Sistemas Operativos y Redes

Procesos e Hilos

Grado en Ciencia, Gestión e Ingeniería de Servicios

Profesor:

David Granada

david.granada@urjc.es

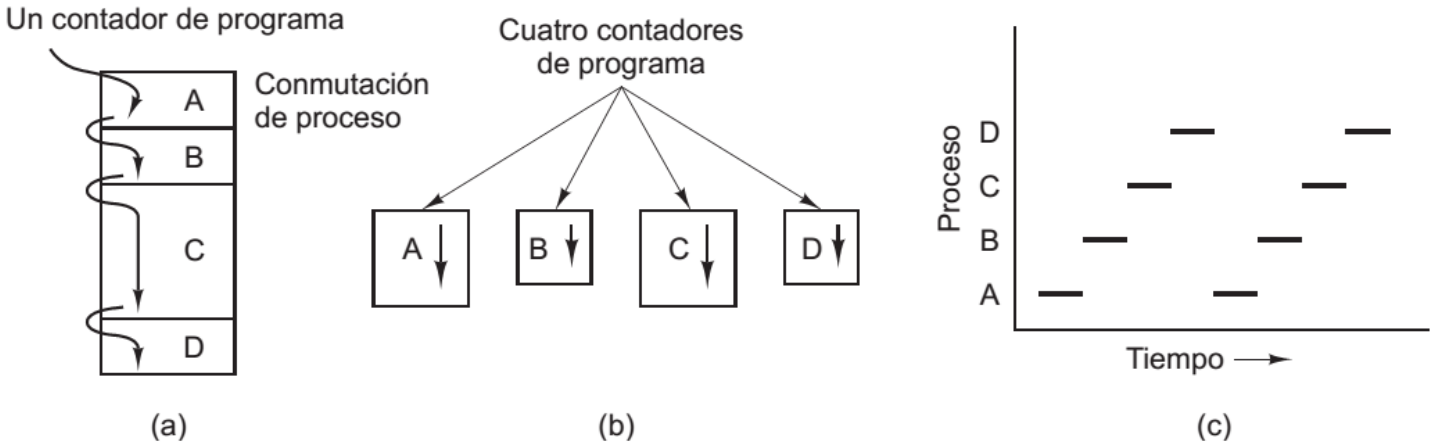
- El concepto más importante en cualquier S.O. es el de proceso:
  - Abstracción de un programa en ejecución
- Todo lo demás está relacionado con este concepto, con lo cual es fundamental su comprensión.

- Los procesos son una de las abstracciones más **antiguas** e **importantes** que proporcionan los S.O.
  - Proporcionan la capacidad de operar **concurrentemente**
  - Convierten una CPU en **varias** CPU virtuales
  - Sin los procesos, la **computación moderna** no podría existir

Procesos

- Un proceso es una **instancia de un programa en ejecución**, incluyendo los valores actuales del contador de programa, los registros y las variables.
- En concepto, cada proceso tiene su propia CPU **virtual**; en la realidad, la CPU **real** conmuta de un proceso a otro.
- Es más fácil pensar en una colección de procesos que se ejecutan en (pseudo) **paralelo**, en lugar de entender cómo **conmuta** la CPU de programa en programa.

- La **conmutación** de la CPU de un proceso a otro se conoce como **multiprogramación**.



(a) Multiprogramación de 4 programas. (b) 4 procesos secuenciales independientes. (c) Sólo hay un programa activo a la vez

- Vamos a suponer que solo hay **una** CPU (actualmente los nuevos chips son **multinúcleo**, con 2, 4 o más CPUs), de este modo supondremos que sólo se puede ejecutar **un proceso a la vez**.

Diferencia entre **proceso** y **programa**

1. Científico. Hornear tarta. Receta. Cocina equipada. Ingredientes.

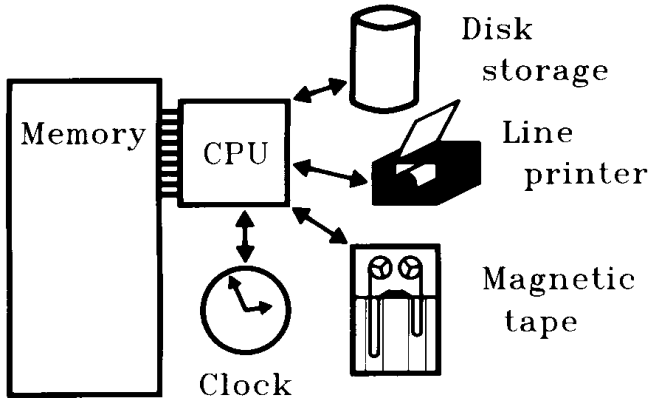
2. Abeja pica hija. Científico registra punto de la receta. Prioridad cuidados médicos.

3. Científico regresa a la tarta. Continúa en el punto donde lo había dejado.

- **Receta** =
  - **Programa** o algoritmo bien expresado
- **Científico** =
  - **Procesador** (CPU)
- **Ingredientes** =
  - **Datos** de entrada
- **Leer** la receta, **ir** cogiendo los ingredientes, **hornear** la tarta =
  - **Proceso**

- Con la picadura de la abeja a la hija, el científico **registra** el punto de la receta en el que estaba (**estado del proceso**).
- Acude a medicar a su hija (**conmuta** a un proceso de **mayor prioridad**, el cual tiene un **programa** distinto: las instrucciones de los primeros auxilios).
- Cuando se ha ocupado de su hija, **vuelve** a la receta (**conmuta** al proceso inicial en el **punto** en el que se había quedado).

- La idea clave es comprender que un **proceso** es una **actividad** de cierto tipo:
  - Tiene un **programa**, una **entrada**, una **salida** y un **estado**
- Varios procesos pueden **compartir** un solo procesador mediante el uso de un algoritmo que permita **planificar la conmutación**



Creación de un proceso

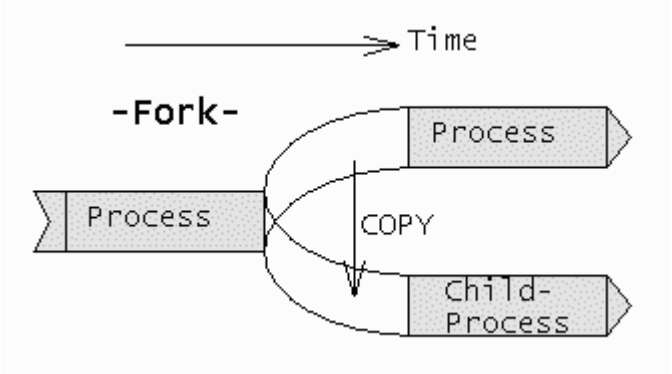
- Los S.O. necesitan un modo de **crear** procesos.
- En sistemas **simples** hechos para ejecutar sólo una aplicación (ej. controlador de microondas), es posible tener todos los procesos que se van a requerir cuando el **sistema inicie**.
- En sistemas de **propósito general** es necesario **crear** y **terminar** procesos según vaya siendo **necesario**.

Creación de un proceso

- Hay **4 eventos principales** que provocan la creación de procesos:
  1. El **arranque** del sistema
     - Daemons correo electrónico, páginas web, etc
  2. La **ejecución**, desde un proceso, de una **llamada al sistema** para creación de procesos.
     - Obtener datos a través de una red. Obtener, procesar y remover los datos
  3. Una **petición de usuario** para crear un proceso
     - Escribir un comando o doble clic en un icono de un programa
  4. El **inicio de un trabajo** por lotes.
     - Mainframes grandes. Envío remoto de procesamiento por lotes, cuando el S.O. tiene lo necesario -> crea un proceso

Creación de un proceso

- Técnicamente, en todos los casos, para crear un proceso es necesario que otro proceso existente ejecute una llamada al sistema de creación de proceso.
- En **UNIX** solo hay una llamada al sistema para crear un proceso: **`fork`**
- En **Windows**: **`CreateProcess`**



Terminación de un proceso

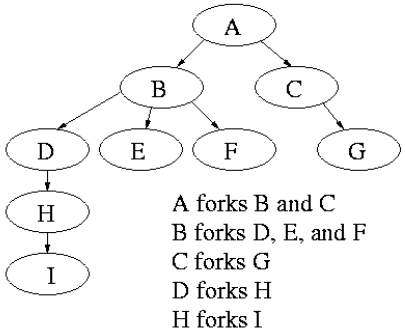
- Un proceso se **crea**, se **ejecuta**, **realiza** su trabajo y tarde o temprano **termina** debido a :
  - Salida **normal** (voluntario)
  - Salida por **error** (voluntario)
  - Error **fatal** (involuntario)
  - **Eliminado** por otro proceso (involuntario)
- La mayor parte termina porque ha concluido su trabajo.

Jerarquías de procesos

- Cuando un proceso crea a otro, el proceso **padre** y el **hijo** continúan **asociados** de diferentes maneras.
- El hijo puede **crear** más procesos, formando una **jerarquía** de procesos.
- Un proceso tiene sólo **un** padre, pero **cero**, **uno**, **dos** o **más** hijos.

