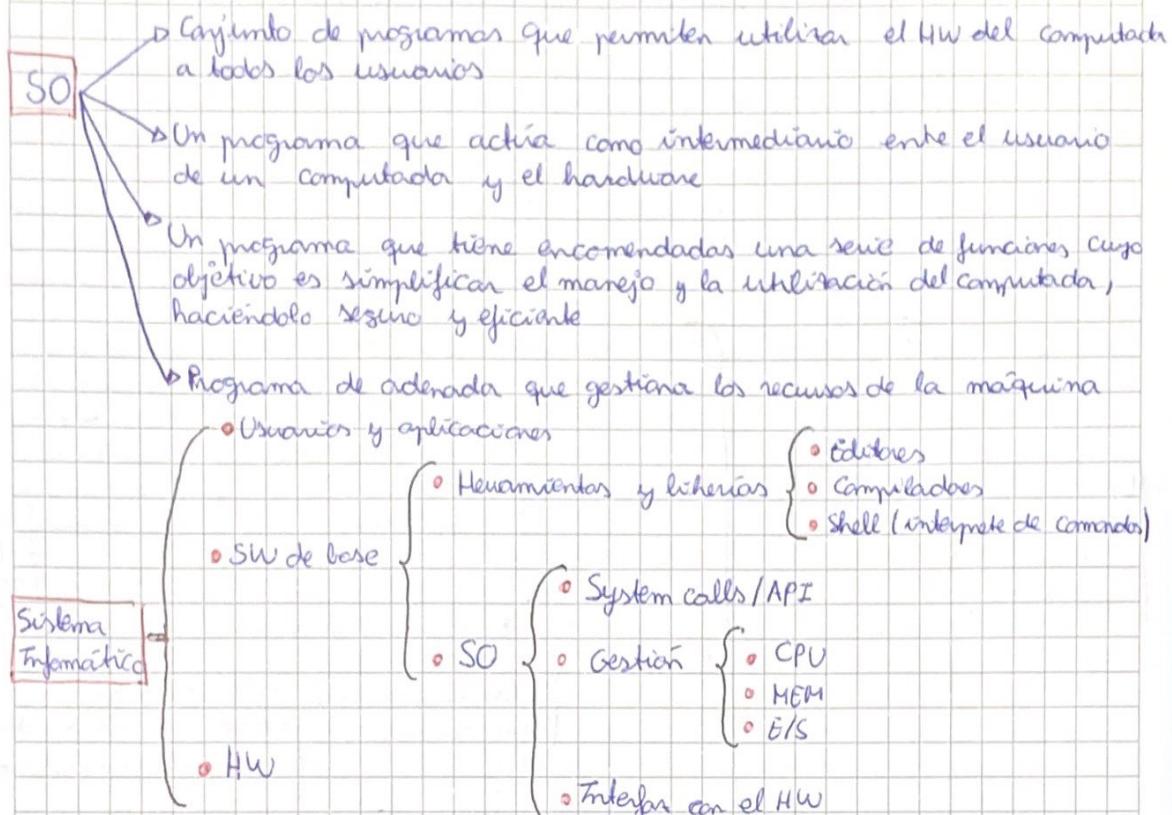


## Tema 1: Fundamentos de los SO



### Objetivos del SO:

\* Una máquina "amigable"

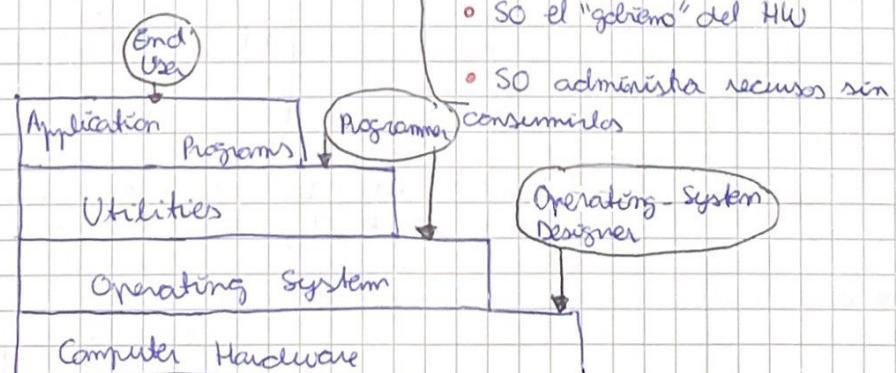
\* Eficiencia HW del computador

### Visiones del SO

• Descendente: Máquina extendida

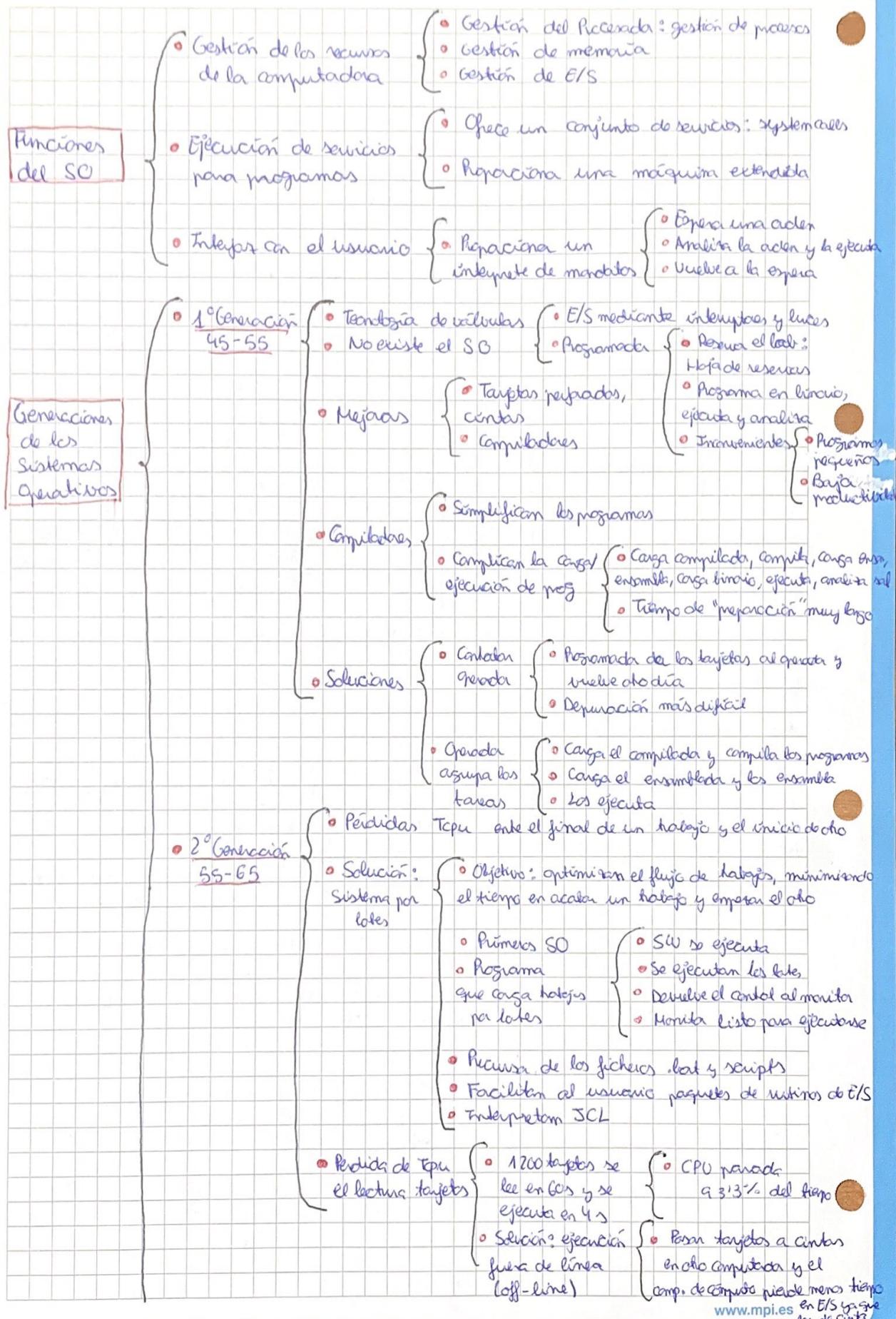
• Ascendente: Gestión del HW

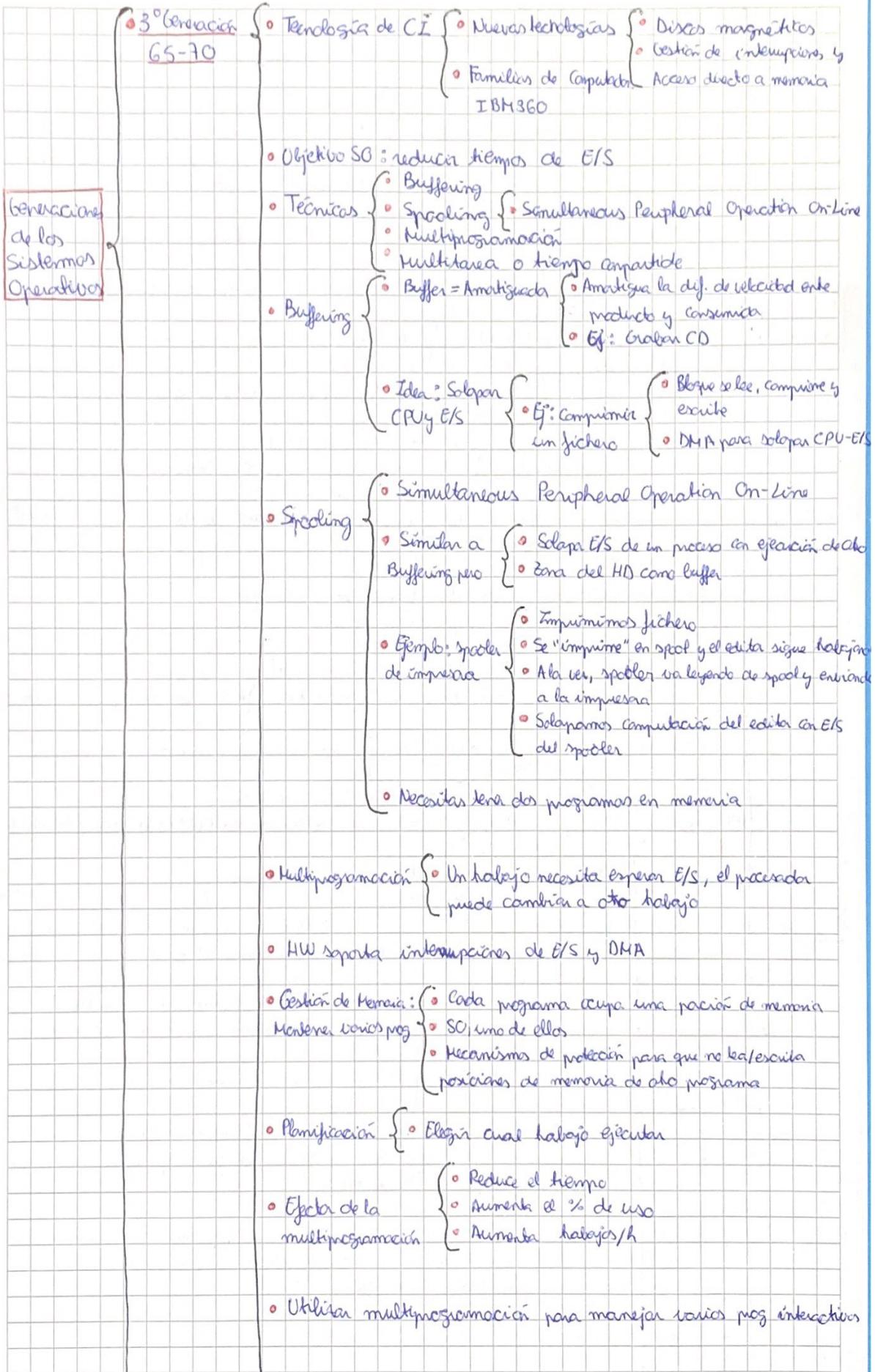
### Niveles del Computador



- Descentraliza: máquina fácil de usar
- SO hace el uso cómodo del HW
- Sin SO programaríamos la máquina desde cero

- Miran desde HW
- SW gestiona y administra el HW
- SO es el "gobierno" del HW
- SO administra recursos sin consumirlos







