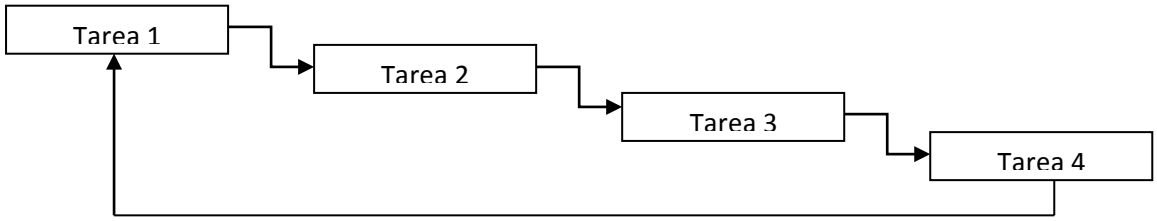
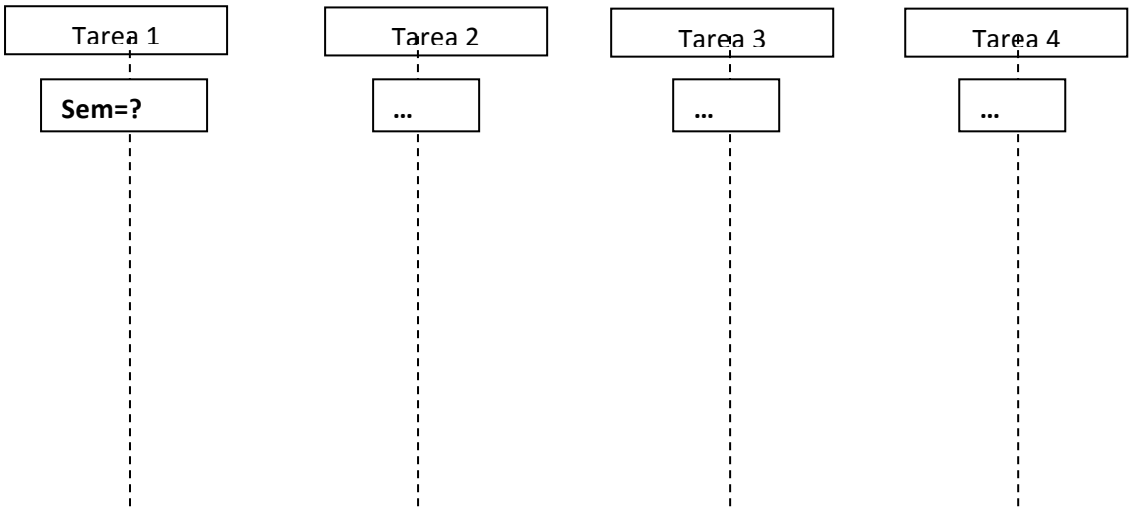
Ejercicio 1

Se desea implementar una aplicación de *n* procesos concurrentes, donde cada proceso ejecuta una determinada tarea en forma de *pipeline.* Un ejemplo de esta aplicación con n=4 sería la siguiente:



Los procesos ejecutarán de manera ordenada, desde el primero hasta el último. La primera tarea debe comenzar la ejecución mientras que el resto de tareas se suspenden. Cada tarea calcula sus resultados y, antes de finalizar, notifica a la siguiente tarea que puede comenzar. Los resultados de cada tarea son privados, es decir, no necesitan compartirse con otras tareas. Se pide:

a) Diseñar un programa que permita la sincronización de los procesos para n=4 utilizando procesos ligeros y semáforos. Para ello, complete la siguiente plantilla indicando el valor inicial del semáforo y las primitivas de sincronización genéricas necesarias para llevar a cabo la sincronización (wait y signal).



b) Implementación de la aplicación en el lenguaje de programación C para n=4, de acuerdo al diseño anterior.

Ejercicio 2

Dado el sistema de ficheros de la figura, que tiene las siguientes características:

- Tamaño de bloque: 1024 bytes.
- Tamaño de dirección de bloque: 2 bytes.
- Número de sectores por bloque: 2
- Tiempo de lectura de un sector: 1 ms.
- Cada i-nodo ocupa un bloque.
- Campos de un i-nodo:
  - Identificador de i-nodo (ID)
  - Metadatos (atributos del fichero, identificador propietario y grupo, etc.)
  - Tipo de elemento: directorio (dir) , fichero (fil) o enlace (lnk) .
  - Contador de enlaces (CE)
  - 1 punteros directos (PD) ,
  - 1 puntero indirecto simple (PIS)
  - 1 puntero indirecto doble (PID) .

La siguiente figura muestra una configuración del sistema de ficheros. Un valor en blanco significa que la entrada asociada está vacía (es tipo void/nill) .