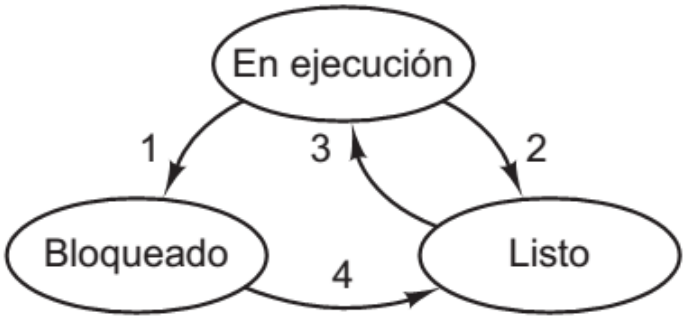
Jerarquías de procesos

- En **UNIX** hay un proceso especial llamado ***init*** en la imagen del inicio del sistema.
  - Cuando se ejecuta, **recupera** cuántas terminales hay
  - Luego utiliza ***fork*** para **crear** un proceso para cada terminal
  - Estos procesos esperan a que alguien **inicie** la **sesión**
  - Si alguien inicia sesión, el proceso de inicio **ejecuta** una **Shell** para aceptar comandos
  - Estos pueden **iniciar más procesos** y así sucesivamente
  - **TODOS** pertenecen a un solo **árbol**, con ***init*** en la raíz.



Estado de un proceso

- A menudo los procesos **interactúan entre ellos** y lo que genera uno de ellos puede ser usado como entrada por otro proceso. Ej: `cat fichero1 fichero2 | grep casa`
  - Concatena dos ficheros y en dicha concatenación busca la palabra “casa” (el output de uno es el input del otro)
- Existen **tres estados** en los que se puede encontrar un proceso:



1. El proceso se bloquea para recibir entrada
2. El planificador selecciona otro proceso
3. El planificador selecciona este proceso
4. La entrada ya está disponible

Implementación de los procesos

- El S.O. mantiene una **tabla de procesos** con información importante acerca del estado del proceso (**contador**, **apuntador de pila, asignación de memoria**, etc.)

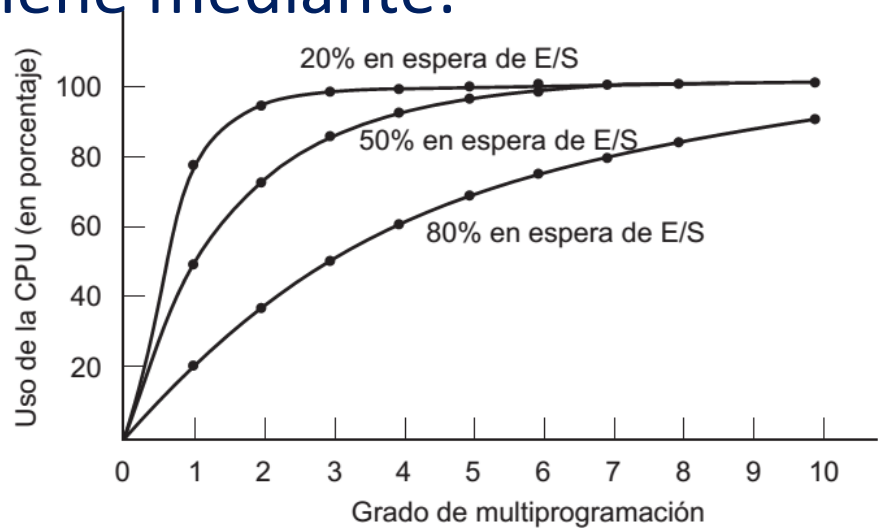
Administración de procesos	Administración de memoria	Administración de archivos
Registros	Apuntador a la información del segmento de texto	Directorio raíz
Contador del programa	Apuntador a la información del segmento de datos	Directorio de trabajo
Palabra de estado del programa	Apuntador a la información 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		

Modelación de la multiprogramación

- Cuando se utiliza la **multiprogramación**, el uso de la CPU se puede **mejorar**.
  - Proceso promedio realiza cálculos sólo el 20% del tiempo en memoria. Con 5 procesos a la vez -> la CPU estará al 100%. (suponiendo que nunca estarán esperando I/O al mismo tiempo)
- Es mejor analizar el uso de la CPU desde un punto de vista **probabilístico**.

Modelación de la multiprogramación

- Supongamos que un proceso **gasta una fracción *p*** de su tiempo **esperando** la operación de **I/O**.
- Con *n* procesos en memoria **a la vez**, la probabilidad de que todos estén **esperando** la I/O es p^n (**CPU inactiva**). Así que el uso de la CPU se obtiene mediante:
  - ***Uso de la CPU = 1 - p^n***



Modelación de la multiprogramación

- Ejemplo: computadora que tiene **512MB** de memoria.
  - **S.O. ocupa 128 MB** y cada programa de usuario **128 MB**
  - Esto permite que hayan **3 programas** de usuario
  - Si el promedio de espera de I/O es del **80**%, el uso de la CPU es:
    - **$1 – 0{,}8^3$, lo que equivale al 49% (redondeando -)**
  - Si **agregamos** 512 MB más de memoria, el sistema puede pasar a una multiprogramación de **siete** vías.
  - En este caso el uso de la CPU se eleva hasta el **79% ($1 – 0{,}8^7$)**
    - Los 512 MB adicionales **mejoran el 30%** del rendimiento
  - Si **agregamos** otros 512 MB:
    - El uso de la CPU **($1 – 0{,}8^{11}$)** pasa del **79% al 91%**. Lo que equivale a una mejora del **12**%.
  - La primera es una buena inversión, la segunda no tanto.

Modelación de la multiprogramación

- Ejercicio: una computadora tiene **256MB** de memoria.
  1. **El S.O. ocupa 64 MB** y cada programa de usuario **48 MB.** ¿Cuántos programas de usuario pueden estar en la memoria?
     - 4
  2. Si el promedio de espera de I/O es del **60**%, ¿Cuál es el uso de la CPU?
     - **$1 - 0{,}6^4$, lo que equivale al 87% (redondeando -)**

     Uso de la CPU = $1 - p^n$
  3. Si **agregamos** 192 MB más de memoria, ¿a cuántas vías de multiprogramación puede pasar el sistema?
     - 8
  4. En este último caso, ¿cuál es el uso de la CPU?
     - **98% ($1 - 0{,}6^8$) (redondeando -)**
  5. ¿Cuál es el porcentaje de mejora del rendimiento?
     - El uso de la CPU pasa del **87% al 98%**. Lo que equivale a una mejora del **11**%.

Modelación de la multiprogramación

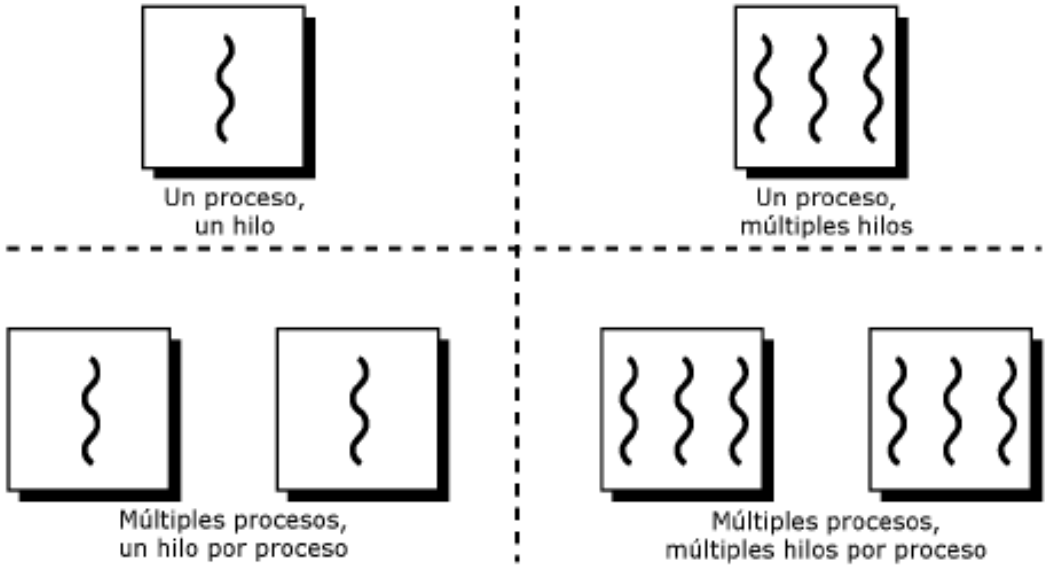
- Ejercicio: una computadora tiene **1024MB** de memoria.
  1. **El S.O. ocupa 256 MB** y cada programa de usuario **128 MB.** ¿Cuántos programas de usuario pueden estar en la memoria?
     - 6
  2. Si el promedio de espera de I/O es del **70**%, ¿Cuál es el uso de la CPU?
     - **$1 - 0{,}7^6$, lo que equivale al 88% (redondeando -)**
  3. Si **agregamos** 512 MB más de memoria, ¿a cuántas vías de multiprogramación puede pasar el sistema?
     - 10
  4. En este último caso, ¿cuál es el uso de la CPU?
     - **97% ($1 - 0{,}7^{10}$) (redondeando -)**
  5. ¿Cuál es el porcentaje de mejora del rendimiento?
     - El uso de la CPU pasa del **88% al 97%**. Lo que equivale a una mejora del **9**%.