# WUOLAH + #QuédateEnCasa

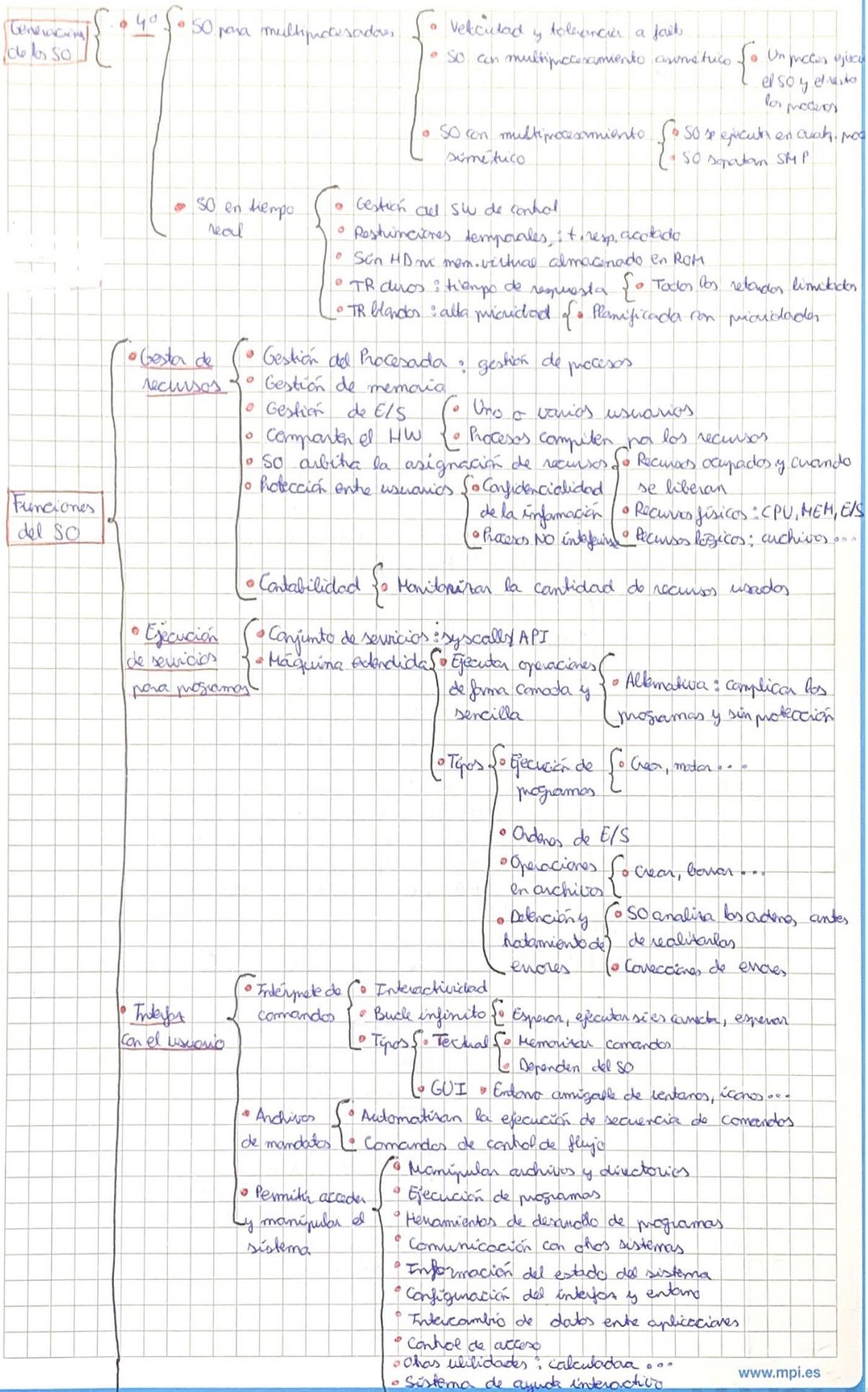
#KeepCalm #EstudiaUnPoquito

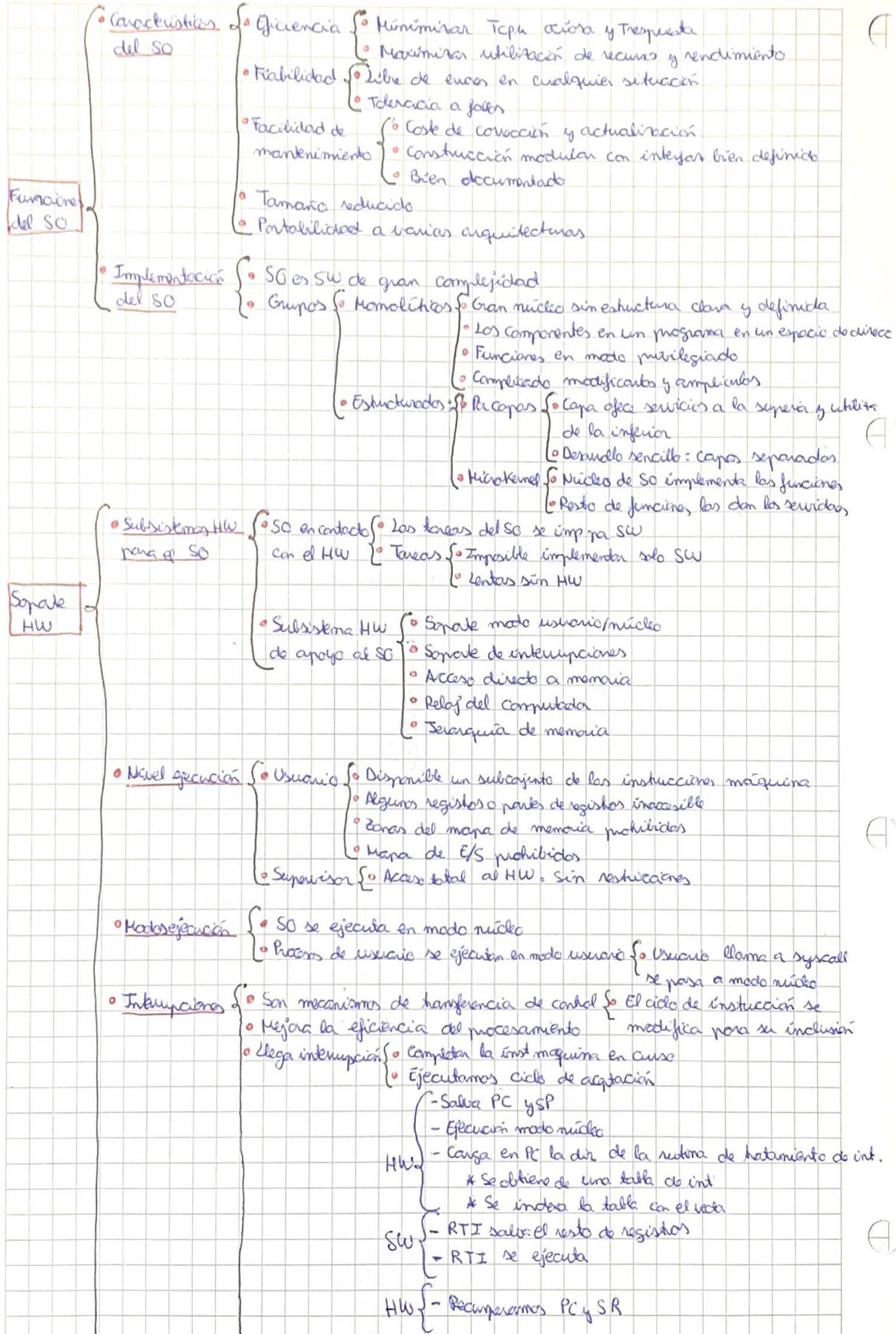
Ahora más que nunca **anima al resto de tus compañeros** subiendo a redes sociales **este cartel** que hemos puesto entre **tus apuntes**.  
Hay días que es más difícil estudiar, pero tú **ya lo estás haciendo**.



Cervantes\*

- 3º Mejorar la interactividad
- La lotes
    - Usuario no interactúa con el programa
    - Dificulta la depuración de programas
  - Interactuar con la máquina
    - Aprovech el interfaz comando y sistema de ficheros
    - Usuaria actua segun los resultados anteriores
    - Interactividad en tiempo de respuesta breve
    - Maquina cara para un solo usuario
  - Interactividad a bajo coste
    - Varios terminales para varios usuarios. Comparten entre si
    - Uso intérn CPU: "multiplexa en el tiempo" la ejecución de los procesos
  - Multiprogramación por lotes
    - Objetivo
      - Maximizar el uso del procesador
    - Fuentes de directivas al SO
      - Comandos del lenguaje de control de trabajos proporcionados con el hardware
  - Tiempo Compartido
    - Objetivo
      - Minimizar el tiempo de respuesta
    - Fuentes de directivas al SO
      - Comandos introducidos desde el terminal
  - Importancia del subsistema del SO para planificación
    - autenticación
    - moleción
  - Sistema multiusuario
    - Para lotes se siguen usando dentro de sist. multitarea
    - Grandes trámites sin intervención
  - Multiusuario suele ser multitarea
    - prog reservas acuerdos (NO SIEMPRE)
  - 4º Generación 70-Hoy
    - Microprocesadores
      - IBM-PC salió al mercado en 1981 con el DOS 1.0 y MS-DOS
    - SO para PC
      - No multitarea ni multiusuario
        - No protección, malos virus MS-DOS
      - Entornos GUI
        - Windows 3.11 es multitareas pero monousuario
      - Redes propietaria
        - Proveedores distribuidos: comparten PCs y perifericas
        - Multitarea y soporte multiusuario
      - BD sustituyen al sist. de archivos
    - SO distribuidos
      - SO sólo una red de computadoras
      - Proporciona la visión de un computadora más potente
      - Ej: Nach, Amoeba.
    - Middlewares
      - Una capa SW ejecutada sobre una red de computadoras
      - Cada computadora tiene su SO convencional
      - Ej: Oracle, BEA





Soporte HW

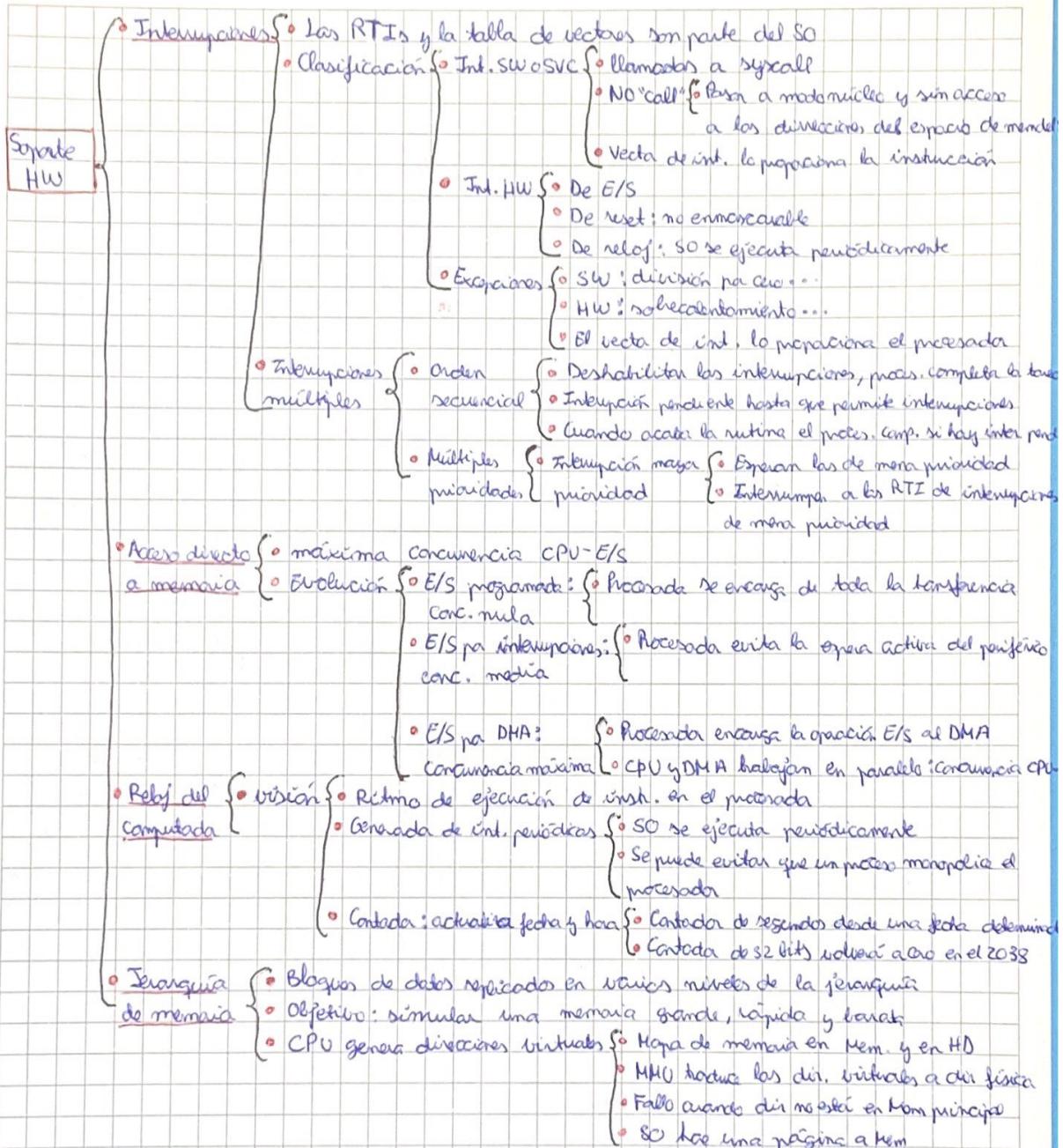
- Subsistemas HW para el SO
  - SG en contacto con el HW
  - Tareas
    - Los tareas del SG se limpian SW con el HW
    - Tareas
    - Imposible implementar solo SW
    - Tareas sin HW
- Subsistema HW de apoyo al SG
  - Soporte modo usuario/núcleo de apoyo al SG
    - Soporte de interrupciones
    - Acceso directo a memoria
    - Reloj del computadora
    - Secuencia de memoria

- Nivel ejecución
  - Usuario
    - Disponible un subconjunto de las instrucciones máquina
    - Algunos registros o partes de registros inaccesible
    - Zonas del mapa de memoria prohibidas
    - Mapa de E/S prohibidos
  - Supervisor
    - Acceso total al HW, sin restricciones

- Modo ejecución
  - SO se ejecuta en modo núcleo
  - Procesos de usuario se ejecutan en modo usuario
    - Usuario llama a syscall
    - pasa a modo núcleo

