BUFFER_SIZE;
} // en ítem_consumido se guarda el elemento consumido
```

COMUNICACIÓN ENTRE PROCESOS (IPC)

Sistema de mensajes

Los procesos se comunican uno con otro sin necesidad de variables compartidas.

- Los IPC proveen dos operaciones
  - **send**(mensaje)
  - **receive**(mensaje)
- Si P and Q desean comunicarse, necesitan:
  - Establecer un vínculo de comunicación entre ellos
  - Intercambiar mensajes vía send/receive
- Implementación de un vínculo de comunicación
  - lógico (p.e., propiedades lógicas)
  - físico (p.e., memoria compartida, canal hardware)

COMUNICACIÓN DIRECTA

- Los procesos deben nombrar al otro explícitamente:
  - **send** (P, mensaje) – envía un mensaje al proceso P
  - **receive**(Q, mensaje) – recibe un mensaje del proceso Q
- Propiedades del vínculo de comunicación
  - Un vínculo está asociado con exactamente un par de procesos que se comunican.
  - Entre cada par existe exactamente un vínculo.
  - El vínculo puede ser unidireccional, pero es usualmente bi-direccional.

COMUNICACIÓN INDIRECTA

- Los mensajes son dirigidos y recibidos desde ***mailboxes***
- Vínculo de comunicación
  - Se establece solo si los procesos comparten un mailbox común.
  - Puede ser asociado con muchos procesos.
  - Cada par de procesos puede compartir varios vínculos de comunicación.
  - Puede ser unidireccional o bi-direccional.

Operaciones

- crear un nuevo mailbox
- enviar y recibir mensajes por medio del mailbox
- destruir un mailbox

Primitivas

send(A, message) – enviar un mensaje al mailbox A

receive(A, message) – recibir un mensaje del mailbox A

SINCRONIZACIÓN

El pasaje de mensajes puede ser bloqueante o no bloqueante.

- **Bloqueante** es considerado **sincrónico**
  - ***Send bloqueante***
  - ***Receive bloqueante***
- **No bloqueante** es considerado **asincrónico**
  - ***Send no bloqueante***
  - ***Receive no bloqueante***

BUFFERING

La cola de mensajes asociada al vínculo se puede implementar de tres maneras.

1. Capacidad – 0 mensajes
   El enviador debe esperar por el receptor (rendezvous).
2. Capacidad limitada – longitud finita de n mensajes
   El enviador debe esperar si el vínculo está lleno.
3. Capacidad ilimitada – longitud infinita
   El enviador nunca espera.

AGENDA

1. Concepto de Proceso
2. Planificación de Proceso
3. Operaciones sobre Procesos
4. Comunicaciones Interprocesos (IPC)
    1. Ejemplos de Sistemas de IPC
5. **Pipe**

COMUNICACIÓN ENTRE PROCESOS - PIPE

Actúa como un conducto que permite que dos procesos se comuniquen

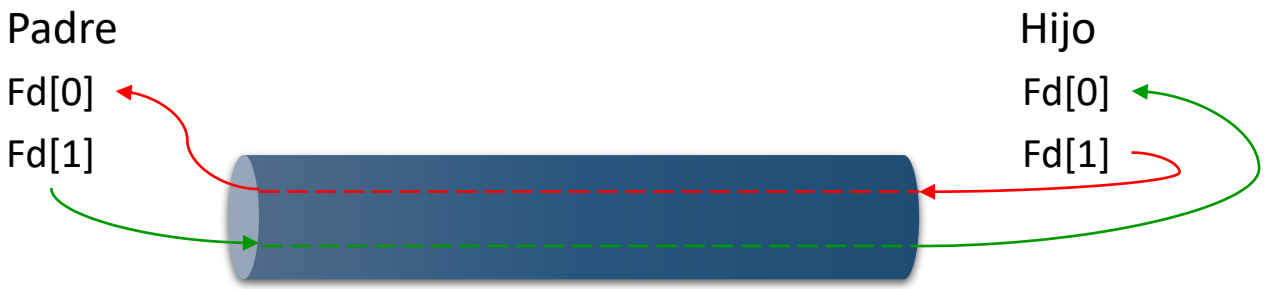
Cuestiones:

- Comunicación: unidireccional o bidireccional?
- Comunicación bidireccional, ¿es half o full-duplex?
- ¿Debe existir una relación (es decir, padre-hijo) entre los procesos de comunicación?

Pipe ordinario: no se puede acceder desde fuera del proceso que lo creó. Normalmente, un proceso padre crea un pipe y lo usa para comunicarse con un proceso hijo que creó.