Modelación de la multiprogramación

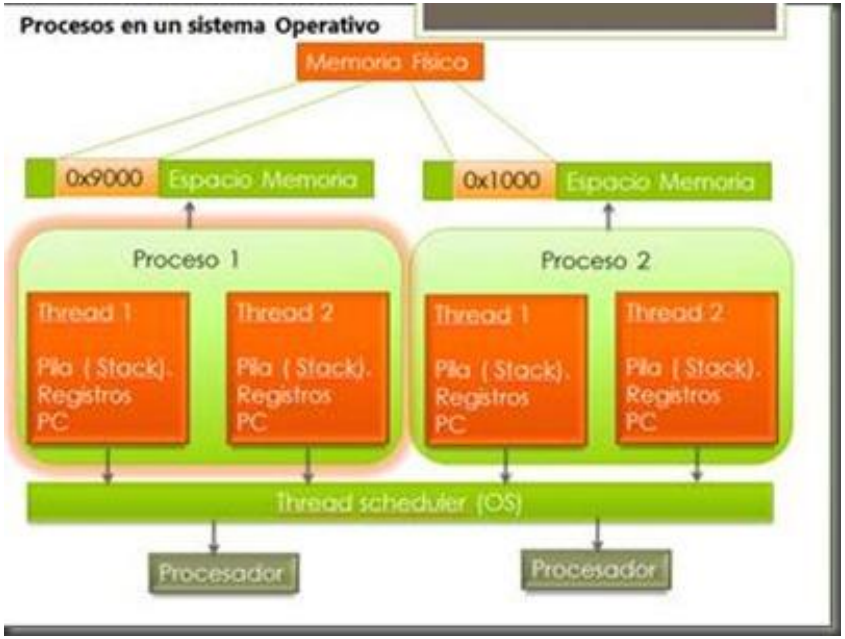
- Ejercicio: una computadora tiene **2 GB** de memoria.
  1. **El S.O. ocupa 256 MB** y cada programa de usuario **64 MB.** ¿Cuántos programas de usuario pueden estar en la memoria?
     - 28
  2. Si el promedio de espera de I/O es del **95**%, ¿Cuál es el uso de la CPU?
     - **$1 - 0{,}95^{28}$, lo que equivale al 76,2%**
  3. Si **agregamos** 1 GB más de memoria, ¿a cuántas vías de multiprogramación puede pasar el sistema?
     - 44
  4. En este último caso, ¿cuál es el uso de la CPU?
     - **89,5% ($1 - 0{,}95^{44}$)**
  5. ¿Cuál es el porcentaje de mejora del rendimiento?
     - El uso de la CPU pasa del **76,2% al 89,5%**. Lo que equivale a una mejora del **13,3**%.

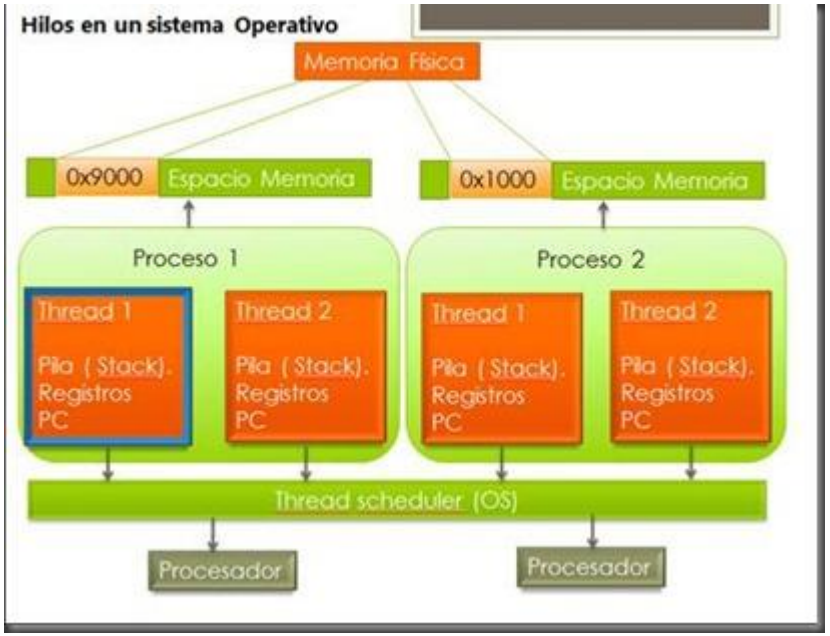
Hilos (Threads)

- **Secuencia de control dentro de un proceso** que ejecuta sus instrucciones de forma independiente
- **Comparten** el espacio de memoria del usuario
- Pueden compartir variables entre **distintos** hilos.
- Puede encontrarse en un **estado**, pero los cambios de contexto entre hilos consumen poco tiempo



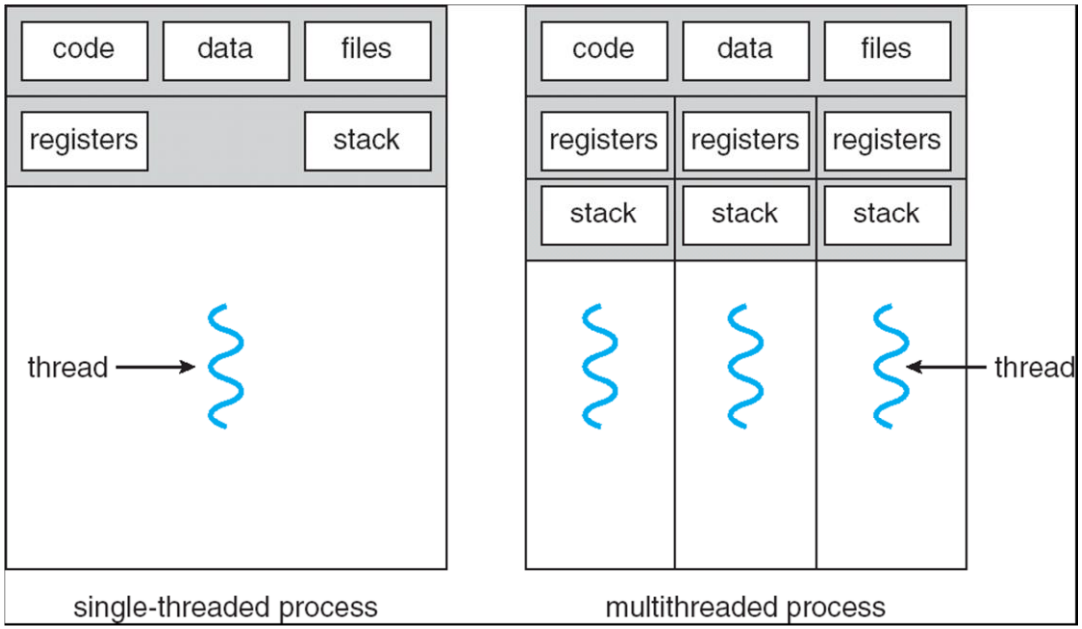
Procesos Vs. Hilos en un S.O.





Hilos

- Los hilos son una **unidad básica** de utilización de la CPU
- Cada hilo tiene su **propio** contador de programa, conjunto de registros de la CPU y pila, además de un identificador **único**
- **Comparten** con otros hilos la sección de código, la sección de datos y otros recursos



Hilos – Ventajas de la programación multihilos:

- **Capacidad de respuesta**: permite a los programas continuar atendiendo al usuario aunque alguna de las tareas que esté realizando el programa sean muy largas
- **Compartición de recursos:** por defecto los hilos comparten la memoria y recursos del proceso al que pertenecen
- **Economía**: es más barato en términos de uso de memoria y demás recursos el crear nuevos hilos respecto a crear nuevos procesos
- **Uso en arquitecturas multiprocesador**: permiten escribir aplicaciones que aprovechen la existencia de más de una CPU en el sistema

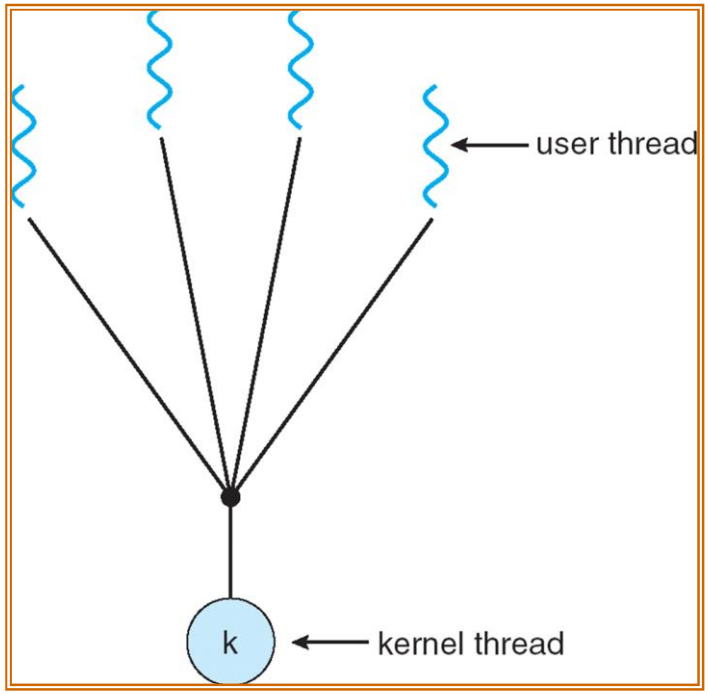
Hilos – Dos tipos:

- **Hilos de usuario**: proporcionados por una librería de usuario
  - Pthreads
  - Hilos de Win32
  - Hilos de Java
- **Hilos del kernel**: proporcionados por el sistema operativo

En último término debe existir una relación entre los hilos de **usuario** y los del **kernel**