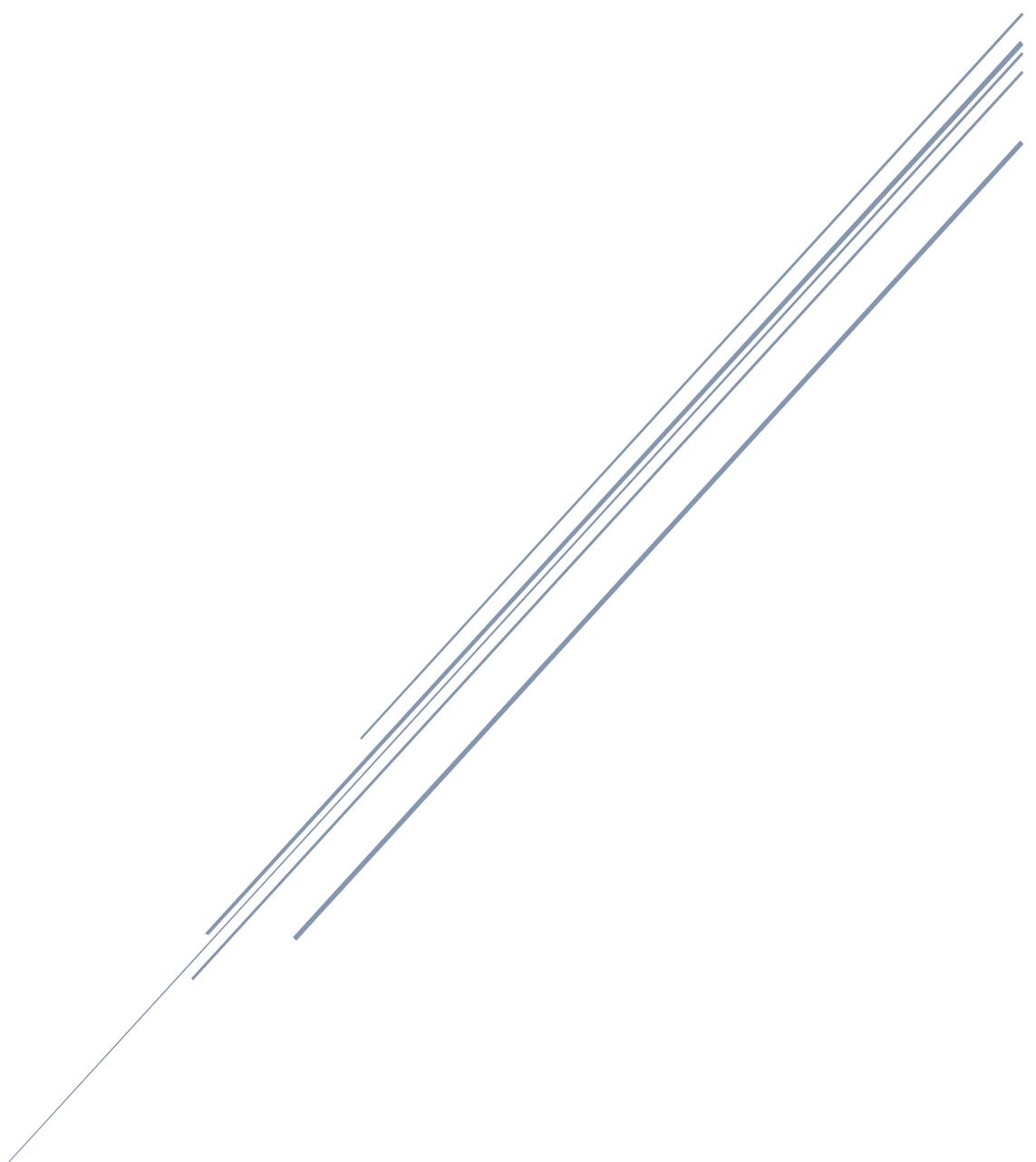
- Interrupciones
  - Son mecanismos de transferencia de control
  - El ciclo de instrucción se modifica para su inclusión
  - Mejora la eficiencia del procesamiento
  - Llega interrupción
    - Comienza la instrucción en curso
    - Ejecutamos ciclo de captación
      - Salva PC y SP
      - Ejecución modo núcleo
      - Carga en PC la dir. de la rutina de tratamiento de int.
      - \* Se obtiene de una tabla de int.
      - \* Se indica la tabla con el vector

- HW
  - RTI salvo el resto de registros
  - RTI se ejecuta
- SW
  - Recuperaremos PC y SR
- HW
  - Recuperaremos PC y SR



# SISTEMAS OPERATIVOS

## Tema 2.1



Escuela Técnica Superior de Ingeniería Informática

## INDICE

Sistemas Operativos .....	- 2 -
Procesos .....	- 2 -
Requerimientos de procesos .....	- 2 -
Estados de procesos .....	- 3 -
Modelo de cinco estados .....	- 3 -
Procesos suspendidos .....	- 4 -
Motivos para la suspensión de procesos .....	- 4 -
Rastreo de procesos .....	- 5 -
Bloque de Control de Procesos PCB .....	- 5 -
Información de procesos .....	- 6 -
Qué define a un proceso .....	- 6 -
Organización de procesos .....	- 7 -
Cambio de procesos en la CPU .....	- 7 -
Cambio de contexto .....	- 8 -
Intercalación de procesos .....	- 8 -
Creación de procesos .....	- 9 -
Terminación de procesos .....	- 10 -
Manejo de señales .....	- 12 -
Procesos demonio .....	- 13 -
Cooperación de procesos .....	- 14 -
Desventajas de los procesos .....	- 15 -

# Sistemas Operativos

## Tema 2.1

Procesos

T2.5

Un Sistema operativo ejecuta programas.

Programa:

- Describe cómo realizar una actividad (algoritmo).
- Tiene instrucciones y valores de datos estáticos.
- Archivo estático.

Proceso:

- Un programa en ejecución.
- Una captura de un programa en ejecución.
- Una instancia de un programa en ejecución.

Un proceso es la unidad básica de ejecución en un Sistema Operativo.

Cada proceso cuenta con un número identificador, PID (Process Identifier).

Requerimientos de procesos

T2.7

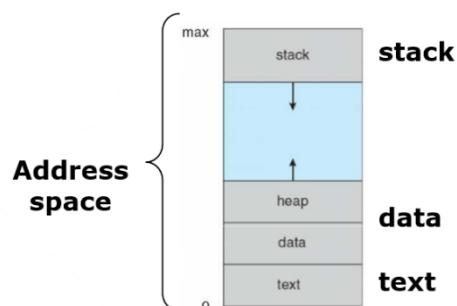
Como mínimo, la ejecución de procesos requiere los siguientes recursos:

Memoria para alojar el código del programa y los datos.

Pila la cual contenga datos temporales (parámetros de funciones, direcciones de retorno...).

Un conjunto de registros de la CPU para su ejecución.

- Estado de la CPU
  - Registros
  - Contadores de programa (PC).
  - Puntero de pila
- Estado del Sistema Operativo.

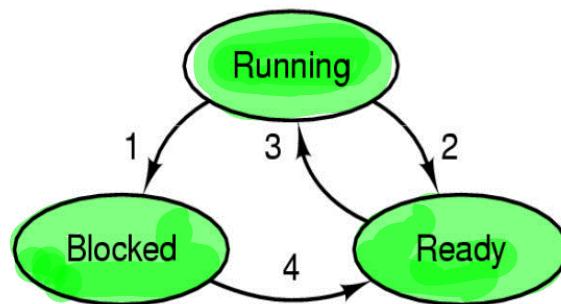


## Estados de procesos

T2.9

Cuando un proceso se ejecuta, su **estado cambia**.

- **Ejecución.** Se ejecutan instrucciones.
- **Bloqueado o en espera.** El proceso espera que ocurra algún evento.
- **Listo.** El proceso espera a ser asignado a algún procesador.



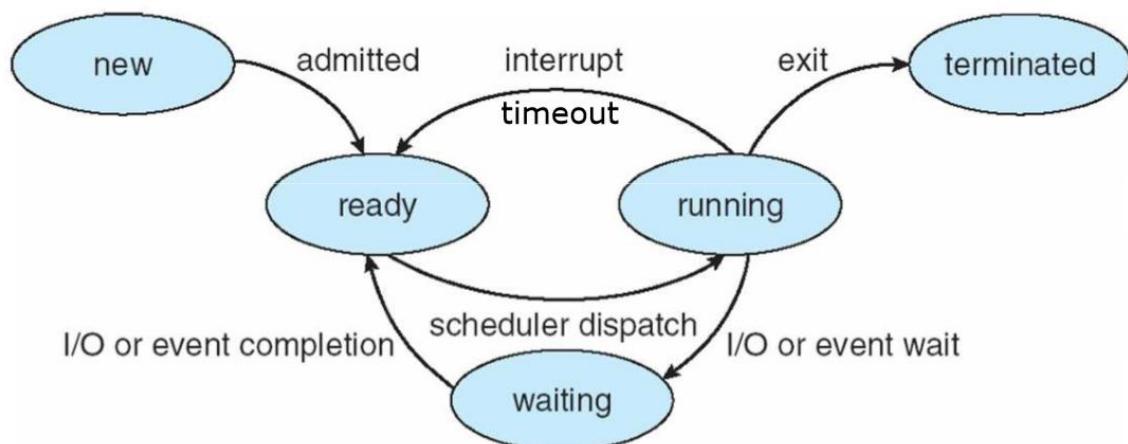
Los procesos pueden estar en **uno de dos estados**.

**Ejecución o no ejecución.**

## Modelo de cinco estados

T2.11

- **Nuevo.**
- **Ejecución.**
- **Bloqueado.**
- **Listo.**
- **Terminado.**



## Procesos suspendidos

T2.12

El procesador es más rápido que la entrada / salida así que todos los procesos podrían estar simplemente esperando esa entrada / salida, por lo que se **traslada** a estos procesos **al disco para liberar memoria**.

En consecuencia se pueden ejecutar más procesos.

## Sobrecarga del Sistema.

El estado bloqueado se convierte en suspendido cuando se transfiere al disco.

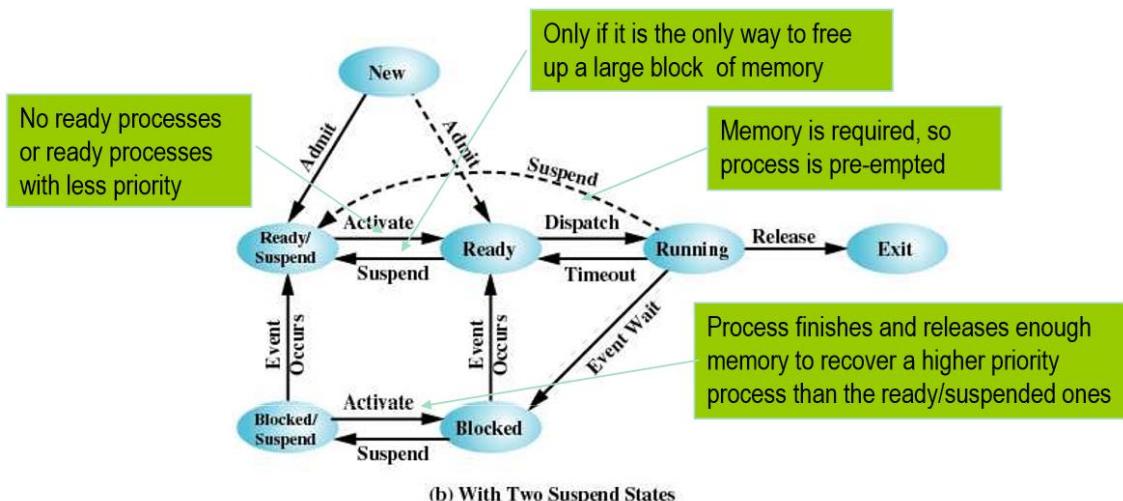
## Aparecen dos nuevos estados

- Bloqueado/Suspendido
- Listo/Suspendido

## Motivos para la suspensión de procesos

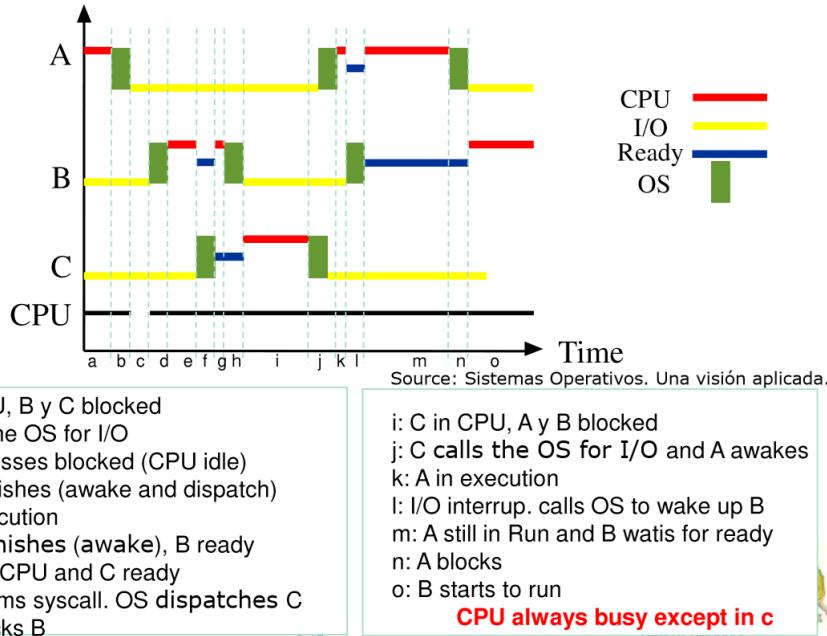
T2.13

- **Swapping**, si el SO necesita **liberar memoria** para ejecutar un proceso listo.
- **Seguridad**, el SO puede suspender un proceso por sospechas de que cause problemas.
- **Petición del usuario**, el usuario puede querer suspender la ejecución por ejemplo para depuración.
- **Timing**, procesos que se ejecutan de manera periódica y se suspenden esperando la próxima ejecución.
- **Otros motivos**.



