

SISTEMAS OPERATIVOS: PROCESOS

Planificación de procesos

ADVERTENCIA

2

- Este material es un simple guión de la clase: no son los apuntes de la asignatura.
- El conocimiento exclusivo de este material no garantiza que el alumno pueda alcanzar los objetivos de la asignatura.
- Se recomienda que el alumno utilice los materiales complementarios propuestos.

fork = Crea procesos idénticos al padre

exec = Ejecuta un programa en ese proceso, el proceso recibe toda su info.

wait = bloquea la ejecución hasta que muere el proceso hijo.

Devuelve el identificador del proceso que ha finalizado.

En **status** se almacena el valor que devolvió el hijo en la llamada **exit()**.

Si hijo **exit()**

en el padre

wait(&x)

$x = 4$ ~ la dir. para que lo rellene

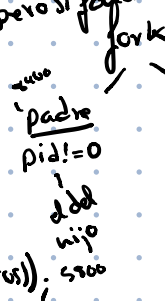
Cambia la imagen, desaparece el código anterior, pero si falla devuelve -1 y termina.

recibe como parámetro nombre fun, arg1, arg2, ..., ls, ..., -1, NULL

while (pid != wait(&status));

El padre espera a que el hijo muera para continuar la ejecución.

Se aprovecha que pid almacena la var. el pid del hijo.



este se usa para que ejecute un programa el hijo y no el padre. Cambiará todo el hijo al código de ese programa.

Cambio de contexto: Pasado ejecutar un proceso a otro, para ello se salvan los valores de los registros de ese proceso y se cargan los valores del proceso que le va a suceder.

- Almacena viejo.

- Carga nuevo.

Variables de entorno (env)

- **PATH**: Almacena la ruta en la que se está ejecutando.

Para evitar que el padre espere **sleep 20 &**, sin el **&** espera a que muera. Se queda en background.

Ctrl + ? → Congela el proceso

bg → lo envía a background

fg → lo pone en primer plano

} el congelado

Ctrl + C → Mata el proceso.

Contenido

3

- **Creación de procesos.**
- Terminación de procesos.
- Ciclo de vida de un proceso.
- Tipos de planificación.
- Algoritmos de planificación.

Creación de procesos

4

- Los SO proveen mecanismos para que los procesos puedan crear otros procesos → Llamada al sistema
- El proceso de creación se puede repetir recursivamente creándose una “estructura familiar” → Arbol de procesos
- Asignación de recursos al nuevo proceso:
 - Los obtiene directamente del SO
 - El padre debe repartir sus recursos con el proceso hijo o compartir todos o parte de ellos con él.
 - Se evita así que un proceso bloquee el sistema multiplicándose indefinidamente

Creación de procesos

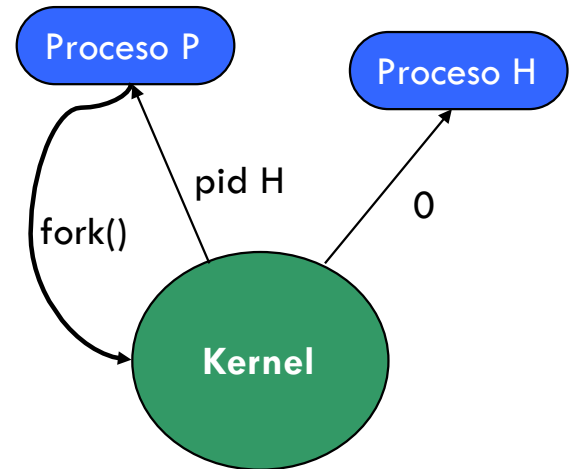
5

- Cuando se crea un proceso:
 - En términos de ejecución
 - El padre continua ejecutándose en paralelo con su/s hijo/s
 - El padre espera a que alguno o todos sus hijos hayan terminado
 - En términos del espacio en memoria
 - El proceso hijo es un clon del proceso padre
 - El proceso hijo tiene ya un programa cargado en memoria

Creación de procesos en UNIX

6

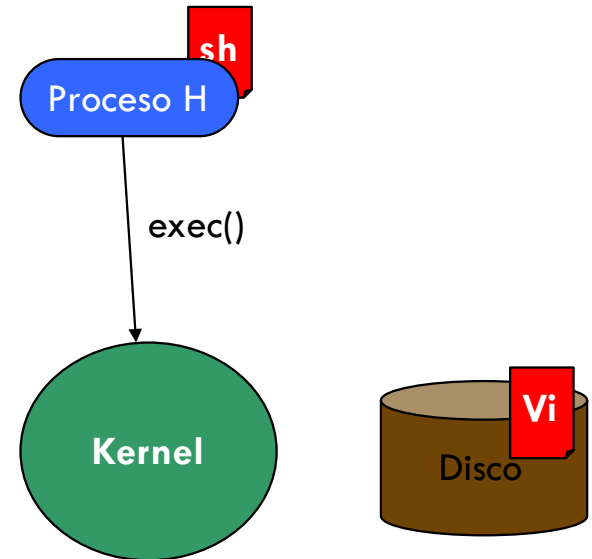
- En la familia Unix se distingue entre crear procesos y ejecutar nuevos programas.
- La llamada al sistema para crear un nuevo proceso se denomina **fork()**. *No cambia la imagen de memoria, y las modif. no afectan al padre.*
- Esta llamada crea una copia casi idéntica del proceso padre
 - Ambos procesos, padre e hijo, continúan ejecutándose en paralelo
 - El padre obtiene como resultado de la llamada a `fork()` el pid del hijo y el hijo obtiene 0
 - Algunos recursos no se heredan (p.ej. señales pendientes)