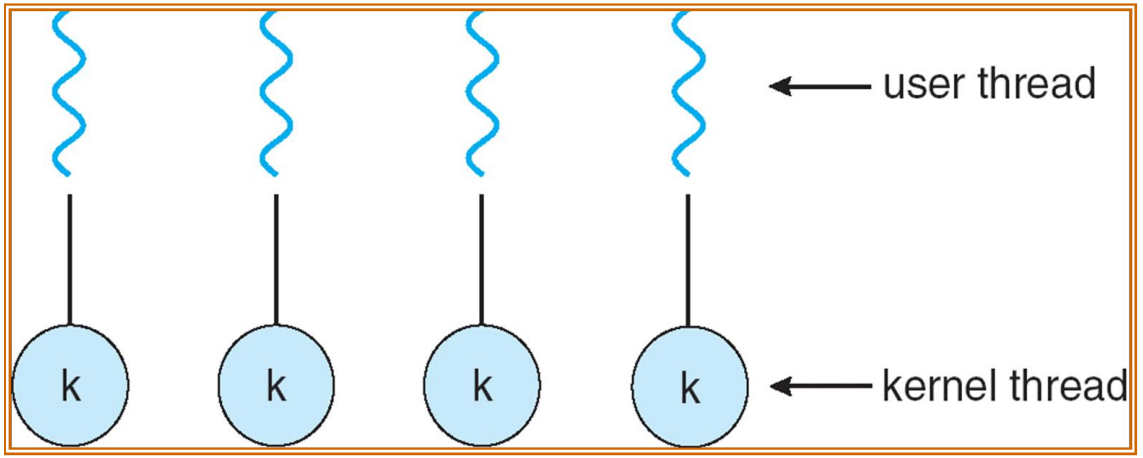
Modelos muchos a uno:

- Los hilos de usuario se implementan con un **único** hilo de kernel
- Las llamadas al sistema **bloquean** al resto de hilos
- No pueden **aprovecharse** de la existencia de **varias** CPUs
- Son "**baratas**" de crear



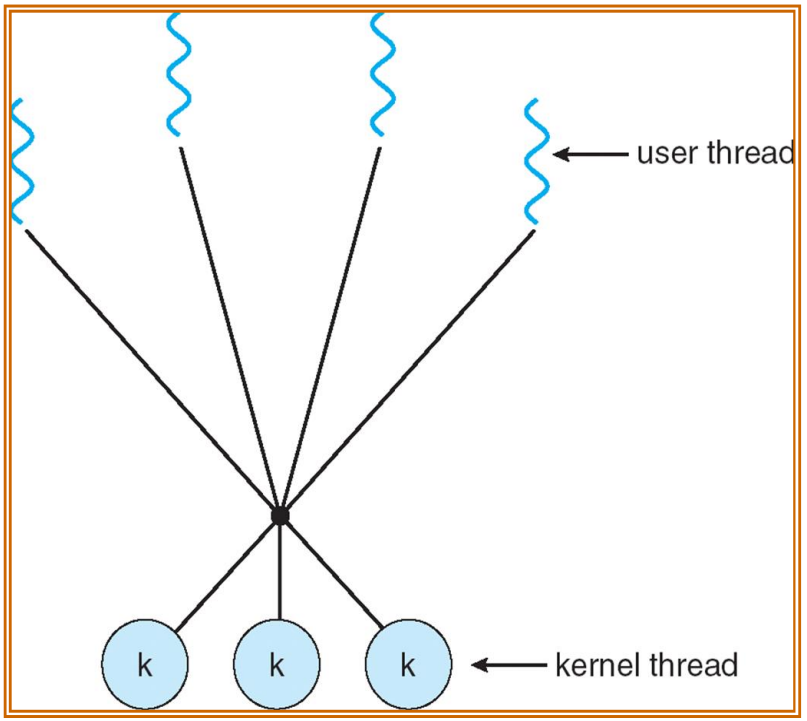
Modelos uno a uno:

- Cada hilo de **usuario** se implementa con un hilo de **kernel**
- Las llamadas al sistema **no bloquean** otros hilos
- Pueden **aprovechar** la existencia de **varias** CPUs
- Son "**caras**" de crear
- Ej:
  - Windows
  - Linux
  - Solaris



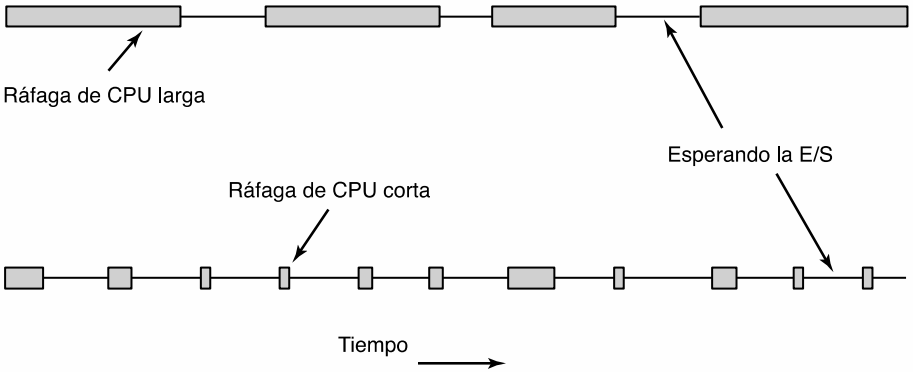
Modelos de hilos

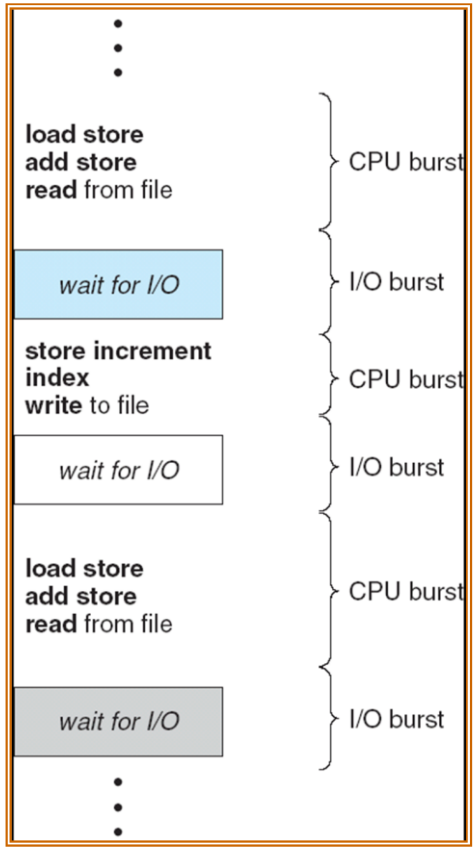
- **Modelos muchos a muchos:**
  - M hilos de usuario se **multiplexan** en N hilos del kernel (M>=N)
  - Las llamadas al sistema **no bloquean** a todos los hilos, solo a algunos
  - Pueden **aprovechar varias** CPUs
  - Más **baratas** que uno a uno y más **caras** que muchos a uno
  - Ej: Solaris 8, HP-UX



Planificación de la CPU

- Los sistemas **multiprogramados** buscan maximizar el uso de la CPU.
- En un proceso se **alternan ráfagas de uso de la CPU** con **ráfagas de Entrada/Salida** (I/O).
- La **duración** y **frecuencia** de las ráfagas de CPU **depende** mucho de cada proceso.





Planificación de la CPU

- Cuando la CPU **queda inactiva** el S.O. debe seleccionar un proceso de la cola de procesos preparados y **cederle** el uso de la CPU.
- La parte del S.O. que selecciona el proceso a ejecutar es el **planificador** (**scheduler**).
- La parte del S.O que realiza la cesión de la CPU a un proceso es el **despachador** (**dispatcher**).

Planificación de la CPU

- Puede ser necesario tomar una **decisión** sobre **planificación** cuando:
  - Un proceso pasa del estado de **ejecución** al estado de **espera**
  - Un proceso pasa del estado de **ejecución** al estado de **preparado**
  - Un proceso pasa del estado de **espera** al estado de **preparado**
  - Un proceso **termina**
- Las decisiones de planificación pueden ser **cooperativas** o **apropiativas**

Diferentes criterios de planificación:

- **Maximizar** el uso de la **CPU**
- **Maximizar** la tasa de **procesamiento** (número de procesos que se completan por unidad de tiempo)
- **Minimizar** el **tiempo** de **ejecución**: es el tiempo que pasa desde que se ordena ejecutar el proceso hasta que termina su ejecución
- **Minimizar** el **tiempo** de **espera**: es el tiempo que pasa en las distintas colas
- **Minimizar** el **tiempo** de **respuesta**: es el tiempo que pasa desde que se envía una solicitud hasta que se empiezan a recibir resultados

Políticas de reparto de tiempo de proceso

- **Objetivo**: intentar satisfacer a todos los procesos y mantener la CPU en ejecución el mayor tiempo posible
- **Políticas de reparto** -> Basadas en **algoritmos** de **planificación**
  - Deciden **qué** procesos tienen que ejecutarse en cada momento y **por qué**
  - **Algoritmos** basados en **equidad**, **eficiencia**, **imparcialidad**, **tiempo** de **respuesta** y **rendimiento**

FCFS: First-Come, First-Served

- Se asigna la CPU a los procesos en función de su **orden de llegada** hasta que termine completamente, una vez terminado se ejecuta el **siguiente**
- Muy **fácil** de **implementar** con una cola **FIFO**. Ejemplo:

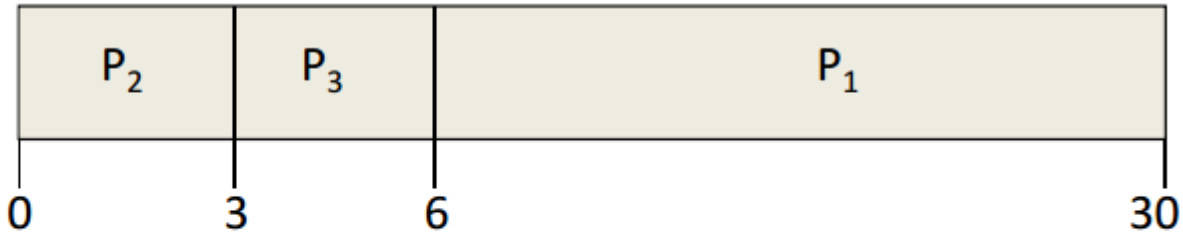
Proceso	Duración	t llegada
P1	24	0
P2	3	0
P3	3	0

Tiempo de espera: P1 = 0; P2 = 24; P3 = 27
Tiempo medio de espera: (0 + 24 + 27) / 3 = 17

FCFS: First-Come, First-Served

- El tiempo de espera depende mucho del **orden de llegada** de los procesos y su duración (**efecto convoy**)
- **Ejemplo**: **mismos procesos de antes** en otro **orden** de llegada

Proceso	Duración	t llegada
P2	3	0
P3	3	0
P1	24	0

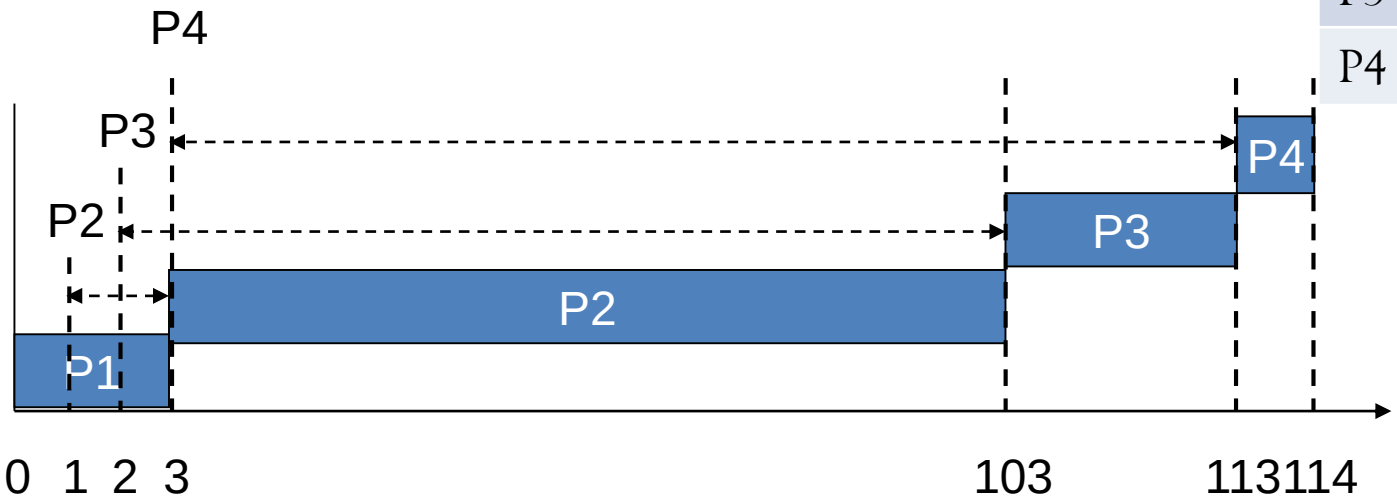


Tiempo de espera: P1 =6; P2 = 0; P3 = 3
Tiempo medio de espera: (6 + 0 + 3) / 3 = 3

FCFS: First-Come, First-Served

- Perjudica a procesos cortos que llegan tras procesos largos. Ej:

Proceso	Duración	T llegada
P1	3	0
P2	100	1
P3	10	2
P4	1	3

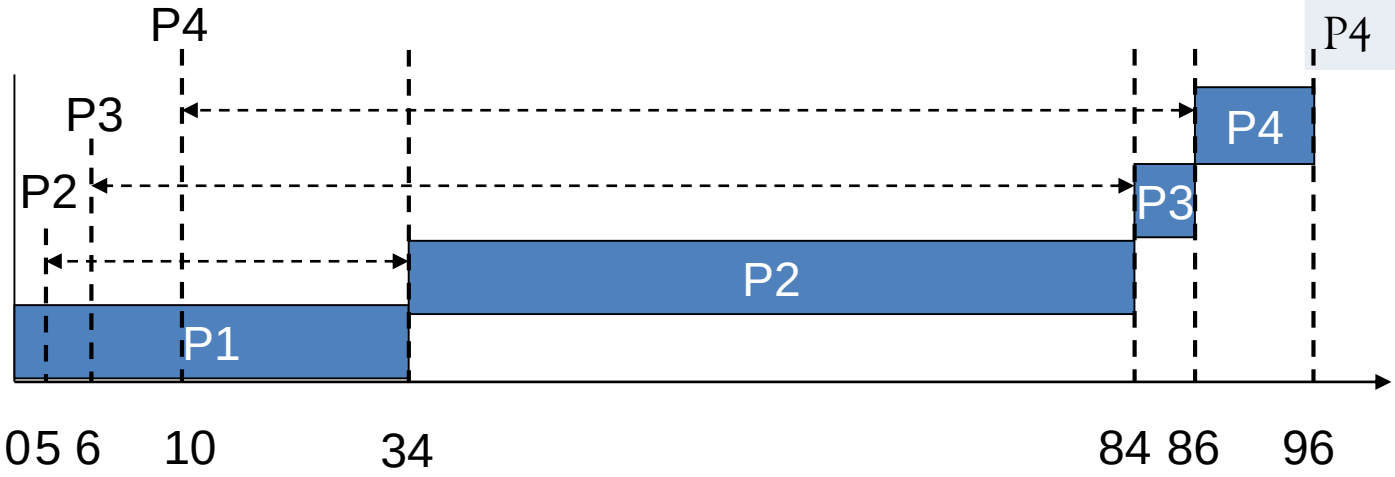


Tiempo de espera:
P1=0; P2=2; P3=101; P4=110
Tiempo medio de espera:
(0 + 2 + 101 + 110) / 4 = 53,25

FCFS: First-Come, First-Served

- Ejercicio: calcular el Tiempo de espera de cada proceso y el Tiempo medio de espera con los datos de la tabla

Proceso	Duración	T llegada
P1	34	0
P2	50	5
P3	2	6
P4	1	10



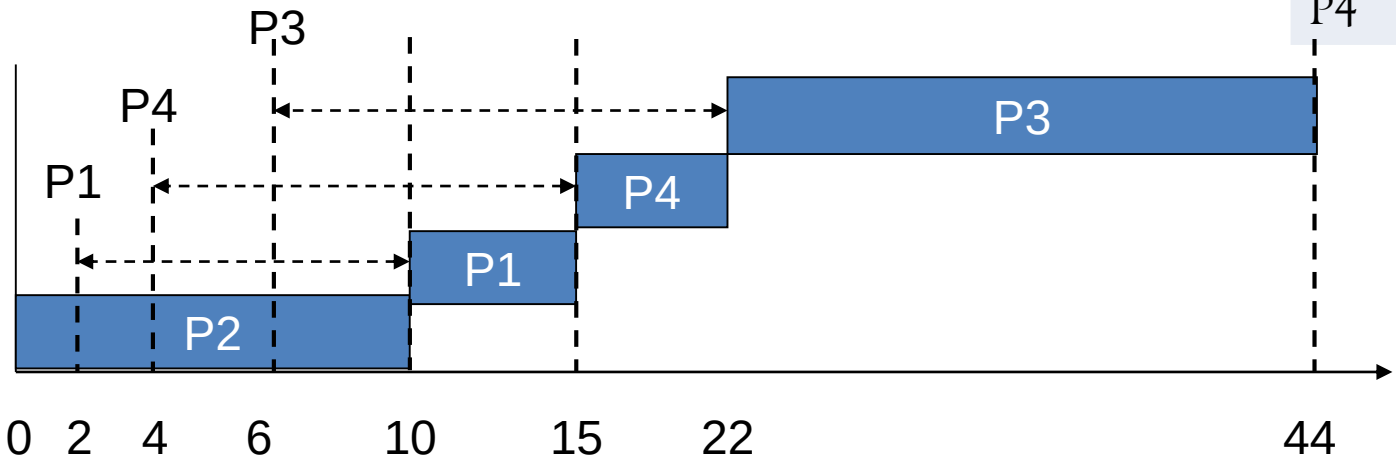
Tiempo de espera:
P1=0; P2=29; P3=78; P4=76
Tiempo medio de espera:
(0 + 29 + 78 + 76) / 4 = 45,75

*Valores no en escala correcta

FCFS: First-Come, First-Served

- Ejercicio: calcular el Tiempo de espera de cada proceso y el Tiempo medio de espera con los datos de la tabla

Proceso	Duración	T llegada
P1	5	2
P2	10	0
P3	22	6
P4	7	4



Tiempo de espera:
P1=8; P2=0; P3=16; P4=11
Tiempo medio de espera:
(8 + 0 + 16 + 11) / 4 = 8,75