## Rastreo de procesos

T2.15



## Bloque de Control de Procesos PCB

T2.17

*Process Control Block*

Cuando el procesador cambia de proceso o suspende un proceso, debe guardar información sobre él y el estado de la CPU para poder continuar por donde se dejó.

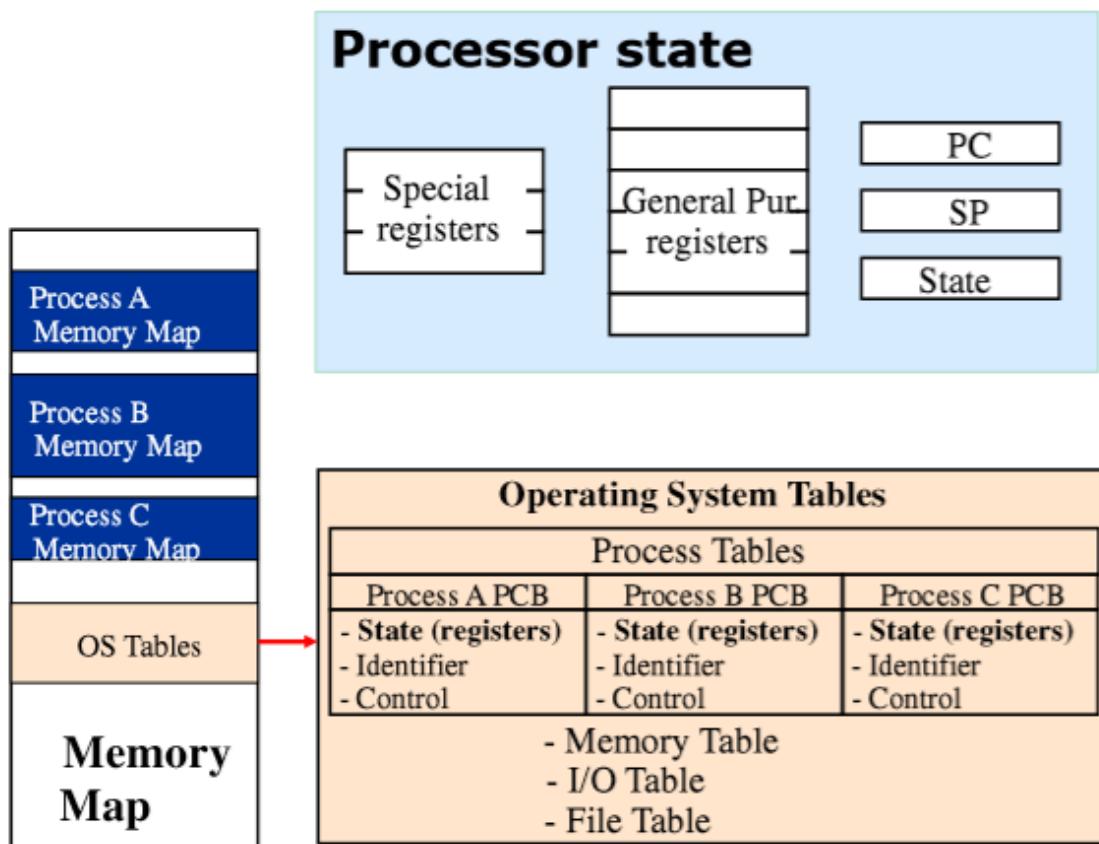
La información guardada es la siguiente

- Estado del proceso – Ejecución...
- Contador del programa – Cuál es la próxima instrucción a ejecutar.
- Registros de la CPU.
- Planificación de la CPU.
- Memoria.
- Contabilidad – Tiempo de CPU usado, límites de tiempo.
- Información de estado de entrada / salida.

Identifier
State
Priority
Program counter
Memory pointers
Context data
I/O status information
Accounting information

Información de procesos

T2.21



Qué define a un proceso

T2.22

- El **estado del procesador**
  - Datos en sus registros
- **Imagen del núcleo**
  - Datos de la memoria
- Información / **estado de cada proceso**
  - **PCB (Process Control Block)**

Todo esto es la **imagen de un proceso**.

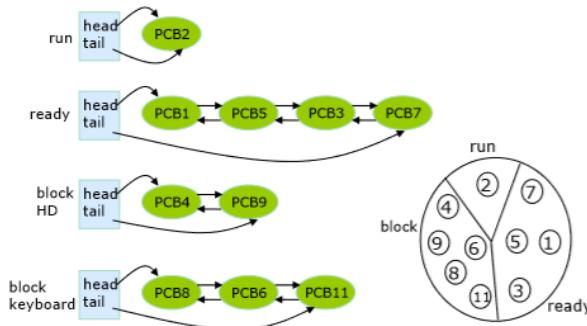
## Organización de procesos

T2.23

Se ejecutan tantos procesos como procesadores haya.

Están en **listo** siguiendo una cola creada por el **planificador**.

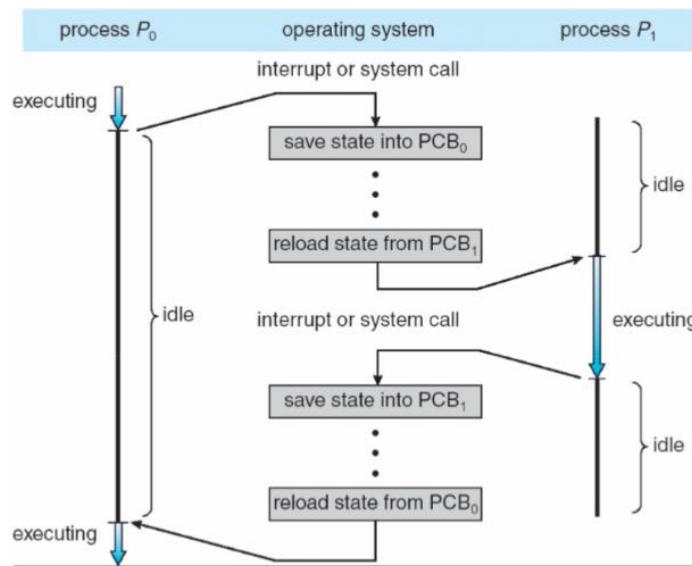
Están en **bloqueado** en varias listas no ordenadas.



## Cambio de procesos en la CPU

T2.24

Cuando un proceso es sacado de la CPU sin haber concluido su ejecución, se guarda su **PCB** y otro proceso ocupa su lugar, se produce un **cambio de contexto**.



## Cambio de contexto

T2.25

Un cambio de contexto se produce cuando la CPU de un ordenador cambia de un proceso o subprocesso a un proceso o subprocesso diferente.

El cambio de contexto permite que una CPU maneje numerosos procesos o subprocessos sin la necesidad de procesadores adicionales.

Un cambio de contexto es el mecanismo para almacenar y restaurar el estado o el contexto de una CPU en el bloque de control de procesos (*PCB*).

Durante el cambio de contexto, el sistema no realiza ninguna acción útil.

El tiempo que tarda esto en ocurrir varía desde 1 hasta 1000 microsegundos.

## Intercalación de procesos

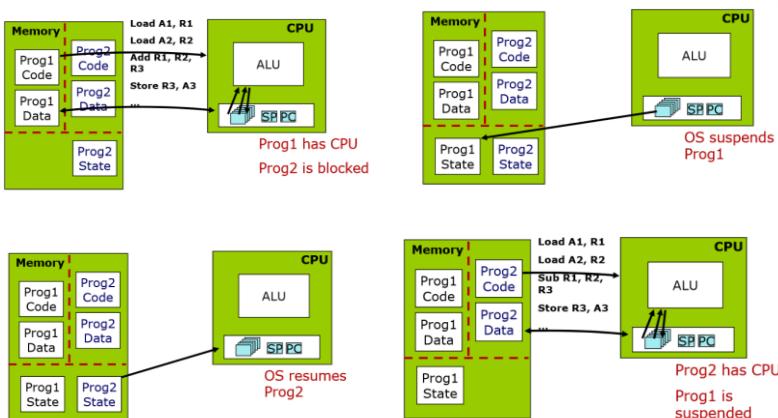
Las instrucciones de un programa operan en memoria y de manera temporal en registros

Cuando suspendemos o bloqueamos un proceso guardamos toda su información en el bloque de control de procesos, o *PCB* por sus siglas en inglés. Gracias a esto podemos reanudarlo más tarde continuando desde donde se suspendió.

De este modo si tenemos el proceso P1 y proceso P2, y queremos ejecutarlos simultáneamente en un procesador deberíamos:

- 1) Ejecutar P1
  - a) Bloquear P1 y guardar su estado en el PCB.
- 2) Ejecutar P2
  - a) Bloquear P2 y guardar su estado en el PCB.
- 3) Ejecutar P1

Y así sucesivamente hasta terminar la ejecución de ambos procesos.



## Creación de procesos

T2.32

Eventos que pueden causar creación de procesos.

Inicialización del sistema.

Inicialización de batch.

Petición de usuario de crear un nuevo proceso.

Creación de un proceso por otro proceso.

Los procesos padre crean procesos hijo, los cuales crean otros procesos, creando un árbol de procesos.

Existen llamadas a sistema especiales para comunicarse y esperar a procesos hijo.

Cada proceso tiene un identificador único PID.

Los procesos padre e hijo pueden o no compartir recursos.

El padre puede ejecutarse concurrentemente con el hijo o esperar a su finalización.

## Creación de procesos – UNIX

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().

Esta llamada crea una copia casi idéntica del proceso padre.

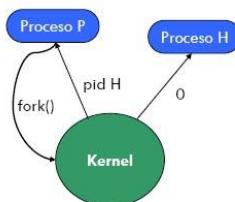
- Ambos procesos, padre e hijo, continúan ejecutándose en paralelo.
- El padre obtiene como resultado de la llamada a fork() el pid del hijo y el hijo 0.
- Algunos recursos no se heredan (p.ej. señales pendientes).

El proceso hijo puede invocar la llamada al sistema exec\*():

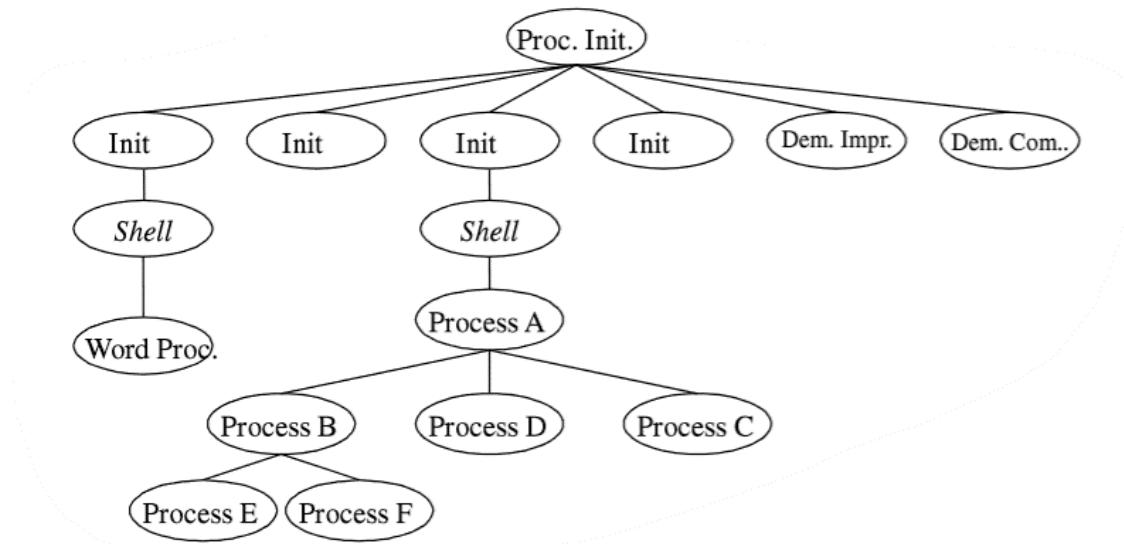
Sustituye su imagen en memoria por la de un programa diferente.

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.



### Árbol de procesos en UNIX



Con **getpid()** obtenemos nuestro **PID** y con **getppid()** obtenemos el PID del proceso padre.

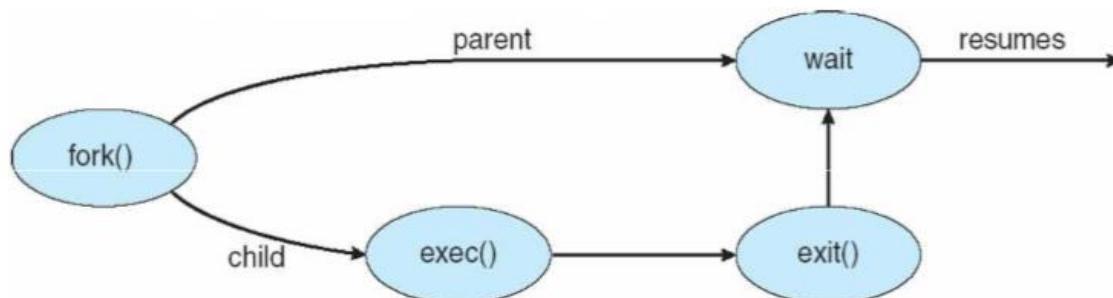
### Terminación de procesos

T2.45

Cuando un proceso termina, hace un **exit()** y esto puede devolver un **valor de estado** al proceso padre a través de un **wait()**.

Los recursos del proceso son desalojados por el sistema operativo.