Creación de procesos en UNIX

7

- El proceso hijo puede invocar la llamada al sistema exec*()
 - sustituye su imagen en memoria por la de un programa diferente
- El padre puede dedicarse a crear más hijos, o esperar a que termine el hijo
 - wait() lo saca de la cola de “listos” hasta que el hijo termina



Jerarquía de procesos (pstree)

8

```
ssh.arcos.inf.uc3m.es - ssharcos - SSH Secure Shell
File Edit View Window Help
Quick Connect Profiles

jdaniel@ssh:~$ pstree
init--+-apache2---10*[apache2]
      |-aprsd---aprsd---14*[aprsd]
      |-atd
      |-cron
      |-events/0
      |-6*[getty]
      |-gmond---gmond---6*[gmond]
      |-inetd
      |-khelper
      |-klogd
      |-kssoftirqd/0
      |-kthreadd--+-ata/0
                  |-ata_aux
                  |-kblockd/0
                  |-kjournald
                  |-kamisrord
                  |-kseriod
                  |-kswapd0
                  |-2*[pdflush]
                  |-rpciod/0
                  |-xenbus
                  |-xenwatch
```

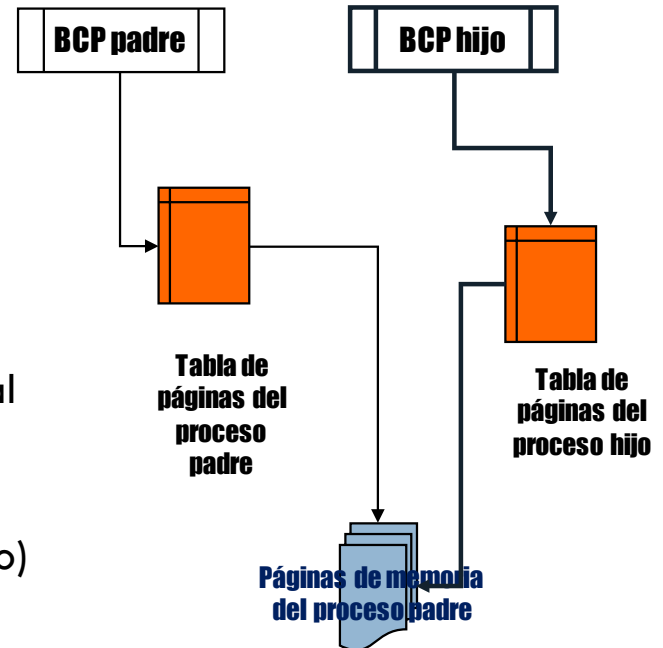
Sistemas Operativos – Planificación de procesos

Connected to ssh.arcos.inf.uc3m.es SSH2 - aes128-cbc - hmac-md5 - none 80x24

Creación de procesos: Copy on Write (COW)

9

- Ineficiencias del modelo *fork()*
 - Se copian muchos datos que podrían compartirse
 - Si al final se carga otra imagen, todavía es peor porque todo lo copiado se deshecha
- Muchos UNIX usan COW
 - Copy-on-Write es una técnica que retrasa o evita la copia de los datos al hacer el fork
 - Los datos se marcan de manera que si se intentan modificar se realiza una copia para cada proceso (padre e hijo)
 - Ahora *fork()* sólo copia la tabla de páginas del padre (no las páginas) y crea un nuevo BCP para el hijo

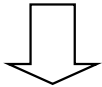


Ejemplo de compartición para evitar duplicar datos

Creación de procesos en Linux

10

fork:



*“Copia al proceso padre
y le da una nueva
identidad al hijo”*

*Mapa de
Memoria*

*Ambos listos para ejecutar,
no importa que sea el padre o el hijo
el que ejecute, puede ser cualquier proceso*

Obtener entrada libre en
Tabla de Procesos

Copiar BCP del padre

Duplicar mapa de
M del padre
(incluyendo pilas)

Estado \leftarrow listo
PCB en cola de listos

También limpiar
señales, eventos
...