Bloque 0	Bloque 1	Bloque 2	Bloque 3	Bloque 4	Bloque 5	Bloque 6	Bloque 7	Bloque 8
Superbloque i-nodo raíz: 0	ID: 0 Metadatos Tipo: dir CE: 3 PD: 51 PIS: PID:	ID: 1 Metadatos Tipo: fil CE: 1 PD: 100 PIS: PID:	ID: 2 Metadatos Tipo: fil CE: 1 PD: 103 PIS: 52 PID:	ID: 3 Metadatos Tipo: dir CE: 4 PD: 53 PIS: PID:	ID: 4 Metadatos Tipo: dir CE: 2 PD: 54 PIS: PID:	ID: 5 Metadatos Tipo: lnk CE: 1 PD: 55 PIS: PID:	ID: 6 Metadatos Tipo: dir CE: 2 PD: 56 PIS: PID:	ID: 7 Metadatos Tipo: fil CE: 1 PD: 120 PIS: 121 PID: 57

Bloque 51	Bloque 52	Bloque 53	Bloque 54	Bloque 55	Bloque 56	Bloque 57	Bloque 58	Bloque 59
. 0 .. 0	104 105	. 3 .. 0	. 4 .. 3	/Murcia	. 6 .. 3	130 58	131 132	

Madrid 1 Lugo 2 Murcia 3	106	Norte 4 Sur 6 Centro 7	Ciudad 5				133	

Se pide:

1. Representar la estructura del árbol de ficheros/directorios. ¿Qué problema puede existir al recorrer la estructura anterior en la búsqueda de un fichero?
2. ¿Cuál es el tamaño máximo que puede tener un fichero?
3. Describa cómo se realizan la siguiente operación y qué cambios se efectuarían en el sistema de ficheros.
   rm /Murcia/Norte/Ciudad
4. Calcule el tiempo necesario para leer el primer byte del fichero /Madrid
5. Calcule el tiempo necesario para leer el último byte del fichero /Murcia/Centro

NOTA: Comando *rm*: borra un fichero.

Ejercicio 3

Se desea implementar una interfaz que mejore el acceso al sistema de ficheros de cara a cierto tipo de necesidades. Este sistema estará basado en las llamadas estándar POSIX e incluirá la siguiente interfaz de usuario:

- `int sustituir(char *fichero_in, char *fichero_out, char car_viejo, char car_nuevo)`
  - Recibe la ruta absoluta de un fichero a leer, la ruta absoluta de un fichero a escribir, un carácter a sustituir y el carácter por el que sustituir el parámetro anterior. Abrirá el fichero a leer y copiará los caracteres en el fichero de salida, reemplazando todas las ocurrencias de **car_viejo** por **car_nuevo** en el proceso.
  - Devuelve el número de sustituciones realizadas o -1 si se ha producido algún problema.

Se pide:

a) Describa el pseudocódigo de la función **sustituir**.
b) Implemente la función **sustituir** en lenguaje C en base a la interfaz proporcionada.
c) Implementar una función **main** que reciba como parámetros dos nombres de fichero y dos caracteres (viejo y nuevo) y controle que se han recibido exactamente estos cuatro

parámetros. En caso de no recibirlos, mostrará un error y terminará el programa. En caso contrario, llamará a **sustituir** con los mismos e imprimirá el resultado.

Ejercicio 4

Se quiere implementar un programa que juegue al juego conocido como "los chinos". El código inicial es una función *main* y una variable global *monedas_totales*. El programa deberá realizar lo siguiente:

- La función main creará 3 hilos pasándole a cada hilo un ID.
- Cada hilo generará una cantidad aleatoria de monedas entre 0 y 5.
- Cada hilo generará una estimación de monedas totales formado por sus monedas más una cantidad aleatoria entre 0 y 10.
- Sumará su cantidad de monedas a *monedas_totales.*
- Una vez TODOS los hilos hayan sumado sus monedas, cada hilo comprobara si SU estimación es igual al *monedas_totales*. Si lo es, imprimirá por pantalla su ID y la frase "Yo gano".

Se pide:

a) Implemente el código necesario para que la función *main* cree 3 hilos y espere por ellos, pasando a cada hilo un identificador (int).
   Implemente además la función de los hilos. Puede utilizar la función rand()
   - rand() % X; genera un número entre 0 y X - 1

b) Implemente los mecanismos de sincronización necesarios para acceder a la variable *monedas_totales* sin que se den condiciones de carrera.

c) Implemente el mecanismo de sincronización necesario para que ningún hilo compruebe si ha ganado ANTES de que todos los hilos hayan terminado de sumar su cantidad a *monedas_totales*.