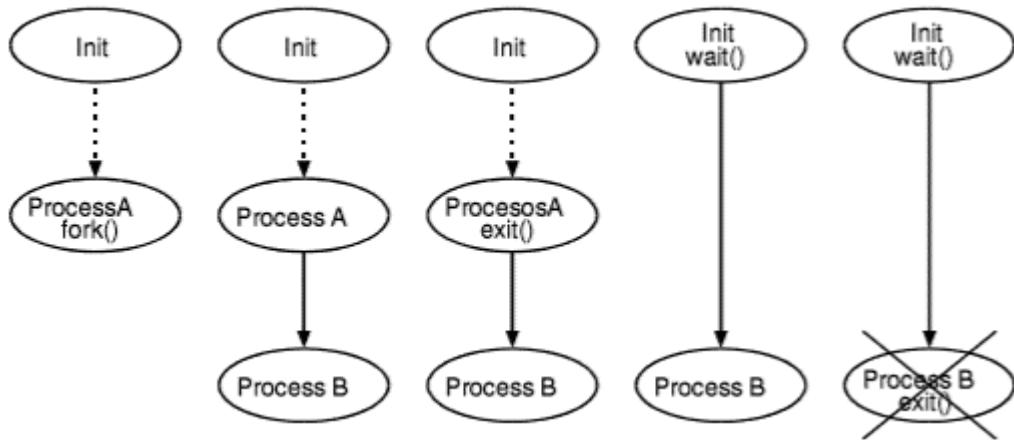
El proceso padre puede terminar la ejecución de un proceso hijo con **abort()**.



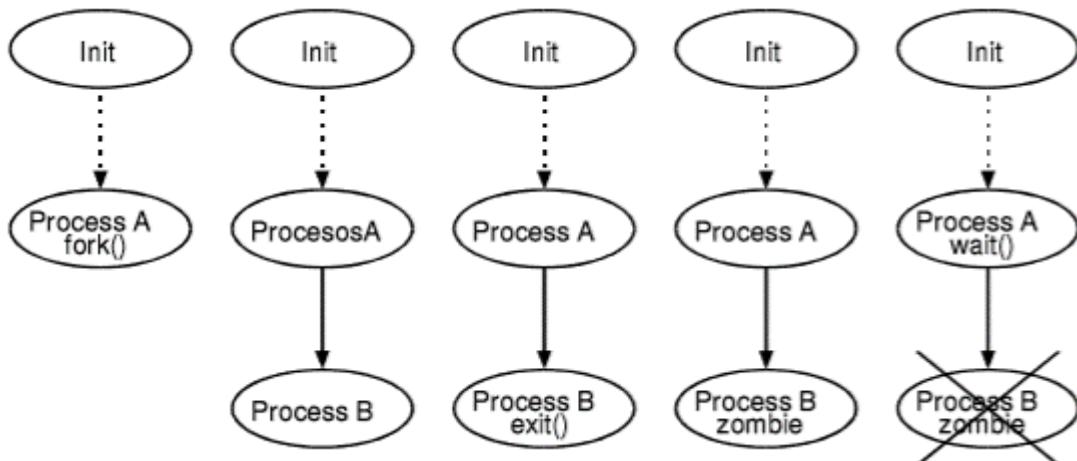
Terminación de procesos en UNIX

init se convierte en su nuevo padre

Si el padre termina pero los hijos no, se le asignan al proceso *init()*.



Un proceso se queda **zombie** si termina pero su padre **no le hace un *wait()***.



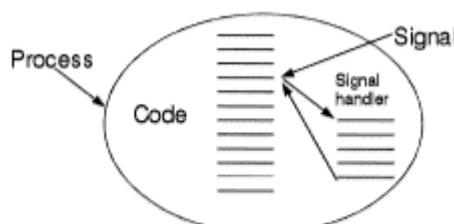
## Manejo de señales

T2.52

Una señal (del inglés *signal*) es una forma limitada de comunicación entre procesos empleada en Unix y otros sistemas operativos compatibles con POSIX. En esencia es una notificación asíncrona enviada a un proceso para informarle de un evento. Cuando se le manda una señal a un proceso, el sistema operativo modifica su ejecución normal. Si se había establecido anteriormente un procedimiento (handler o manejador) para tratar esa señal se ejecuta éste, si no se estableció nada previamente se ejecuta la acción por defecto para esa señal.

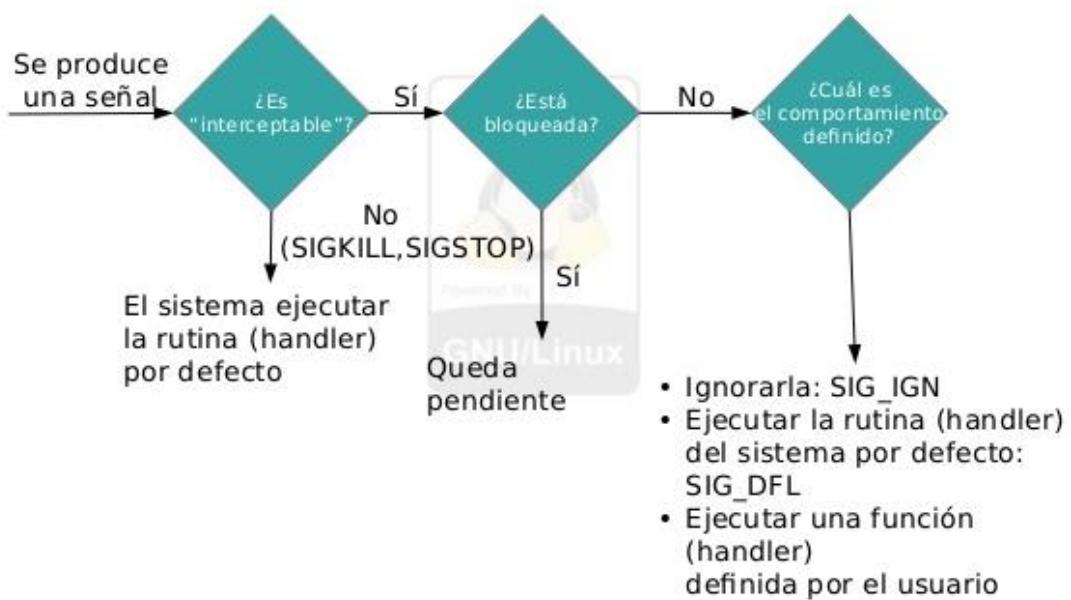
`Signal()`, `kill()` son llamadas a sistema que permiten que se termine un proceso, o se le envíen señales específicas.

Las señales pueden ser enviadas desde el SO a un proceso o desde un proceso a otro.



Source: Sistemas Operativos. Una visión aplicada.

Utilizando `kill -l` en el terminal de Linux podemos obtener una lista de señales.



`sigprocmask()` puede utilizarse para bloquear y desbloquear envío de señales.

Manejo de señales en sistemas POSIX

- `int kill(pid_t pid, int sig)`
  - It sends to the process "pid" the signal "sig"
- `int sigaction(int sig, struct sigaction *act, struct sigaction *oact)`
  - It allows to specify the action to be taken when the signal "sig" is received
- `int pause(void)`
  - It blocks the process until the reception of a signal
- `unsigned int alarm(unsigned int seconds)`
  - It generates the reception of signal SIGALARM after "seconds" seconds
- `sigprocmask(int how, const sigset_t *set, sigset_t *oset)`
  - It is used to explore or modify the signal mask of a process

Procesos demonio

T2.59

Un **daemon** (nomenclatura usada en sistemas POSIX), **servicio** (nomenclatura usada en Windows) o **programa residente** (nomenclatura usada en MS-DOS) es un tipo especial de proceso informático **no interactivo**, es decir, que **se ejecuta en segundo plano** en vez de ser controlado directamente por el usuario. Este tipo de programas continua en el sistema, puede ser ejecutado en forma persistente o reiniciado si se intenta matar el proceso dependiendo de la configuración del daemon y de las políticas del sistema.

Características:

- Se ejecutan con el inicio del sistema.
- No se terminan.
- Normalmente esperan un evento.
- No realizan ninguna acción, sino que crean otros procesos o hilos.
- Pueden encontrarse en una máquina diferente a la del cliente.

## Cooperación de procesos

T2.60

Los procesos pueden ser **independientes o cooperativos**.

Los procesos **independientes** no pueden afectar o ser afectados por la ejecución de otro proceso.

Los procesos **cooperativos** pueden afectar o ser afectados por la ejecución de otro proceso.

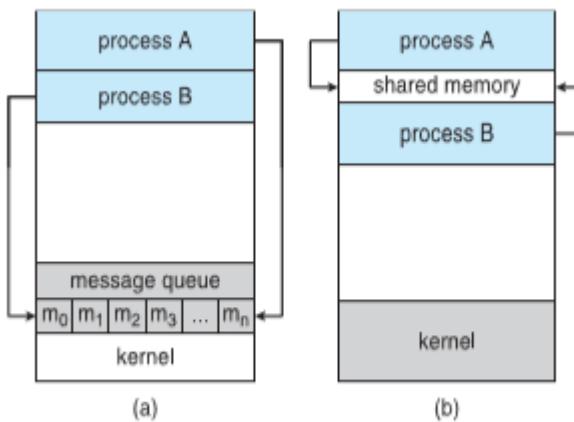
La cooperación de procesos tiene diversas ventajas.

- Compartición de información.
- Mejora en la velocidad de computación debido a subtareas paralelas.
- Modularidad al dividir funciones en procesos independientes.
- Conveniencia

Para poder funcionar correctamente los procesos cooperativos necesitan **comunicación entre procesos (IPC, interprocess communication)**.

Existen dos modos de IPC:

- **Memoria compartida.**
  - a. Gran velocidad y conveniencia.
  - b. Las llamadas de sistema sólo se necesitan para establecer las regiones de memoria compartida, el resto de entrada / salida no necesita el kernel.
- **Paso de mensajes.**
  - a. Útil para pequeñas cantidades de datos.
  - b. Más fácil de implementar que la memoria compartida.
  - c. Requiere llamadas al sistema, intervención del kernel.



Desventajas de los procesos

T2.63

Existen aplicaciones que son paralelas por naturaleza, y necesitan compartir el mismo espacio de direcciones.

El **paralelismo** implica que un sistema pueda ejecutar más de una tarea simultáneamente.

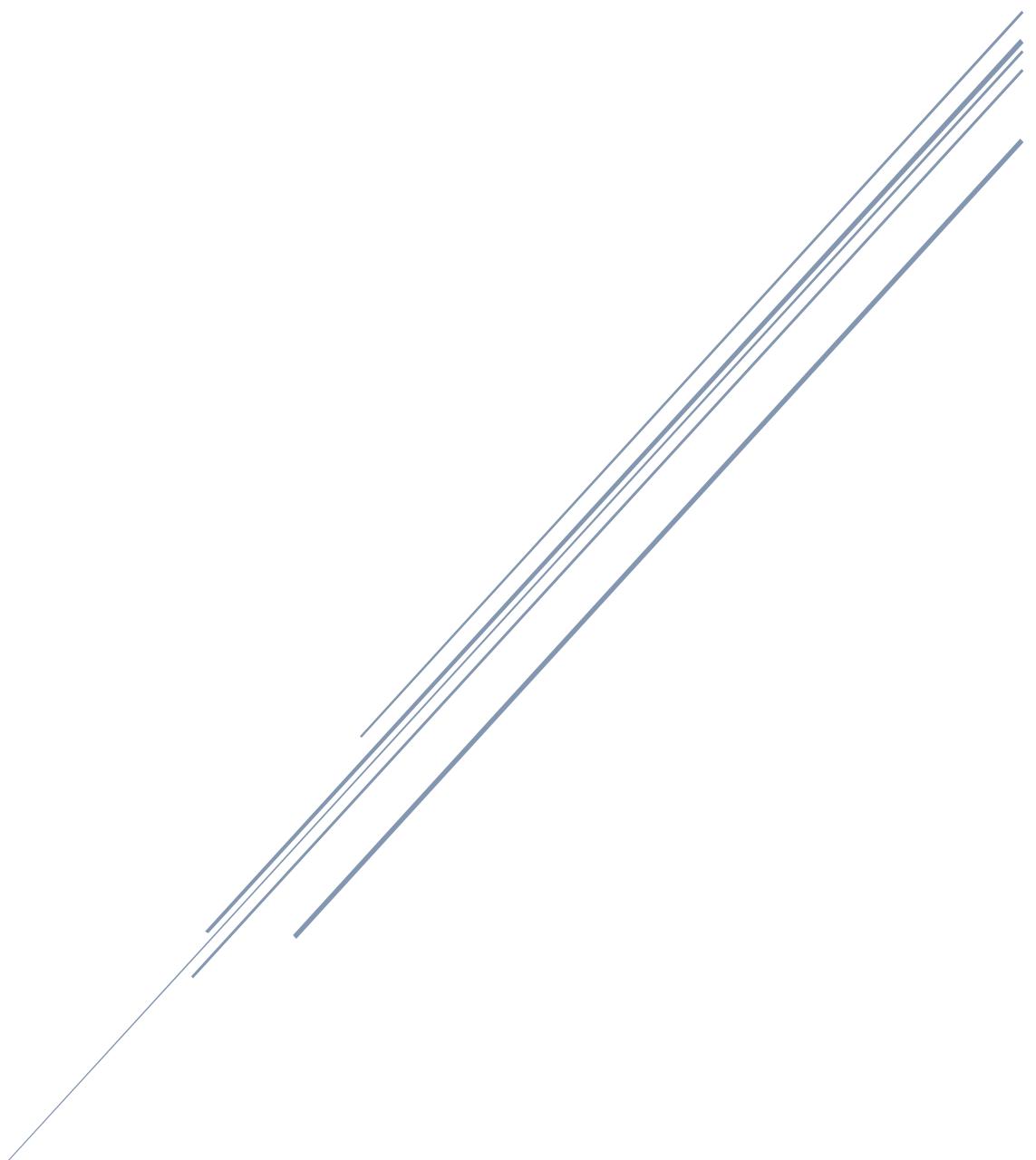
No es fácil ejecutar paralelismo utilizando procesos pues la creación de procesos es costosa en recursos y tiempo, así como la comunicación entre procesos.