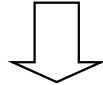
Devuelve PID
al padre

Devuelve 0 al
hijo

Creación de procesos en Linux

11

exec :



“Cambia la imagen de M de un proceso usando como “recipiente” uno previo”

Los datos de pila los pone el SO, según los pasados por exec.

Solo mantiene los descriptores abiertos del que procede.

Sistemas Operativos – Planificación de procesos

Liberar imagen de M del proceso

Leer ejecutable

Crear nueva imagen M → BCP

Cargar secciones .texto y .datos

Crear pila inicial U
Crear pila S: dir. principio programa

Inicio BCP: regs.;
PC ← dir SO: RETI

Contenido

12

- Creación de procesos.
- **Terminación de procesos.**
- Ciclo de vida de un proceso.
- Tipos de planificación.
- Algoritmos de planificación.

Terminación de procesos

13

- Cuando un proceso termina todos los recursos asignados son liberados: *Excepto el valor del exit, que se queda en B CP esperando al wait del padre, pero si el padre muere el init hace wait e aborta. Lo simple hacer zombie.*
 - memoria, ficheros abiertos, entradas en tablas,...
- y el kernel notifica al proceso padre el evento.

- Un proceso puede terminar de 2 formas:
 - Voluntariamente: Llamada al sistema `exit()`
 - Involuntariamente:
 - Excepciones: división por cero, violación de segmento
 - Abortado por el usuario (`ctrl-c`) u otro proceso (`kill`), es decir, señales que no puede manejar o ignorar

Terminación de procesos

14

- Cuando un proceso termina pueden suceder dos cosas:
 - ▣ Sus hijos no se ven afectados
 - ▣ Todos los hijos acaban también → **terminación en cascada (Ej. VMS)**

- En Unix,
 - ▣ los hijos del proceso terminado pasan a depender del proceso *init*
 - ▣ el proceso finalizado pasa a estado Zombie hasta que el proceso padre recoge su código de finalización

¿Cuándo se elimina el BCP?

15

- Las terminación de un proceso y la eliminación de su BCP son tareas diferenciadas
 - Cuando el padre obtiene la información del hijo, se procede a eliminar las estructuras de datos
 - Llamada al sistema *wait()*
 - Bloquea al proceso hasta que termina el/un hijo
 - Devuelve el pid del hijo finalizado y