COMUNICACIÓN ENTRE PROCESOS - PIPE

- Son utilizados para comunicaciones unidireccionales
- Permiten la comunicación en el estilo estándar de productor-consumidor
- El productor escribe en un extremo (el final de la tubería)
- El consumidor lee desde el otro extremo (el extremo de lectura de la tubería)
- Requieren una relación padre-hijo entre los procesos de comunicación.



COMUNICACIÓN ENTRE PROCESOS - PIPE

- Crear un pipe

```
int pfd[2];
pipe(pfd);
```

pdf[1]
escribe

pdf[0]
lee

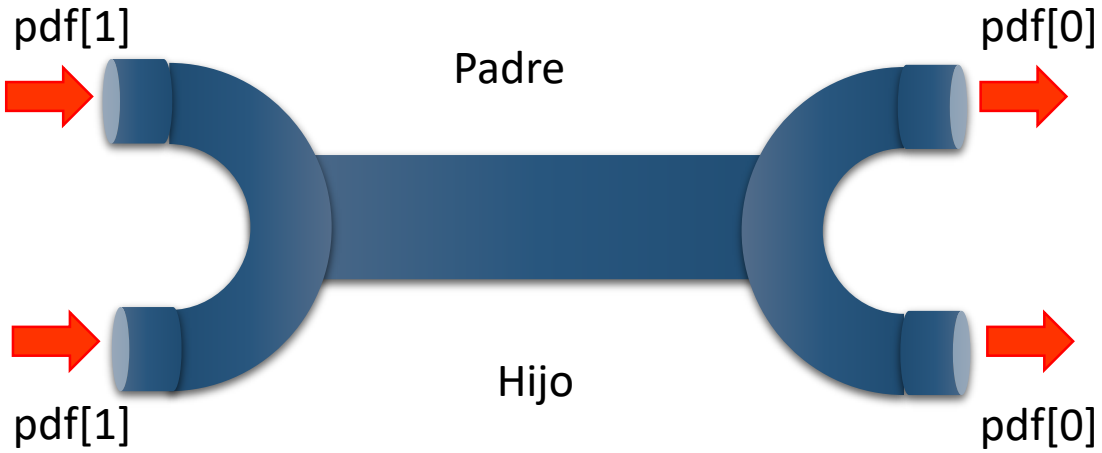
- Operaciones

```
write(pfd[1], buf, size);
read(pfd[0], buf, SIZE);
```

COMUNICACIÓN ENTRE PROCESOS - PIPE

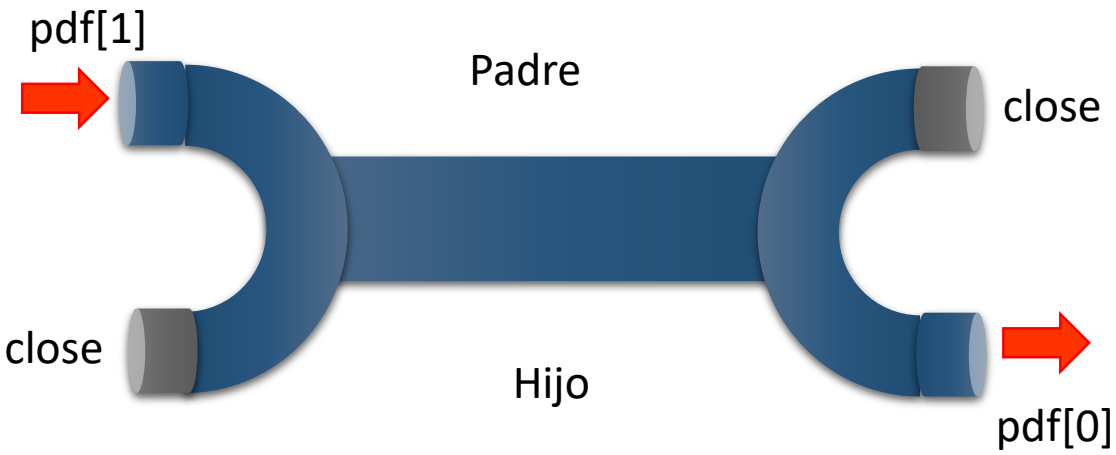
- Antes de realizar un fork

- Después de realizar el fork



COMUNICACIÓN ENTRE PROCESOS - PIPE

- Para comportamientos predecibles se debe tener un único punto de entrada y un único punto de salida



Bibliografía:

- Silberschatz, A., Gagne G., y Galvin, P.B.; "Operating System Concepts", 7ma Edición 2009; 9na Edición 2012; 10ma Edición 2018.
- Stallings, W. "Operating Systems: Internals and Design Principles", Prentice Hall, 5ta Edición 2005; 6ta Edición 2009; 7ma Edición 2011; 9na Edición 2018.
- Tanenbaum, A.; "Modern Operating Systems", Addison-Wesley, 3ra. Edición 2008, 4ta. Edición 2014.