Por ello, aparece un nuevo modelo de ejecución, los **threads**.



# SISTEMAS OPERATIVOS

## Tema 2.2



Escuela Técnica Superior de Ingeniería Informática

## INDICE

Sistemas Operativos.....	- 2 -
Ráfagas de CPU... .....	- 2 -
Problemas de planificación.....	- 2 -
Planificadores.....	- 3 -
Criterios de planificación... .....	- 3 -
Criterios del algoritmo optimizador del planificador .....	- 4 -
Diversidad de entornos .....	- 4 -
Planificador de la CPU .....	- 5 -
Algoritmos de planificación .....	- 5 -
First-Come, First-Served <i>FSFC</i> .....	- 6 -
Shortest Job First <i>SJF</i> .....	- 7 -
Round Robin .....	- 8 -
Shortest Remaining Time First <i>SRTF</i> .....	- 9 -
Planificación con prioridad .....	- 10 -
Cola multinivel .....	- 10 -
Cola multinivel con retroalimentación .....	- 11 -
Planificación con procesadores múltiples .....	- 12 -
Sistema multiprocesador y multihilo .....	- 12 -
Balanceo de carga en planificación con multi procesador .....	- 13 -
Windows .....	- 14 -
Propiedades del planificador Windows .....	- 16 -
Linux .....	- 16 -
Planificación de tiempo real en Linux.....	- 16 -
Planificador Linux .....	- 17 -
Planificación de tiempo compartido en Linux .....	- 18 -
Planificador Linux: CFS .....	- 19 -
Tiempo de ejecución virtual .....	- 20 -
Ejemplo de examen.....	- 21 -

## Sistemas Operativos

### Tema 2.2

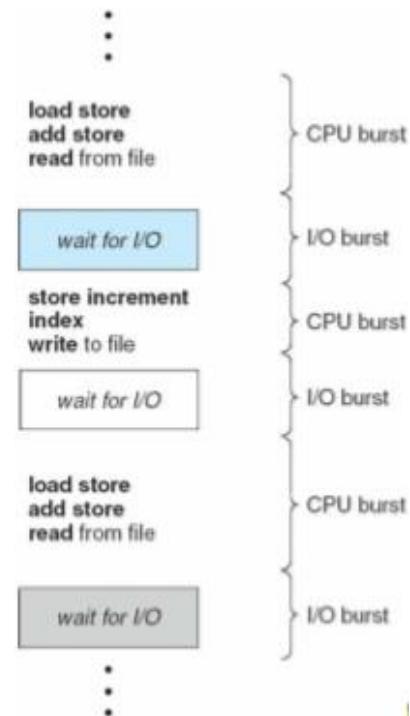
Ráfagas de CPU

T2.2-3

Los programas alternan entre ráfagas de CPU y entrada / salida.

**E/S:** Consume más tiempo en entrada / salida que en computación, muchas ráfagas pequeñas de CPU.

**CPU:** Consume más tiempo en computación, pocas pero muy largas ráfagas de CPU.



Problemas de planificación

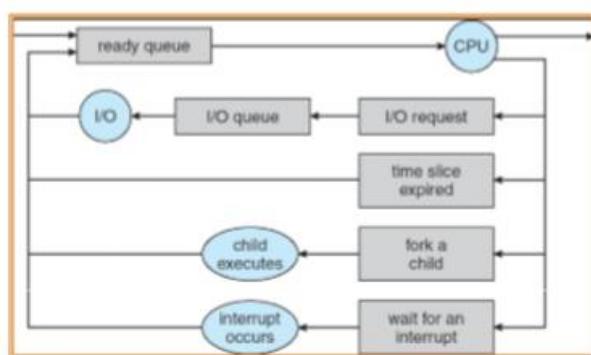
T2.2-4

Cada decisión de planificación consiste en decidir qué proceso entrará en la CPU en la próxima ráfaga de CPU.

Con partición temporal, los procesos o hilos pueden ser forzados a abandonar la CPU aunque no haya terminado la actual ráfaga de CPU.

La planificación es algo central en el diseño de Sistemas Operativos.

Tanto la CPU como la mayoría de recursos son planificados antes de su uso.



**Ya puedes sacarte tu B1/B2/C1 de inglés desde casa**



#LinguaskillEnCasa

Apuntes de Sistemas Operativos  
2º Ingeniería Informática ETSII  
Gerardo Marín

## Planificadores

T2.2-5

**Planificador a corto plazo:** selecciona el siguiente proceso a ser ejecutado y lo aloja en la CPU.

Se invoca de manera frecuente (100ms) y debe ser rápido o malgastaría demasiado tiempo planificando y no ejecutando.

**Planificador a plazo medio:** selecciona qué procesos deberían ser “swappeados” de memoria.

**Planificador a largo plazo:** selecciona qué procesos deberían entrar en la cola de **listos**.

Este planificador debería seleccionar un buen conjunto de procesos para mejorar el rendimiento del sistema.

Se invoca de manera poco frecuente (segundos o minutos). Controla el **grado de multiprogramación**.<sup>1</sup>

## Criterios de planificación

T2.2-7

- Utilización de la CPU
  - Intentamos mantener la CPU tan ocupada como sea posible.
- Rendimiento (Throughput)
  - Número de procesos que completan su ejecución por unidad de tiempo.
- Tiempo de vuelta (Turnaround time **Tr**)
  - Tiempo en ejecutar un proceso concreto. Esperar a entrar en memoria, esperar en la cola de listos, ejecución en la CPU y entrada / salida.
- Tiempo de espera (Waiting time **Te**)
  - Tiempo que un proceso ha esperado en la cola de listos.
- Tiempo de respuesta (Response time **Ta**)
  - Tiempo desde que se envía una petición hasta que la primera respuesta se produce.

---

<sup>1</sup> Como vimos en el tema 2.1, el grado de multiprogramación corresponde al número de procesos en memoria.

Criterios del algoritmo optimizador del planificador

T2.2-8

Utilización **máxima** de la CPU.

**Máximo rendimiento.**

Tiempo de vuelta mínimo.

Tiempo de espera mínimo.

Tiempo de respuesta mínimo.

*En la mayoría de casos optimizamos la medición media, sin embargo en algunos casos conviene optimizar los valores mínimo o máximo*

Diversidad de entornos

T2.2-9

Cada entorno tiene diferentes metas, por lo que cada uno tiene distintas **políticas de planificación.**<sup>2</sup>

- **Sistemas batch:**
  - Capacidad de procesamiento, rendimiento.
  - Tiempo de vuelta *Turnaround time*.
  - Uso de la CPU.
- **Sistemas de tiempo compartido**
  - Tiempos de respuesta.
  - Complejidad.
- **Sistemas de tiempo real**
  - Cumplimiento de plazos.
  - Predictibilidad o consistencia.

---

<sup>2</sup> Estrategias y decisiones utilizadas en el diseño del planificador con objeto de alcanzar las metas establecidas.

## Planificador de la CPU

T2.2-11

Selecciona entre los procesos de memoria que están listos para su ejecución y lo aloja uno de ellos en la CPU.

El planificador de la CPU puede activarse cuando un proceso

- Cambia de **ejecución** a **espera**.
- Cambia de **ejecución** a **listo**.
- Cambia de **espera** a **listo**.
- Termina.

La planificación cuando cambia **de ejecución a espera o termina** es **no expropiativa**.

Una vez el proceso entra en la CPU **no sale** hasta que o bien termina o pasa a estado de espera.

El resto son **expropiativas**.

Un proceso en ejecución **puede ser sacado de la CPU** por motivos como que otro proceso tenga mayor prioridad.

## Algoritmos de planificación

T2.2-12

Sched.	Preem. <b>Non Preempt.</b>	Preempt.
<b>Arrival Time</b>	FCFS	Round Robin
<b>Execution Time</b>	SJF	SRTF
<b>Priority</b>	Priority	Preemption Priority

**No expropiativa.**

El primero de la cola es el primero en entrar a la CPU sin importar su prioridad.

Los procesos abandonan la CPU cuando terminan o pasan a estado de espera.

FCFS es simple.

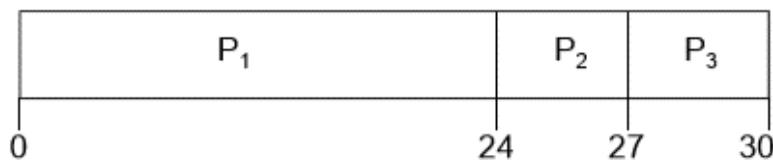
Empeora la CPU y la utilización del dispositivo.

Tiene tiempos de espera y de respuesta altos.

No es bueno en sistemas de tiempo compartido donde cada usuario necesita usar la CPU en intervalos regulares pues puede ocurrir el **efecto convoy**, que un proceso pequeño tenga que esperar durante largos períodos de tiempo a que un proceso más grande termine su ejecución.

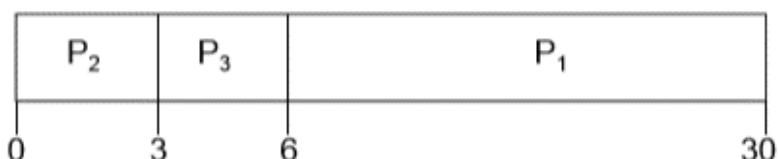
**Planificación FCFS**

Llegando los procesos  $P_1$ ,  $P_2$  y  $P_3$  en ese orden.



Vemos un claro ejemplo de efecto convoy.

Suponiendo que el orden sea  $P_2$ ,  $P_3$  y  $P_1$ .



**Ya puedes sacarte tu B1/B2/C1 de inglés desde casa**



Apuntes de Sistemas Operativos  
2º Ingeniería Informática ETSII  
Gerardo Marín

Shortest Job First SJF

T2.2-18

**No expropiativa.**

Asociado a cada proceso está la longitud de su próxima ráfaga de CPU, que se usan para planificar el proceso con el tiempo de ráfaga más corto.

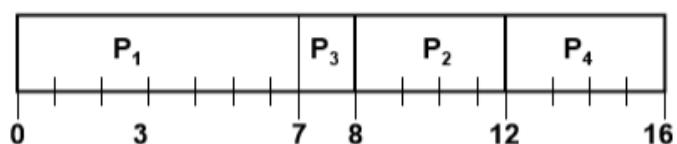
Es bueno para sistemas de tiempo compartido puesto que tiene un tiempo de espera bajo.

Sin embargo tiene contrapartidas como la inanición de procesos grandes, debido a que si hay mucho procesos pequeños los grande podrían no llegar a ejecutarse nunca.<sup>3</sup>

Ejemplo:

Process	Arrival Time	Burst Time
$P_1$	0.0	7
$P_2$	2.0	4
$P_3$	4.0	1
$P_4$	5.0	4

■ SJF scheduling chart



- Average waiting time ( $T_e$ ) :  $(0 + 6 + 3 + 7) / 4 = 4$
- Average completion time ( $T_c$ ) :  $(7+10+4+11)/4 = 8$

Tenemos dificultad al saber cuánto tardará el próximo proceso en ejecutarse, aun más si es un proceso que nunca se ha ejecutado y no podemos utilizar predicción.

Para ello existen estimadores (*Transparencia Nº 22 Tema 2.2*).

<sup>3</sup> Como curiosidad, en clase se habló de cómo un ordenador antiguo tras treinta años de funcionamiento se apagó, para tras revisar su memoria comprobar que había un proceso de gran tamaño aún estaba esperando su entrada a la CPU.

Cada proceso obtiene una pequeña unidad de tiempo de CPU, denominada ***quantum***, normalmente entre 10 y 100 ms.

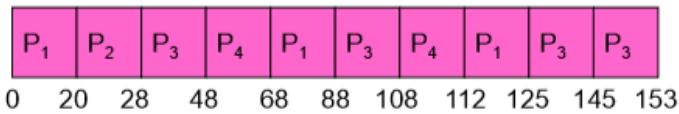
Después de que pase este tiempo, el proceso es **expropiado**, se le saca de la CPU y se pone al final de la cola de listos, para meter en la CPU al inmediatamente próximo. De este modo todos los procesos tienen el mismo tiempo de CPU.

Si un proceso **termina** antes de que acabe el *quantum*, el próximo en la cola entra a ejecución.

El **quantum** no debe ser demasiado grande, o actuaría como FCFS, ni demasiado pequeño, o el cambio de contexto ocuparía todo el espacio del *quantum* y no sería eficiente el procesamiento.

■ Example:	Process	Burst Time
	$P_1$	53
	$P_2$	8
	$P_3$	68
	$P_4$	24

- The Gantt chart is:

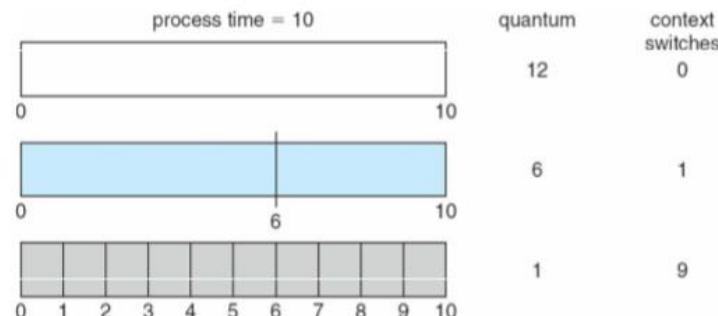


- Waiting time for  $P_1 = (68-20)+(112-88)=72$
- $P_2 = (20-0)=20$
- $P_3 = (28-0)+(88-48)+(125-108)=85$
- $P_4 = (48-0)+(108-68)=88$

- Average waiting time =  $(72+20+85+88)/4=66\frac{1}{4}$
- Average completion time =  $(125+28+153+112)/4 = 104\frac{1}{2}$

■ Thus, Round-Robin Pros and Cons:

- **Better for short jobs, Fair (+)**
- **Context-switching time adds up for long jobs (-)**



A smaller time quantum increases context switches

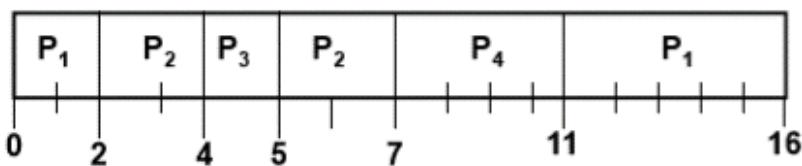
Es la versión **expropiativa** de *Shortest Job First*.