Valores no en escala correcta

- **FCFS: First-Come, First-Served**
  - Ejercicio: calcular el Tiempo de espera de cada proceso y el Tiempo medio de espera con los datos de la tabla

Proceso	Duración	T llegada
P1	9	0
P2	19	5
P3	10	8
P4	19	10

SJF: Shortest Job First

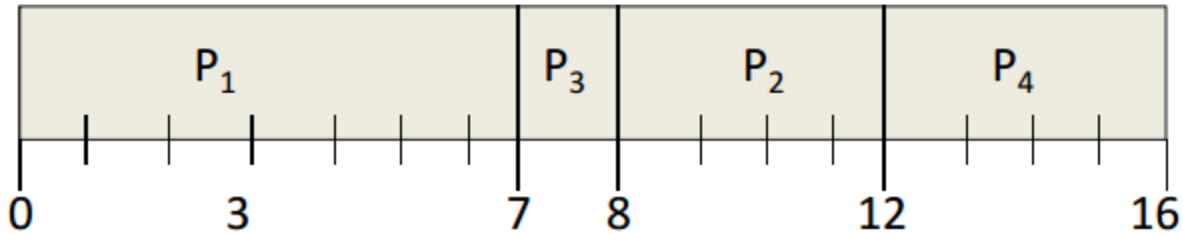
- Cuando la CPU está disponible se asigna al proceso que tiene la ráfaga de **CPU más corta** (quien solicite menos ciclos)
- A **igualdad** de tiempo de ráfaga, se usa **FCFS** para decidir
- Este algoritmo se puede utilizar de forma **cooperativa** o de forma **apropiativa**
- Es el algoritmo **óptimo** en cuanto a tiempo medio de espera
- Mejora a FCFS en cuanto **no perjudica** a procesos **cortos** que llegan después que los largos
- **Inconveniente**: puede **aplazar indefinidamente** los procesos **largos**

SJF: Shortest Job First

- Ejemplo cooperativo

Proceso	Duración	t llegada
P1	7	0
P2	4	2
P3	1	4
P4	4	5

Tiempo de espera: P1 =0; P2 = 6; P3 = 3; P4 = 7
Tiempo medio de espera: (0 + 6 + 3 + 7) / 4 = 4

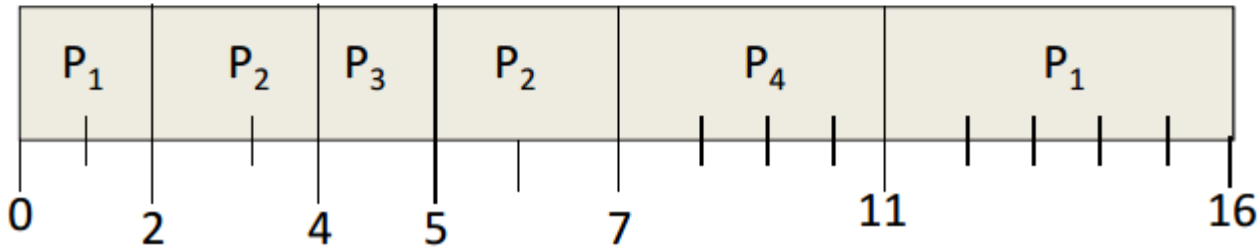


SJF: Shortest Job First

- Ejemplo apropiativo

Proceso	Duración	t llegada
P1	7	0
P2	4	2
P3	1	4
P4	4	5

Tiempo de espera: P1 =9; P2 = 1; P3 = 0; P4 = 2
Tiempo medio de espera: (9 + 1 + 0 + 2) / 4 = 3

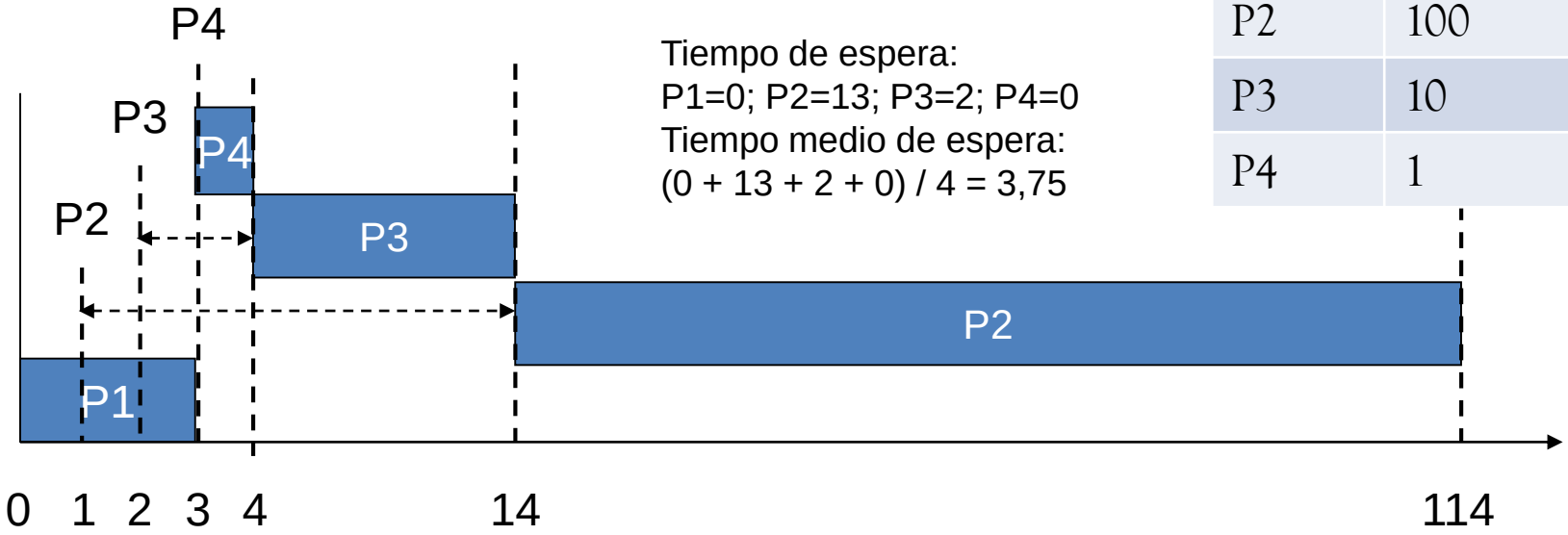


* Si continuasen a llegar procesos cortos, algunos procesos largos como P1 podrían aplazarse indefinidamente

SJF: Shortest Job First

- Ejercicio (cooperativo)

Proceso	Duración	T llegada
P1	3	0
P2	100	1
P3	10	2
P4	1	3



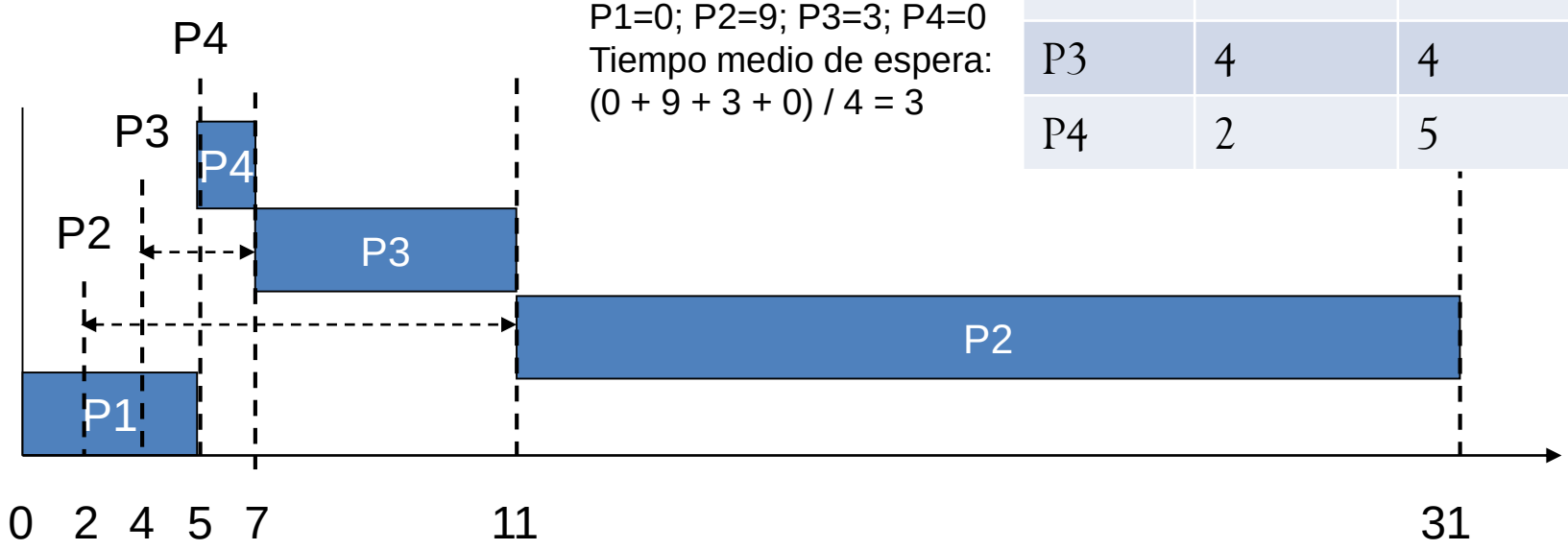
Tiempo de espera:
P1=0; P2=13; P3=2; P4=0
Tiempo medio de espera:
(0 + 13 + 2 + 0) / 4 = 3,75