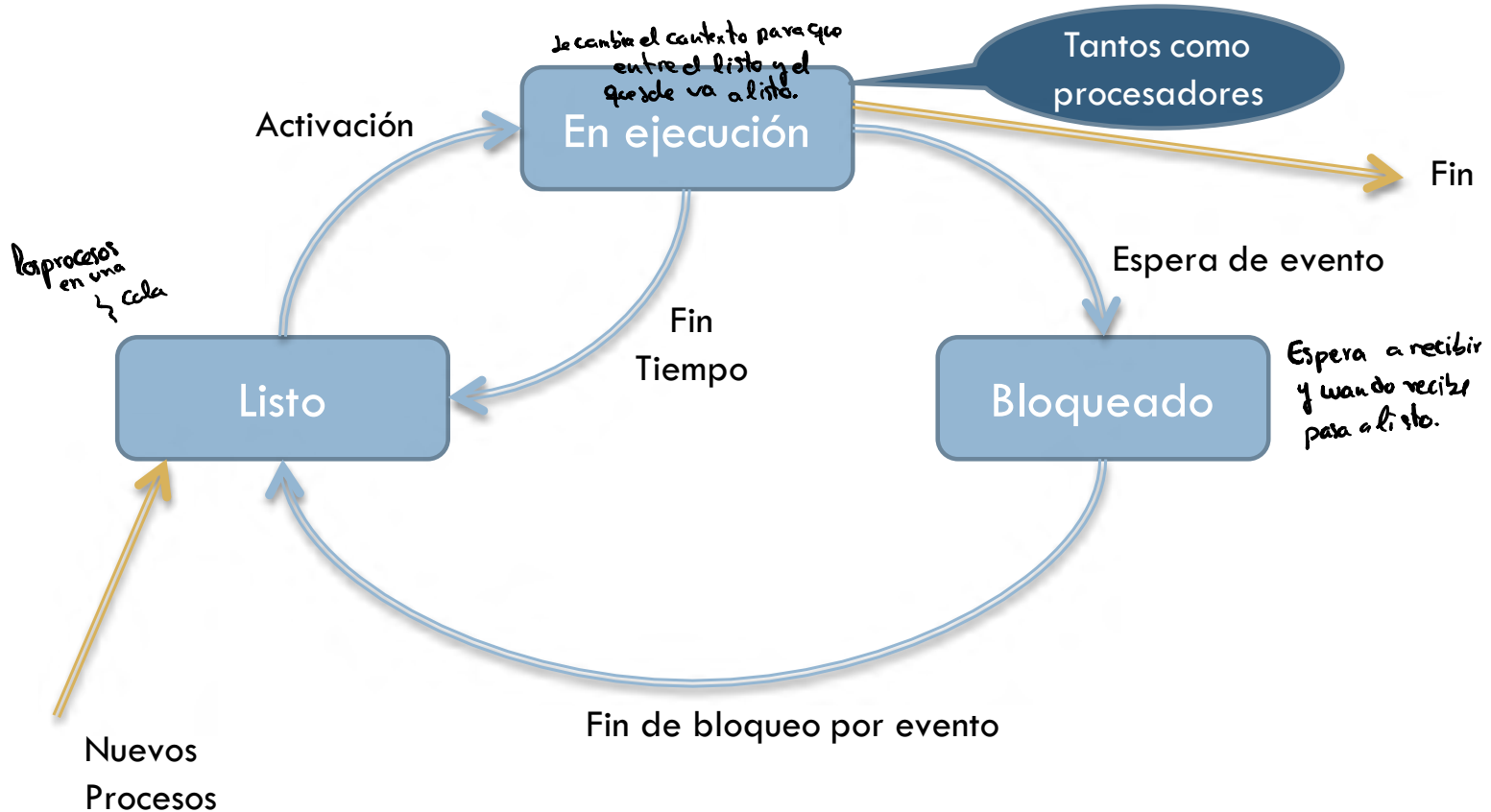
Contenido

16

- Creación de procesos.
- Terminación de procesos.
- **Ciclo de vida de un proceso.**
- Tipos de planificación.
- Algoritmos de planificación.

Ciclo de vida básico de un proceso

17



Expulsión al disco (swap)