Si un nuevo proceso llega con un tiempo de CPU menor que **el tiempo que le queda de uso de CPU al que ya está usándola**, se le echa del procesador y se mete este nuevo proceso.

Al igual que en el *S/J*, puede haber inanición y un proceso grande puede llegar a no ejecutarse nunca.

Sin embargo el tiempo de respuesta es bueno, aunque es difícil hacer predicciones y es injusto, dado que tienen prioridad los más pequeños.

<u>Process</u>	<u>Arrival Time</u>	<u>Burst Time</u>
$P_1$	0.0	7
$P_2$	2.0	4
$P_3$	4.0	1
$P_4$	5.0	4



El proceso  $P_1$  llega y entra en la CPU.

El proceso  $P_2$  llega, y en **ese momento, a  $P_1$  le faltan 5 unidades de tiempo para terminar**, así que **sale  $P_1$  y entra  $P_2$** , que solamente necesita 4.

En el instante 4, llega  $P_3$ , que **sólo necesita 1 unidad de tiempo**, mientras que a  $P_2$  le quedan 2 **unidades de tiempo**, **sale  $P_2$ , entra  $P_3$** .

$P_3$  termina su ejecución y llega  $P_4$  a la cola, pero  $P_4$  tiene 4 **unidades**, mientras que  $P_2$  le faltan 2 **unidades para terminar**, **entra  $P_2$  y termina**.

Quedan por ejecutar  $P_1$  y  $P_4$ , pero a  $P_1$  aún **le quedan 5 unidades de tiempo**, y  $P_4$  sólo necesita 4, **entra  $P_4$  y tras terminar,  $P_1$** .

## Planificación con prioridad

T2.2-31

Un número entero de prioridad se asocia a cada proceso y a la CPU entra el que tenga mayor prioridad (siendo el de número más bajo el de mayor prioridad, 1 tiene más que 5...)

**Expropiativa**, compara la prioridad de cada proceso que ha llegado a la cola de listos con el proceso actualmente en ejecución.

**No expropiativa**, se pone el primero en la cola de listos.

Al igual que en los que el más grande no llegaban a ejecutarse en otras políticas de planificación, en esta podrían no llegar a ejecutarse los de menor prioridad.

Una posible solución sería incrementar la prioridad del proceso conforme el tiempo pasa.

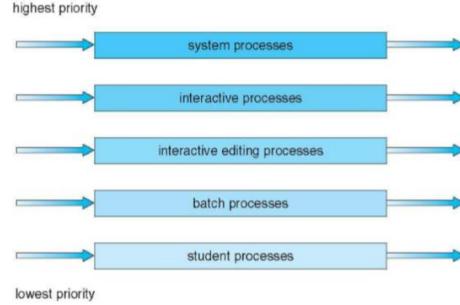
Process	Burst Time	Priority
P <sub>1</sub>	10	3
P <sub>2</sub>	1	1
P <sub>3</sub>	2	4
P <sub>4</sub>	1	5
P <sub>5</sub>	5	2



## Cola multinivel

T2.2-34

Esta cola se partitiona en colas separadas, por ejemplo **cola interactiva, foreground o background**. Los procesos se mantienen permanentemente en la cola asignada y cada una de las colas tiene su propio algoritmo de planificación.



Ya puedes sacarte tu B1/B2/C1 de inglés desde casa



#LinguaskillEnCasa

Apuntes de Sistemas Operativos  
2º Ingeniería Informática ETSII  
Gerardo Marín

Cola multinivel con retroalimentación

T2.2-36

El algoritmo de colas multinivel presenta baja carga de planificación pero es poco flexible.

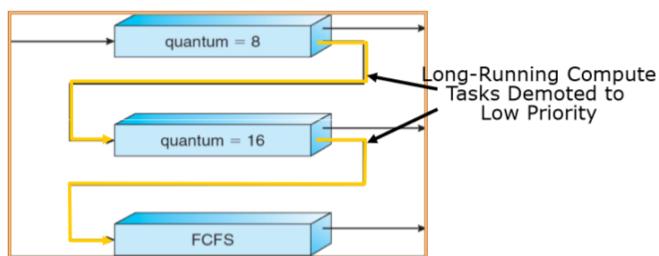
Mediante la planificación con colas multinivel realimentadas, un proceso se puede **mover de una cola a otra** dependiendo de su comportamiento en tiempo de ejecución. Además puede ser implementado el *envejecimiento*<sup>4</sup> de los procesos.

La planificación de Colas Multinivel con retroalimentación está definida por los siguientes parámetros:

- Número de colas
- Algoritmos de planificación por cola
- Método usado para determinar cuándo promover un proceso.
- Método usado para determinar cuándo degradar un proceso.
- Método usado para determinar que cola entrara en el proceso cuando requiera de servicio.

La prioridad de cada proceso se establece de la siguiente manera:

1. El proceso comienza en la cola de prioridad más alta,
2. Si el tiempo de espera expira, baja un nivel,
3. Si el tiempo de espera no vence, sube un nivel.



<sup>4</sup> Con objeto de evitar inanición en casos como *Shortest Job First* o similares, conforme más tiempo pase en espera un proceso más prioridad se le asigna para asegurarnos de que en algún momento se ejecutarán todos los procesos.

## Planificación con procesadores múltiples

T2.2-39

Cuando existen varios procesadores la planificación de éstos es más complicada. Existen distintas arquitecturas multiproceso.

- Procesadores multinúcleo.
- Procesadores multihilo.
- Sistemas NUMA<sup>5</sup>.
- Multiprocesamiento heterogéneo.
- Procesadores homogéneos.
- Multiprocesamiento asimétrico – sólo un procesador accede a la estructura de datos del sistema, haciendo no necesaria la compartición de datos.
- Multiprocesamiento simétrico – cada procesador se autoplanifica.

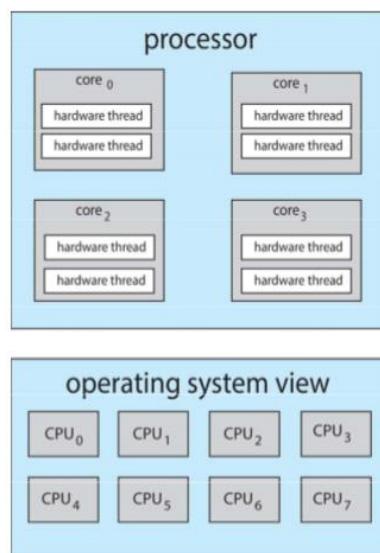
Actualmente el más común es el **multiprocesamiento simétrico o SMP**.

## Sistema multiprocesador y multihilo

T2.2-41

*Multicore and multi threaded*

El **Chip-multithreading** o *CMT* asigna a cada núcleo diversos hilos hardware. También se conoce como **hyperthreading**.



<sup>5</sup> En computación, **NUMA** (del inglés *Non-Uniform Memory Access*, en español "acceso a memoria no uniforme") es un diseño de memoria utilizado en multiprocesamiento donde la memoria se accede en posiciones relativas de otro proceso o memoria compartida entre procesos. Bajo NUMA, un procesador puede acceder a su propia memoria local de forma más rápida que a la memoria no local (memoria local de otro procesador o memoria compartida entre procesadores).

## Balanceo de carga en planificación con multi procesador

T2.2-43

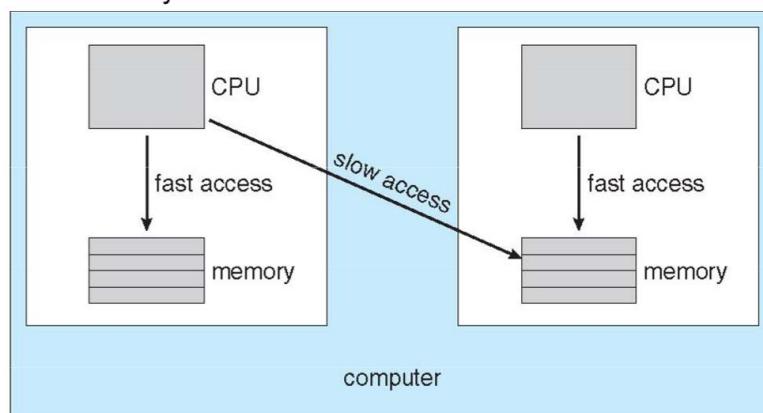
Si existe **SMP** es necesario mantener todos los procesadores ocupados para mayor eficiencia.

El balanceo de carga consiste en **mantener el trabajo de los procesadores lo mejor distribuido posible entre ellos**, de este modo se chequea la carga de cada procesador de manera periódica y si alguno de ellos está sobrecargado, se le pasa ese trabajo a otros procesadores.

Aquellos procesadores que en su cola de listos no tengan ningún proceso o no estén ejecutando ningún proceso cogerán procesos de otros procesadores ocupados.

La afinidad del procesador consiste en que los procesos tienen **mayor afinidad por el procesador en el que están actualmente ejecutándose**, esta afinidad puede ser blanda o dura.

La arquitectura de la memoria puede afectar generando problemas en la afinidad del procesador debido a que un **procesador tiene un acceso más rápido a partes concretas de la memoria (NUMA)**. El planificador del procesador del sistema y los algoritmos de colocación de memoria deberían trabajar en conjunto de modo **que un proceso que tiene afinidad a un procesador en particular puede utilizar la memoria correspondiente a ese procesador**.



Windows 7/10 es un sistema operativo expropiativo que utiliza los **hilos** como unidad a ser planificada.

Utiliza **prioridades** para realizar las expropiaciones, utilizando números entre el 0 y el 31.

Estáticos para tiempo real [16...31], sólo para el administrador, SO.

Variables para el usuario [1...15], aplicaciones.

La prioridad 0 se le asigna al **hilo cero**.

A más bajo el número, mayor prioridad.

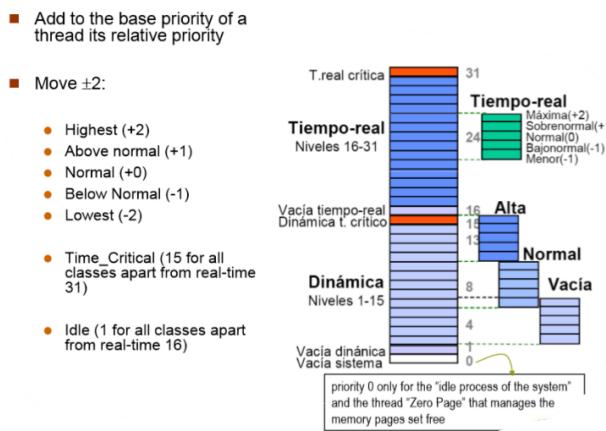
Las normas de planificación en Windows son las siguientes:

- Los procesos en tiempo real tienen la mayor prioridad.
- Los hilos interactivos aumentan de prioridad.
- El tiempo compartido se realiza vía **Round Robin**. El tiempo de un quantum puede variar entre clientes y servidores.

Existen dos estados de asignación de prioridad.

- Prioridad relativa (hilos).
  - Tiempo real.
  - Alto.
  - Por encima de lo normal.
  - Normal.
  - Por debajo de lo normal.
  - Parado.

Es posible modificar la prioridad hasta 2 niveles por encima y 2 niveles por debajo.



Ya puedes sacarte tu B1/B2/C1 de inglés desde casa



Apuntes de Sistemas Operativos  
2º Ingeniería Informática ETSII  
Gerardo Marín

El planificador tiene 32 colas para cada hilo (una cola para cada prioridad).

Se escoge el primer hilo preparado de la cola no vacía con mayor prioridad.

En cada cola se utiliza *Round Robin*.

En el hilo expropiativo, si se expropia por un quantum expirado el proceso se coloca al final de su cola de prioridad (*Round Robin*). Si se expropia por un proceso de mayor prioridad se coloca en la cabeza de su cola de prioridad.

#### Actualización automática de la prioridad.

- Los hilos de usuario cambian en la cola multinivel con retroalimentación.
  - Para prioridades entre 1 – 15 incluidos.
  - Pueden ocurrir aumentos y descensos de la prioridad.
- Aumenta la prioridad de entrada / salida en procesos intensivos y evita inanición.
- No actualiza los hilos de tiempo real (prioridad > 15).

#### Aumento en la prioridad de hilos.

- Cuando un proceso se bloquea suele ser debido a una petición de entrada / salida
  - A los dispositivos lentos se les da grandes aumentos de prioridad mientras que a los rápidos pequeños aumentos.
- Se aumenta en relación a la prioridad base.
  - Nunca se puede superar 15, que es el máximo para los procesos de usuario.
- El tiempo de respuesta es bueno para procesos interactivos, los dispositivos de entrada / salida se mantienen ocupados.

#### Decremento en la prioridad de hilos.

Cada vez que se utiliza un quantum, se decrementa la prioridad en 1 hasta llegar a la prioridad base del hilo.