SJF: Shortest Job First

- Ejercicio (**cooperativo**)

Proceso	Duración	T llegada
P1	5	0
P2	20	2
P3	4	4
P4	2	5

Tiempo de espera:
P1=0; P2=9; P3=3; P4=0
Tiempo medio de espera:
(0 + 9 + 3 + 0) / 4 = 3

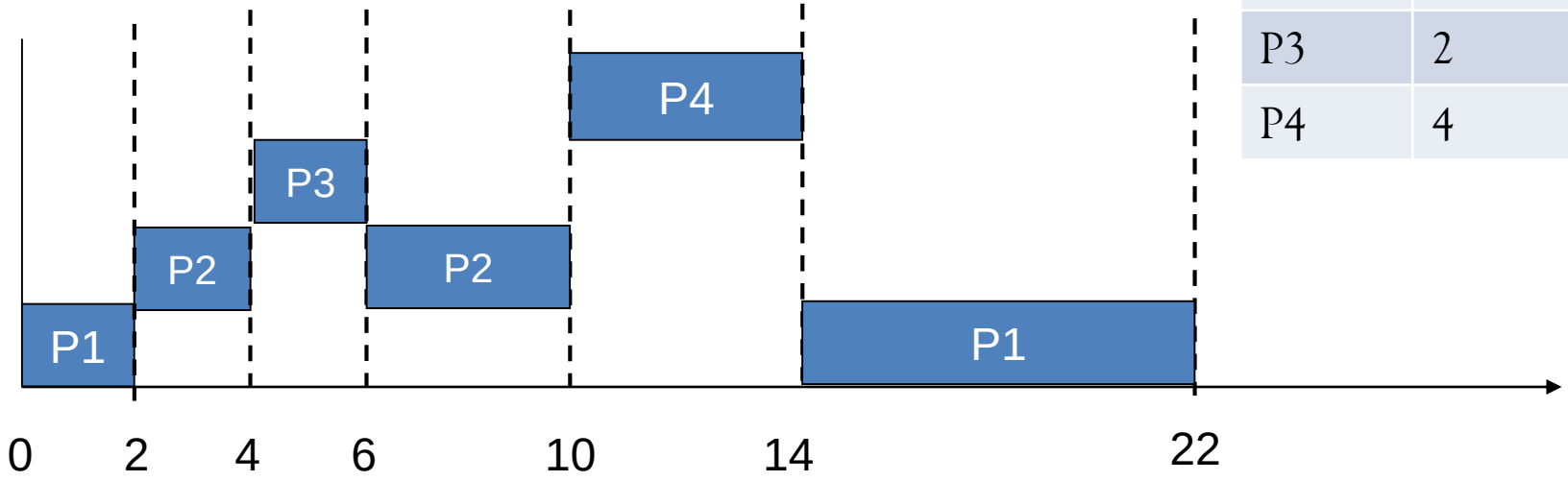


SJF: Shortest Job First

- Ejercicio (**apropiativo**)

Tiempo de espera:
P1=12; P2=2; P3=0; P4=5
Tiempo medio de espera:
(12 + 2 + 0 + 5) / 4 = 4,75

Proceso	Duración	T llegada
P1	10	0
P2	6	2
P3	2	4
P4	4	5



SJF: Shortest Job First

- Ejercicio (**apropiativo**). Calcular tiempo de espera para cada proceso y tiempo medio de espera.

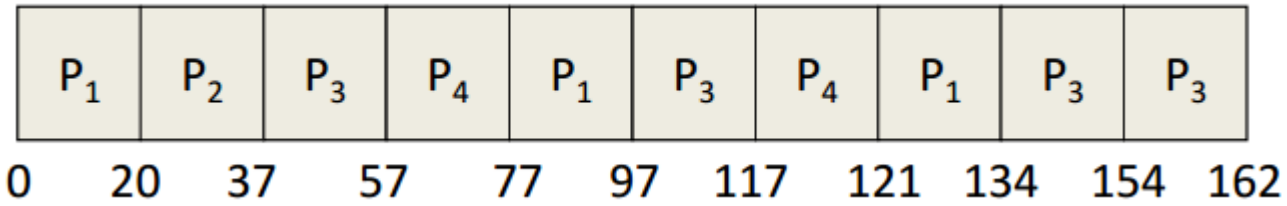
Proceso	Duración	T llegada
P1	8	0
P2	7	1
P3	3	4
P4	1	5

Round Robin: planificación por turnos

- Típico de los sistemas de tiempo compartido. Se caracteriza por la **equidad**
- Es **similar** a **FCFS**, pero cada determinado tiempo (***cuanto***) el S.O. desaloja el proceso que se está ejecutando, lo pone al final de la cola de procesos preparados y el primer proceso de la cola de preparados toma la CPU
- El **rendimiento** del algoritmo depende mucho de la duración del ***cuanto*** de tiempo respecto a la duración media de las ráfagas de CPU
- Un ***cuanto*** muy **grande** -> **misma** situación que **FCFS**
- Un ***cuanto*** muy **pequeño** -> **pérdidas** de **tiempo** por cambios de **contexto** entre procesos

Round Robin: planificación por turnos

- Ejemplo con *cuanto* = 20

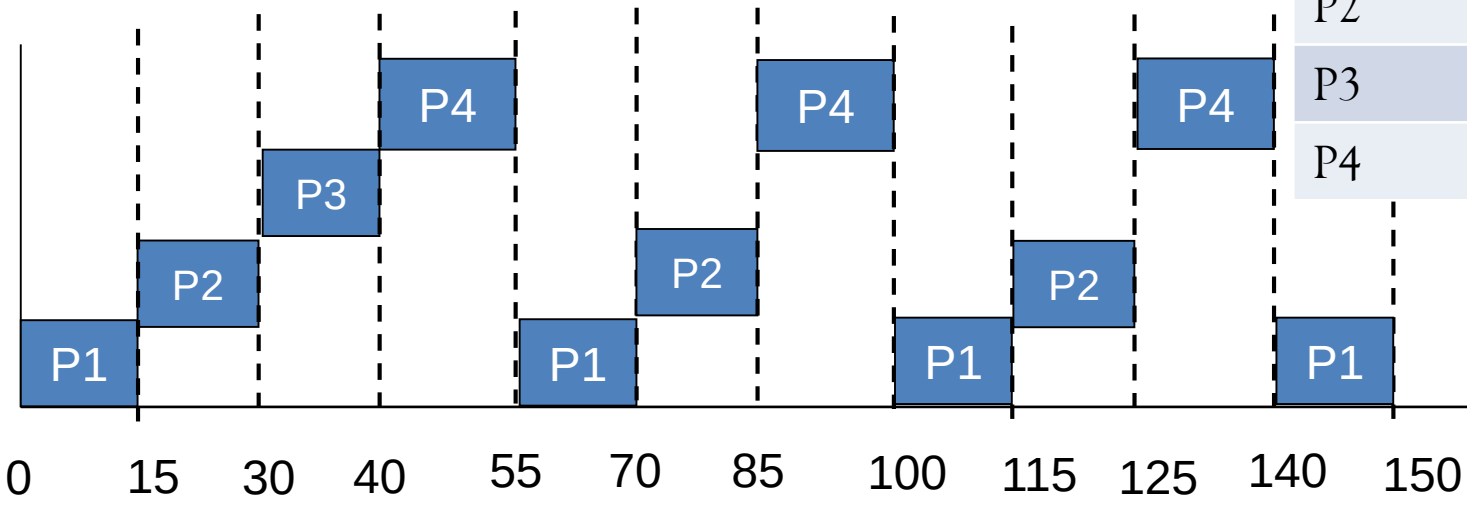


Proceso	Duración	t llegada
P1	53	0
P2	17	0
P3	68	0
P4	24	0

Tiempo de espera: P1 =81; P2 = 20; P3 = 94; P4 = 97
Tiempo medio de espera: (81 + 20 + 94 + 97) / 4 = 73

Round Robin: planificación por turnos

- Ejercicio con *cuanto* = 15

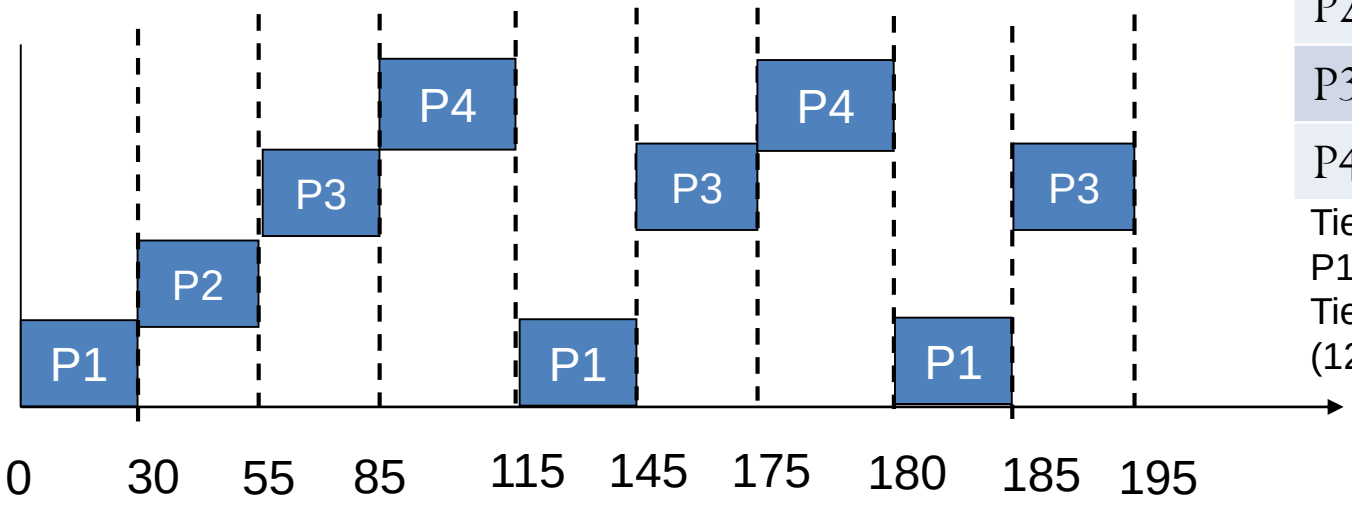


Proceso	Duración	T llegada
P1	55	0
P2	40	0
P3	10	0
P4	45	5

Tiempo de espera:
P1=95; P2=85; P3=30; P4=90
Tiempo medio de espera:
(95 + 85 + 30 + 90) / 4 = 75