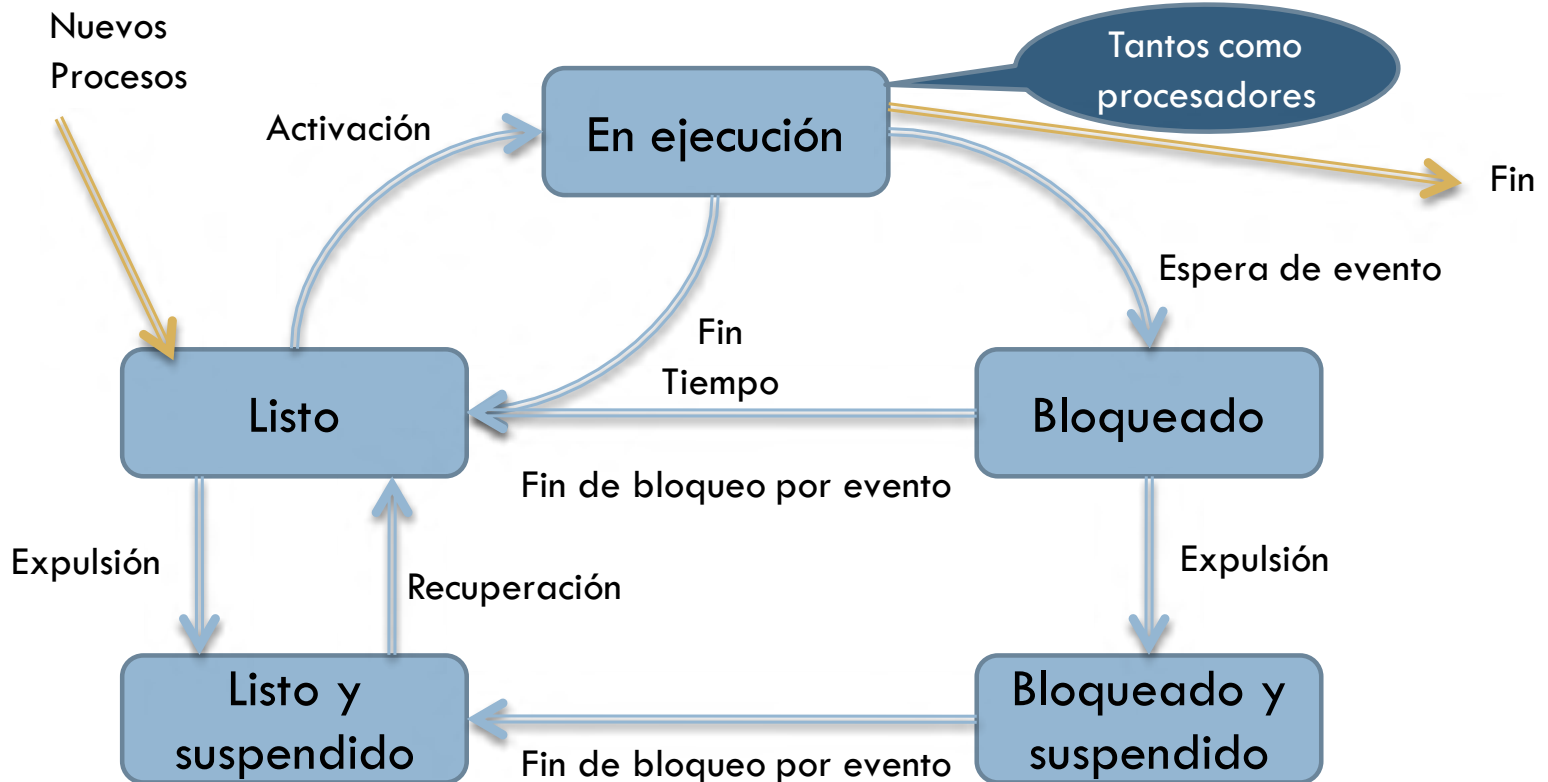
18

- Cuando existen muchos procesos en ejecución el rendimiento puede bajar por excesiva paginación.
 - ▣ Solución: El Sistema Operativo puede expulsar totalmente procesos al área de intercambio del disco.
- Introduce nuevos estados de los procesos.
 - ▣ Bloqueado y suspendido.
 - ▣ Listo y suspendido.

Cuando se satura, en swap se pasan a disco los bloqueados para liberar memoria. Y cuando se quiere seguir ejecutando se pasa de nuevo a memoria.

Ciclo de vida básico de un proceso

19



Contenido

20

- Creación de procesos.
- Terminación de procesos.
- Ciclo de vida de un proceso.
- **Tipos de planificación.**
- Algoritmos de planificación.

Niveles de planificación

21

- Planificación a corto plazo
 - ▣ Selecciona el siguiente proceso a ejecutar.
- Planificación a medio plazo
 - ▣ Selecciona qué procesos se añaden o se retiran (expulsión a swap) de memoria principal.
- Planificación a largo plazo
 - ▣ Realiza el control de admisión de procesos a ejecutar.
 - ▣ Muy usada en sistemas batch.

Tipos de planificación

22

□ No apropiativa.

- El proceso en ejecución **conserva el uso de la CPU** mientras lo desee.

↳ No se puede expulsar hasta que termine

□ Apropiativa.

- El sistema operativo puede expulsar a un proceso de la CPU.

↳ Se puede expulsar, para que pase a ejecutar otro

Puntos de decisión de planificación

23

- Momentos en los que se puede decidir la planificación de un proceso:
 1. Cuando un proceso se bloquea en espera de un evento
 - Realización de una llamada al sistema.
 2. Cuando se produce una interrupción.
 - Interrupción del reloj.
 - Interrupción de fin de E/S.
 3. Fin de proceso.
- Planificación no apropiativa: 1 y 3.
 - Windows95, MacOS anteriores a versión 8.
- Planificación apropiativa: 1, 2 y 3.