#### Evitar inanición.

Existe un “*administrador de balance*” el cual es un hilo con prioridad de sistema 16, que se ejecuta cada segundo, **busca hilos de usuario que se hayan estado ejecutando más de 3 segundos y aumenta su prioridad así como su quantum** (aumentar prioridad = disminuirla, incrementamos su número de prioridad hasta 15) Este cambio es temporal.

## Propiedades del planificador Windows

Prioriza automáticamente el incremento de un bit arbitrario, **dando más prioridad a hilos que utilicen dispositivos de entrada / salida.**

La **inanición no se evita de manera implícita**, no existe envejecimiento por lo que se debe mantener constancia de ello.

El soporte a tiempo real es *blando* por lo que no se garantizan tiempos de respuesta y la planificación es predecible debido a prioridades estáticas.

Linux

T2.2-58

Los objetivos en Unix son bajos tiempos de respuesta, alta productividad para procesos en el *background*, evitar inanición y un balanceo correcto de altas y bajas prioridades.

### El **algoritmo** de planificación

Planifica procesos e hilos, es **expropiativo** basándose en prioridades (colas multinivel), utiliza Round Robin y incrementa la prioridad de procesos interactivos.

## Planificación de tiempo real en Linux

El tiempo real se planifica antes que el resto de procesos, se planifica utilizando **expropiación**.

Siempre se seleccionan los procesos de tiempo real con la mayor prioridad, si hay varios se selecciona el primero de la cola.

Las prioridades son estáticas 1...99.

Existen dos clases de planificación en tiempo real para procesos con misma prioridad.

- FIFO – En orden de llegada.
  - Sin quantum, se ejecuta hasta terminar y sólo se expropia si llega un proceso de tiempo real con mayor prioridad.
- Round Robin
  - El tiempo se comparte entre procesos de tiempo real **con la misma prioridad**.
  - Básicamente es Round Robin pero solo entre este tipo de procesos.
  - En caso de expropiación, se mueven a la cabeza de la cola.

## Planificador Linux

*Kernel 2.5*

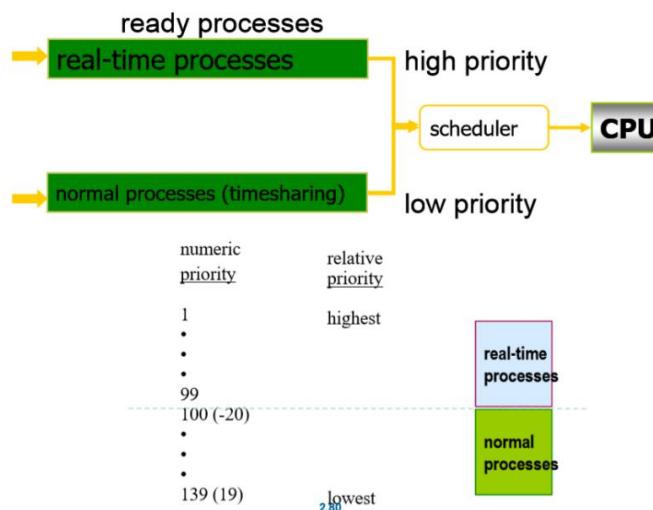
Existen **colas multinivel**

Procesos de tiempo real (sin retroalimentación).

Procesos normales (con retroalimentación).

Dos clases de procesos, por tanto dos estrategias.

- Procesos de tiempo real
  - Siempre se ejecutan antes que los procesos normales.
  - Tienen prioridades estáticas.
  - No es justo, pues tienen prioridad absoluta.
- Procesos normales
  - Prioridades dinámicas.
  - Intentan ser justos.
  - Procesos con baja prioridad pueden llegar a ejecutarse.
  - Se incrementa la prioridad de procesos interactivos.



## Planificación de tiempo compartido en Linux

Siempre se selecciona el proceso **con mayor prioridad, hay expropiación**. Las prioridades son dinámicas aunque también hay estáticas, los procesos heredan la **prioridad de sus procesos padre**, los usuarios pueden utilizar llamadas como *nice()* o *setpriority()* para disminuir la prioridad base de sus procesos.

Los procesos con misma prioridad utilizan Round Robin.

### Determinación de prioridad dinámica

Cada proceso tiene una prioridad base o estática heredada del proceso padre, por defecto esta prioridad es 120, la prioridad dinámica se determina empezando desde la base y aumentando la prioridad de procesos interactivos. Se utilizan “*premios*” y “*castigos*” en función del tiempo de uso de CPU o de entrada / salida.

El premio se determina basándonos en la media de tiempo parado

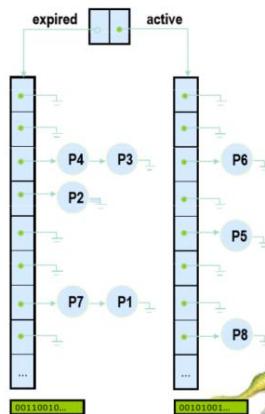
- ▶ When process “awakes” → TMD+= sleep time
- ▶ When process leaves CPU → TMD-= execution time
- ▶  $0 \leq TMD \leq 1000$  ms
- ▶ bonus = floor(TMD/ 100)

### Round Robin modificado

Con dos colas de lista tenemos **lista activa** de procesos que no han utilizado aun su quantum y **lista expirada** de procesos que ya han utilizando su quantum.

El siguiente proceso a ejecutar se selecciona **sólo** de la lista **activa** y si éste está bloqueado, se devuelve a la lista activa cuando se despierte. Si se utiliza su quantum, se coloca en la **lista expirada**.

- Modified Round Robin
  - avoid starvation:
    - ▶ a lower priority process is run despite higher priority ready processes exists. The latter have used their quantum.
  - “running periods” appear
    - ▶ all processes are run during an period
    - ▶ a period finishes if all processes have been run. No ready active exist.
  - period switch
    - ▶ simply interchange the expired queue with the active one



Ya puedes sacarte tu B1/B2/C1 de inglés desde casa



#LinguaskillEnCasa

Apuntes de Sistemas Operativos  
2º Ingeniería Informática ETSII  
Gerardo Marín

### Round Robin modificado

La implementación real es algo más sofisticada, se clasifican los procesos en **Batch** e **Interactivos**, los cuales lo son si su prioridad dinámica **es menor o igual a (3 \* su prioridad estática / 4 + 28)**, y depende de un bonus del proceso (TDM).

En un proceso Batch, si su quantum se ha utilizado, lo movemos a **expirado**. Sin embargo los procesos interactivos suelen mantenerse en **activo** aunque su quantum haya sido utilizado, y se mueven a **expirado** si el expirado más antiguo ha estado esperando por un largo periodo de tiempo o existe un proceso expirado con una prioridad estática mayor que el interactivo.

Planificador Linux: CFS  
Kernel 2.6.23

El planificador es justo.

La idea es compartir la CPU proporcionalmente a cada proceso en ejecución según su valor NICE. No se le asignan quantums a los procesos, sino una porción de tiempo de CPU y el valor NICE actúa como el peso o prioridad para esta porción.

El valor NICE se comprende entre -20 y 19, a menor valor, mayor peso.

Nice	-20	-19	-18	-17	-16	-15	-14	-13	-12	-11
Weight	88761	71755	56483	46273	36291	29154	23254	18705	14949	11916
Nice	-10	-9	-8	-7	-6	-5	-4	-3	-2	-1
Weight	9548	7620	6100	4904	3906	3121	2501	1991	1586	1277
Nice	0	1	2	3	4	5	6	7	8	9
Weight	1024	820	655	526	423	335	272	215	172	137
Nice	10	11	12	13	14	15	16	17	18	19
Weight	110	87	70	56	45	36	29	23	18	15

Dos clases de procesos, por tanto dos estrategias.

- Procesos de tiempo real
  - Siempre se ejecutan antes que los procesos normales.
  - Tienen prioridades estáticas.
  - No es justo, pues tienen prioridad absoluta.
- Procesos normales
  - Prioridades dinámicas.
  - Intentan ser justos.
  - Procesos con baja prioridad pueden llegar a ejecutarse.
  - Se incrementa la prioridad de procesos interactivos.

## Tiempo de ejecución virtual

No existe quantum, sino porcentaje de la CPU usado. El tiempo consumido por cada proceso debes ser contado y el *VirtualRuntime* es la acumulación del tiempo de ejecución por un proceso inversamente “pesado” por su peso.

$$\text{virtual\_runtime}(P_i, t) = \frac{\text{weight}_0}{\text{weight}_{P_i}} \times \text{tiempo\_ejec}(P_i, t)$$

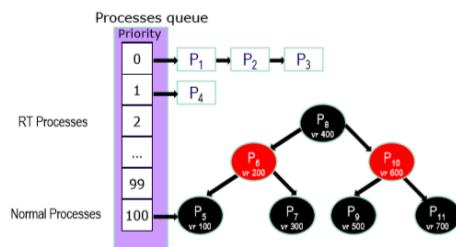
- ▶ Weight<sub>0</sub>= weight of nice 0
- ▶ Processes with nice = 0 counted **exactly** their execution time
- ▶ Processes with nice < 0 counted **less** time than their real execution time
- ▶ Processes with nice > 0 counted **more** time than their real execution time

## Cola de procesos

Sólo una cola de **listos para todos los procesos de prioridad normal**.

Se ordenan incrementando el valor del *Virtual\_Runtime* y se implementa utilizando el árbol rojo-negro.

- ▶ Insertions and deletions in O(log<sub>2</sub>(n))
- ▶ Process to be executed: the leftmost of the tree



## El algoritmo de planificación CFS

Con el planificador CFS se realiza un cambio radical en los planteamientos actuales de los planificadores incluidos en Linux, cambiando la planificación de manera que se base en tiempo, en nanosegundos, en vez de colas de ejecución donde hay tareas en espera, sino que se creará un Árbol rojo-negro de búsqueda, en el que se almacenará una línea de tiempo de las futuras tareas que usarán la CPU. Ya no usará los *jiffies*, basado en los tick de la CPU, para expulsar a un proceso.

No usará intervalos de tiempo (quantum) estáticos, sino que se irán modificando dinámicamente según necesidades del sistema, con una granularidad definida en el fichero /proc/sys/kernel/sched\_granularity\_ns definido en nanosegundos.

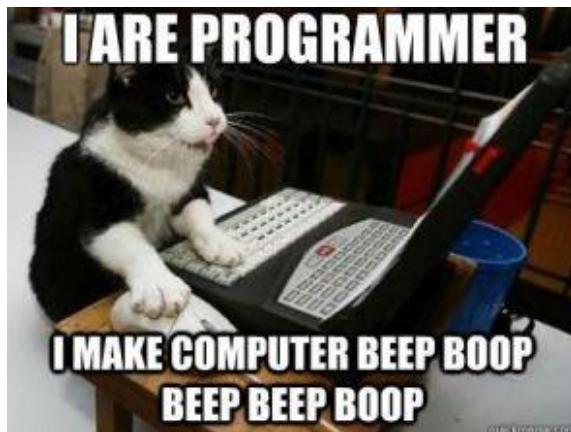
Ejemplo de examen

8.- En un sistema con planificación de colas de realimentación multinivel de tres niveles (expropiativo) se ejecutan tres procesos.

Los tiempos de computación pura de los procesos (total CPU) son los siguientes: P1=60 ciclos, P2=130 ciclos y P3=180 ciclos. Además el proceso P1 realiza una operación de E/S cuando han pasado 20 ciclos de su tiempo de computación. El proceso P2 realiza una operación de E/S cuando han pasado 80 ciclos de su tiempo de computación. El proceso P3 no realiza operaciones de E/S. Cada operación de E/S dura 20 ciclos.

Los cuantos de tiempo asociados a cada cola, a medida que desciende la prioridad de las colas son Cola1: 50, Cola2: 100 y Cola3: 120 ciclos (la Cola1 es la de mayor prioridad). Suponiendo que, inicialmente, todos los procesos llegan al mismo tiempo al sistema de planificación y que el orden inicial en la cola es P1(primer), P2(segundo) y P3 (tercero), completar el diario de ejecución en la tabla siguiente anotando el tiempo en el cual ocurre cada nuevo evento, dónde se encuentra cada proceso en ese momento y la descripción del evento.

Tiempo	Ejecución	Cola1	Cola2	Cola3	Bloqueados	Descripción evento
0	P1	P2,P3				Llegada de los procesos P1,P2, P3 Despacho de P1



- La unidad de punto flotante (FPU) no es un requerimiento hardware imprescindible de un sistema de memoria virtual.
- En caso de fallo de página, el proceso que provoca el fallo está suspendido bloqueado durante el swap-in/out.
- Paginación de memoria sufre fragmentación interna. La segmentación sufre fragmentación externa.
- La política de asignación "Next-Fit" es una política derivada de la asignación "First-Fit".
- La MMU traduce las direcciones virtuales a dir. físicas. (MMU → Unidad gestión memoria)
- Fases de traducción: Compilación (separa código de datos) - Montaje (resuelve cruce entre módulos) - Cargador (Asigna direcciones iniciales a segmentos del código) - Ejecución (dir. lógica → MMU)
- Particiones estáticas: Se divide la memoria en partes (tam. asig.) se asigna a un proceso nuevo un partición ≥ tam. proceso. Se gestiona con tabla de n entradas (a n particiones)
- Particiones dinámica: Proceso nuevo → divide en 2, uno para el proceso, el resto libre. Se gestiona con listas enlazadas.
- Swap-out (Mem → HD), Swap-in (HD → Mem)
- N° entradas (TP) = Tamaño E.L. / Tamaño página
- PTBR: Registro Base Tabla Páginas. Apunta al comienzo de la TP del proceso.
- Llamada a sistema (ficheros): Crear un archivo, borrarlo, abrir, cerrar, leer, escribir, concatenar, buscar, obtener o modificar atributos, renombrarlo.
- FAT → Sistema de ficheros Microsoft. UFS → Sistema Ficheros Unix.