Round Robin: planificación por turnos

- Ejercicio con *cuanto* = 30



Proceso	Duración	T llegada
P1	65	0
P2	25	5
P3	70	5
P4	35	10

Tiempo de espera:
P1=120; P2=25; P3=120; P4=135
Tiempo medio de espera:
(120 + 25 + 120 + 135) / 4 = 100

Round Robin: planificación por turnos

- Ejercicio con *cuanto* = 10. Calcular tiempo de espera para cada proceso y tiempo medio de espera.

Proceso	Duración	T llegada
P1	27	0
P2	15	0
P3	20	20
P4	8	25

HPRN: Por mayor índice de penalización

- Ante varios procesos preparados, la CPU se asigna al que está sufriendo un **mayor índice de penalización**.
- Si es cooperativo, la **Penalización** es: P = T/t, donde
  - T = W+t
  - W = tiempo de espera del proceso (Wait)
  - t = tiempo de duración del proceso
- Es decir: P = (W+t)/t = W/t +1
- **Mejora a SJF**, no aplaza indefinidamente a procesos largos
- **Sigue perjudicando procesos cortos** que llegan cuando ya han comenzado procesos largos

HPRN: Por mayor índice de penalizaci

- Ejemplo (Cooperativo)

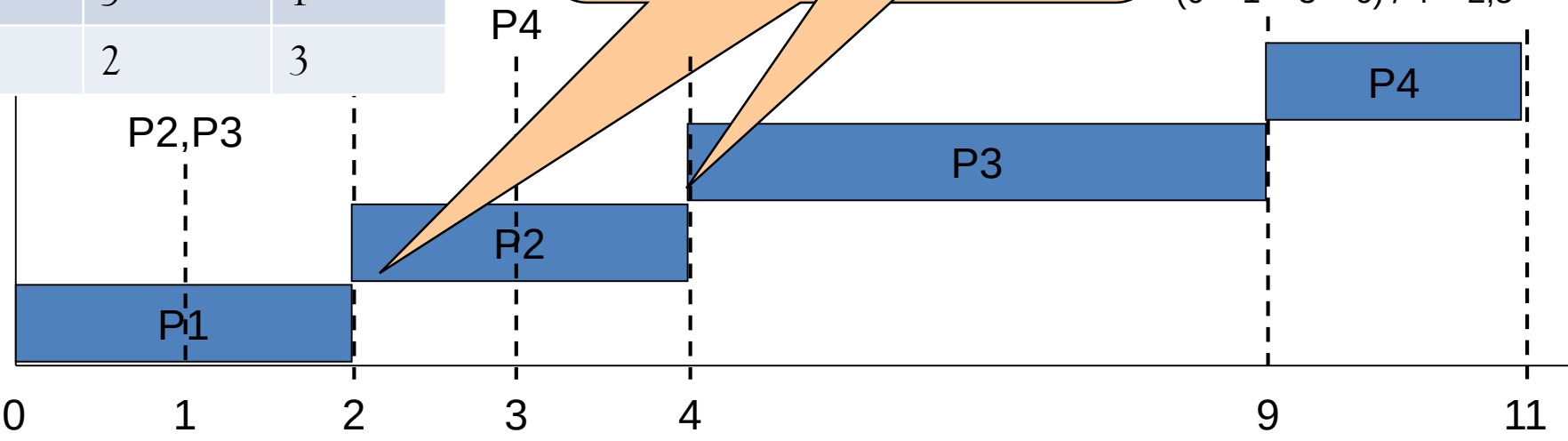
Proceso	Duración	T llegada
P1	2	0
P2	2	1
P3	5	1
P4	2	3

P3: W/t + 1 = 3/5 + 1 = 1,6
P4: W/t + 1 = 1/2 + 1 = 1,5
¡Gana P3!

P2: W/t + 1 = 1/2 +
P3: W/t + 1 = 1/
¡Gana

1,2

Tiempo de espera:
P1=0; P2=1;
P3=3; P4=6
Tiempo medio de espera:
(0 + 1 + 3 + 6) / 4 = 2,5

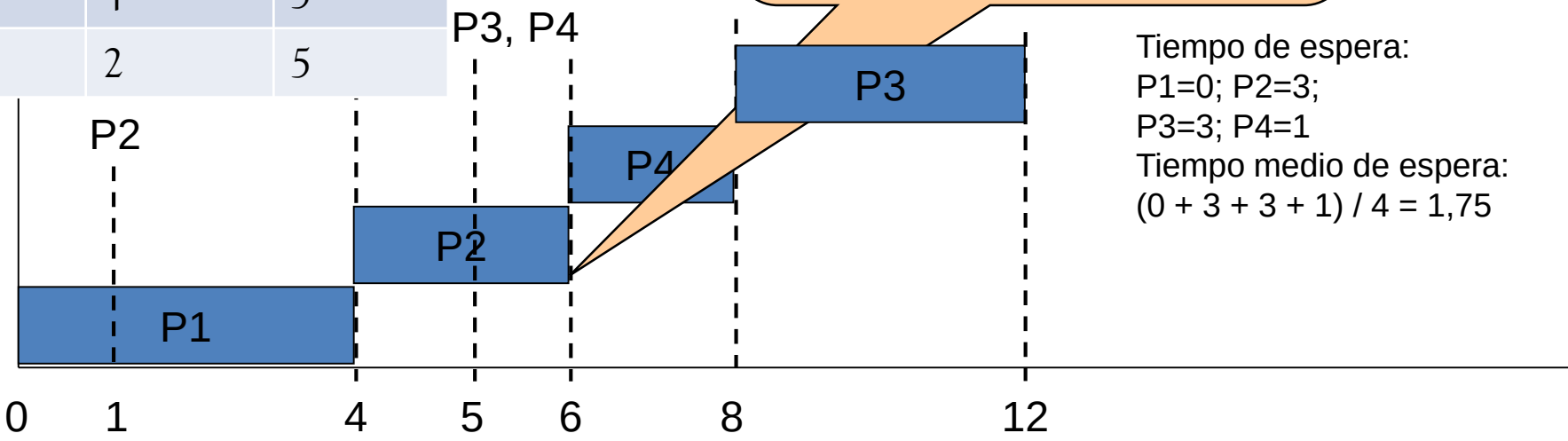


HPRN: Por mayor índice de penalización

- Ejercicio (Cooperativo)

Proceso	Duración	T llegada
P1	4	0
P2	2	1
P3	4	5
P4	2	5

P3: W/t + 1 = 1/4 + 1 = 1,25
P4: W/t + 1 = 1/2 + 1 = 1,5
¡Gana P4!



Tiempo de espera:
P1=0; P2=3;
P3=3; P4=1
Tiempo medio de espera:
(0 + 3 + 3 + 1) / 4 = 1,75

HPRN: Por mayor índice de penalización

- Ejercicio (Cooperativo)

Proceso	Duración	T llegada
P1	5	0
P2	3	3
P3	4	3
P4	2	6

Tiempo de espera:
P1=0; P2=2;
P3=5; P4=6
Tiempo medio de espera:
(0 + 2 + 5 + 6) / 4 = 3,25

Planificación por prioridades

- Cada proceso tiene una prioridad determinada utilizada como criterio
- En caso de empate, se combina con otro algoritmo

- (Nota para el alumno: CONSULTAR BIBLIOGRAFÍA Y RECURSOS WEB)

Planificación con múltiples colas

- El resto de algoritmos únicamente consideran una cola de procesos
- Cada cola puede utilizar un algoritmo diferente

- (Nota para el alumno: CONSULTAR BIBLIOGRAFÍA Y RECURSOS WEB)

Planificación por reparto equitativo

- (Nota para el alumno: CONSULTAR BIBLIOGRAFÍA Y RECURSOS WEB)

- Ejercicio:
  - Simular la repartición del tiempo de procesador según los algoritmos **FCFS**, **SJF (apropiativo)** y **HPRN**, con los siguientes datos y obtener los tiempos de espera de cada proceso y el tiempo medio de espera

Proceso	T llegada	Duración
P1	0	2
P2	1	3
P3	1	5
P4	3	1

Sistemas Operativos y Redes

Procesos e Hilos

Grado en Ciencia, Gestión e Ingeniería de Servicios

Profesor:

David Granada

david.granada@urjc.es