Colas de procesos

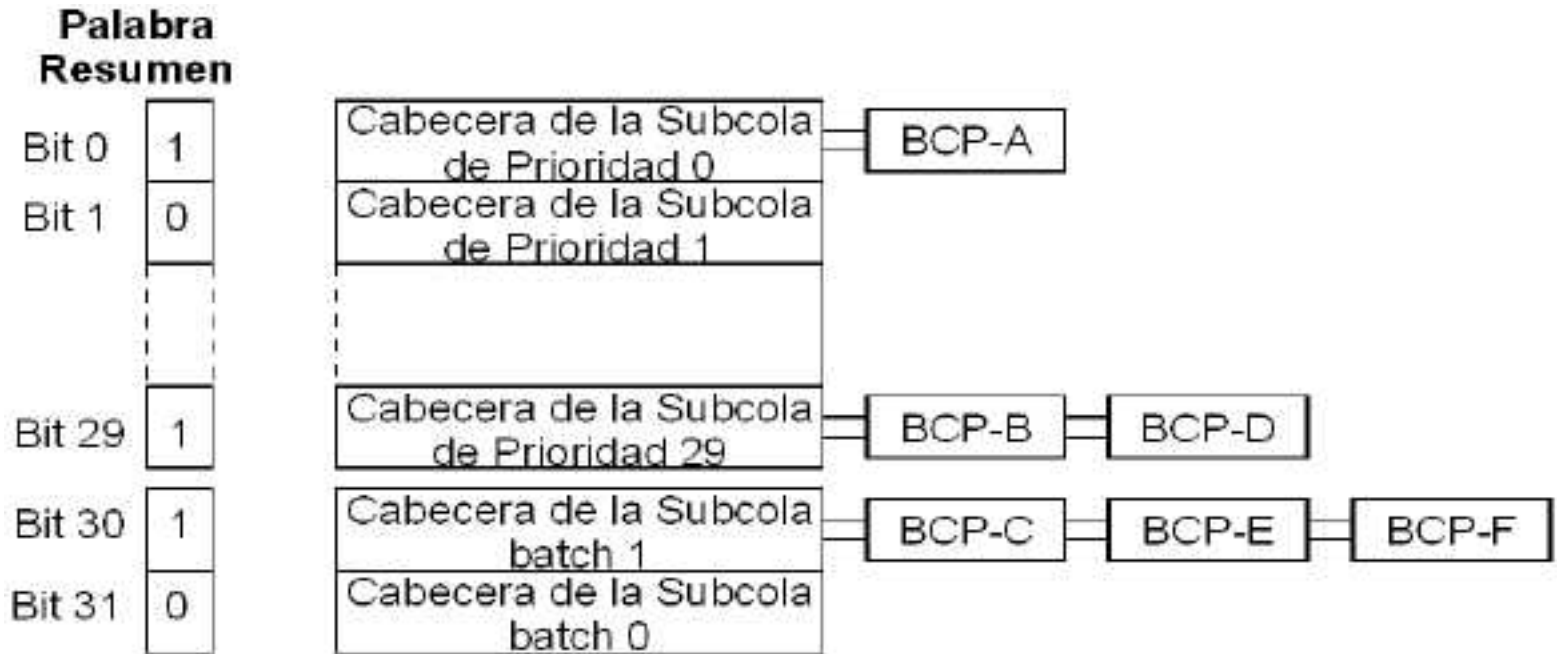
24

- Los procesos listos para ejecutar se mantienen en una cola.

- Alternativas:
 - Cola única.
 - Colas por tipos de procesos.
 - Colas por prioridades.

Colas de procesos

25



Colas de procesos: Implementación

26

- El SO mantiene diversas colas de procesos.
- Se implementa con punteros internos al BCP.
- Acceso eficiente.

Tabla de procesos

BCP1	BCP2	BCP3	BCP4	BCP5	BCP6	BCP7	BCP8	BCP9	BCP10	BCP11	BCP12
0	7		6	1	11	5	0	8		9	

2 4

Punteros de las colas

Contenido

27

- Creación de procesos.
- Terminación de procesos.
- Ciclo de vida de un proceso.
- Tipos de planificación.
- **Algoritmos de planificación.**

Planificación: Medidas

28

↳ Siempre busca maximizar algo:

- Utilización de CPU:
 - ▣ Porcentaje de tiempo que se usa la CPU.
 - ▣ Objetivo: Maximizar.
- Productividad: Mayor nº de programas por unidad de tiempo.
 - ▣ Número de trabajos terminados por unidad de tiempo.
 - ▣ Objetivo: Maximizar.
- Tiempo de retorno (T_q) : Tiempo desde que entra hasta que sale.
 - ▣ Tiempo que está un proceso en el sistema. Instante final (T_f) menos instante inicial (T_i).
 - ▣ Objetivo: Minimizar.

Planificación: Medidas

29

- **Tiempo de servicio (T_s):**
 - ▣ Tiempo dedicado a tareas productivas (cpu, entrada/salida). $T_s = T_{CPU} + T_{E/S}$
- **Tiempo de espera (T_e):**
 - ▣ Tiempo que un proceso pasa en colas de espera.
 $T_e = T_q - T_s$
- **Tiempo de retorno normalizado (T_n):** Porcentaje de tiempo que un proceso ha estado esperando.
 - ▣ Razón entre tiempo de retorno y tiempo de servicio.
 $T_n = T_q / T_s$
 - ▣ Indica el retardo experimentado.

Asignación FCFS

30

- **First to Come First to Serve:** Primer en llegar primero en servir. (FIFO)

- Algoritmo **no apropiativo.** ~ no se puede expulsar un proceso

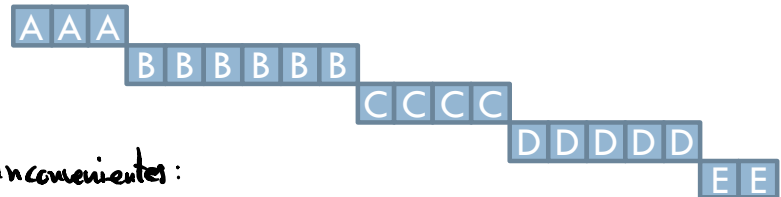
- **Penaliza** a los **procesos cortos.**

Se usa en servidores.

Lo Tienen que esperar aunque no tardarian apenas.

Ventajas:

Proceso	Llegada	Servicio
A	0	3
B	2	6
C	4	4
D	6	5
E	8	2



Inconvenientes:

FCFS: Tiempo de retorno normalizado

31

Proceso	Llegada	Servicio	Inicio	Fin	Retorno	Espera	Retorno normalizado
A	0	3	0	3	3	0	$3/3=1$
B	2	6	3	9	7	1	$7/6=1.16$
C	4	4	9	13	9	5	$9/4=1.25$
D	6	5	13	18	12	7	$12/5=2.4$
E	8	2	18	20	12	10	$12/2=6$

- Tiempo medio de espera: 4.6
- Tiempo medio de retorno normalizado: 2.5

Asignación SJF

32

- *Shortest Job First*: Primero el trabajo más corto.
- Algoritmo no apropiativo.
- Selecciona el trabajo más corto.
- Solamente se puede aplicar si se conoce de antemano la duración de cada trabajo.
- Posibilidad de inanición:
 - ▣ Si continuamente llegan trabajos cortos, los trabajos largos nunca llegan a ejecutarse.

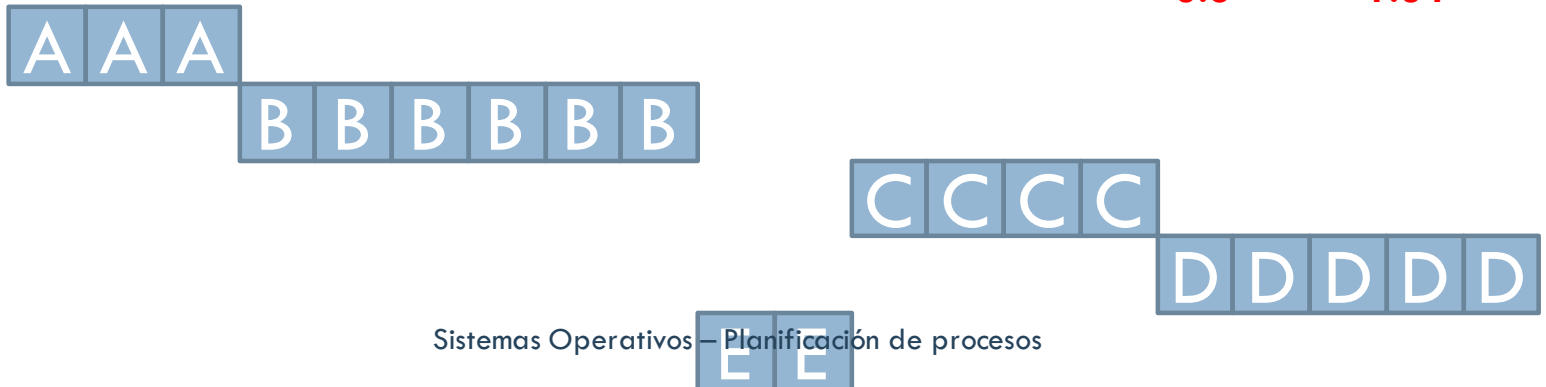
Asignación SJF

33

Proceso	Llegada	Servicio	Inicio	Fin	Retorno	Espera	Retorno normalizado
A	0	3	0	3	3	0	$3/3=1$
B	2	6	3	9	7	1	$7/6=1.16$
C	4	4	11	15	11	7	$11/4=2.75$
D	6	5	15	20	14	9	$14/5=2.8$
E	8	2	9	11	3	1	$3/2=1.5$

3.6

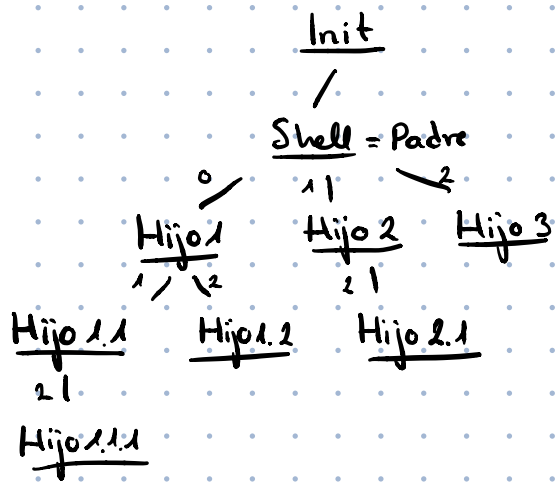
1.84



Cíclico o Round-Robin

34

- Mantiene una cola FIFO con los procesos listos para ser ejecutados.
- Un proceso recibe el procesador durante un cuanto o rodaja de tiempo. *no cuando la agota pasa a la cola y se ejecuta la siguiente. Van ejecutándose un poco cada una, hasta que se completan.*
- Un proceso regresa a la cola listos cuando:
 - ▣ Expira su rodaja de tiempo.
 - ▣ Se produce el evento que lo llevó a la cola de bloqueados.
- Un proceso pasa a la cola de bloqueados cuando:
 - ▣ Pasa a esperar un evento.
- Algoritmo apropiativo.
- Se debe tener en cuenta que cada cambio de contexto genera retraso.
 - ▣ Rodaja de tiempo >> tiempo para cambio de contexto



Round-Robin (q=1)

35

Proceso	Llegada	Servicio	Inicio	Fin	Retorno	Espera	Retorno normalizado
A	0	3	0	4	4	1	$4/3=1.33$
B	2	6	2	18	16	10	$16/6=2.66$
C	4	4	5	17	13	9	$13/4=3.25$
D	6	5	7	20	14	9	$14/5=2.8$
E	8	2	10	15	7	5	$7/2=3.5$



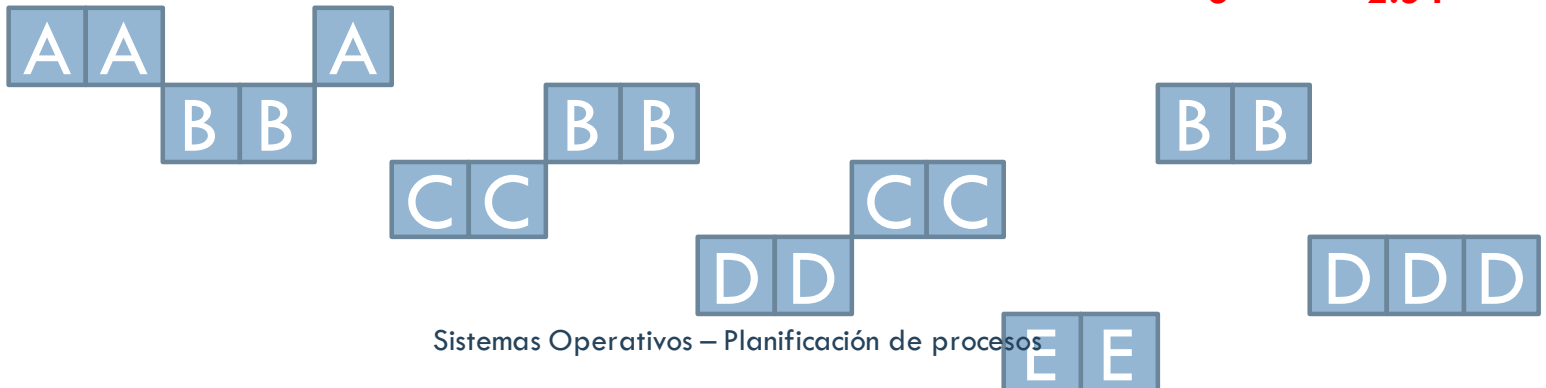
Round-Robin (q=2)

36

Proceso	Llegada	Servicio	Inicio	Fin	Retorno	Espera	Retorno normalizado
A	0	3	0	5	4	1	$4/3=1.33$
B	2	6	2	17	16	10	$16/6=2.66$
C	4	4	5	13	13	9	$13/4=3.25$
D	6	5	9	20	14	9	$14/5=2.8$
E	8	2	13	15	7	5	$7/2=3.5$

6

2.54



Round-Robin (q=4)

37

Proceso	Llegada	Servicio	Inicio	Fin	Retorno	Espera	Retorno normalizado
A	0	3	0	3	3	0	$3/3=1$
B	2	6	3	17	15	9	$15/6=2.5$
C	4	4	7	11	7	3	$7/4=1.75$
D	6	5	11	20	14	9	$14/5=2.8$
E	8	2	17	19	11	9	$11/2=5.5$

6

2.71



X Asignación por prioridades

38

- Cada proceso tiene una prioridad asignada.
- Se selecciona primero los procesos más prioritarios.

- Alternativas:
 - Prioridades fijas → problema de inanición.
 - Solución: mecanismos de envejecimiento.

X Planificación en Windows

39

- Principales características:
 - ▣ Basado en prioridades y uso de cuantos de tiempo.
 - ▣ Planificación apropiativa.
 - ▣ Planificación con afinidad de procesador.
- Planificación por hilos y no por procesos.
- Un hilo puede perder el procesador si hay otro más prioritario que esté listo.
- Decisiones de planificación:
 - ▣ Hilos nuevos → Listo.
 - ▣ Hilos bloqueados que reciben evento → Listo.
 - ▣ Hilo deja del procesador si termina cuanto, finaliza o pasa a bloqueado.

Puntos a recordar

40

- ❑ La creación de un proceso implica la creación de su imagen de memoria y de su BCP.
- ❑ Un proceso pasa por distintos estados durante su ejecución.
- ❑ El sistema operativo realiza la planificación de los procesos.
- ❑ La planificación puede ser apropiativa y no apropiativa.
- ❑ Los distintos algoritmos de planificación de procesos pueden favorecer más o menos a un tipo de procesos.
- ❑ Los sistemas operativos modernos usan planificación apropiativa.

Lecturas recomendadas

41

Básica

- Carretero 2007:
 - ▣ 3.3 Información del proceso.
 - ▣ 3.4 Vida de un proceso.

Complementaria

- Stallings 2005:
 - ▣ 3.2 Estados de los procesos.
 - ▣ 3.3 Descripción de procesos.
 - ▣ 3.5 UNIX SVR4 Process Management.
- Silberschatz 2006:
 - ▣ 3.1 Concepto de proceso.
 - ▣ 3.3 Operaciones sobre los procesos.