

## FUNDAMENTOS DE SISTEMAS OPERATIVOS

### Unix (IV): Concurrencia

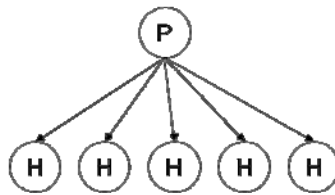
#### Hoja de problemas nº 4 grupos L1 y L2

**NOTA:** Se aconseja seguir la siguiente dinámica de trabajo:

1. Empezar haciendo los ejemplos de los apuntes referentes a creación de procesos pesado.
2. Realizar los ejercicios referentes a este tipo de procesos (Conc\_1, Conc\_2).
3. Realizar los ejemplos de clase referentes a creación de hilos.
4. Realizar los ejercicios referentes a este tipo de procesos (Conc\_3 y Conc\_4).
5. Realizar los ejercicios sobre coordinación mediante semáforos (Conc\_5 y Conc\_6), una vez vistos.

### Ejercicios Procesos Pesados

**Conc\_1.** Usando procesos pesados (`fork()`) realizar un programa C que cree 4 procesos hijos, tal que todos sean hermanos entre sí tal como se muestra en la figura, es decir, que todos sean hijos directos del programa principal `main()`.



Cada hijo escribirá un mensaje en la salida estándar indicando que es un proceso hijo, su número de hijo, cuál es su PID y cuál es el PID del padre (se puede usar para esto las funciones `getpid()` y `getppid()`). Si la creación de hijos es correcta el PID del padre debe ser el mismo para todos.

**Conc\_2.** Vamos a comprobar de manera práctica la ejecución concurrente de procesos. Modificar el ejercicio **Conc\_1** para que ahora cada hijo haga lo siguiente: escribirá en orden creciente 100 números, uno en cada línea y con tres cifras. Los números estarán "encolumnados" a partir de la columna  $n$ , siendo  $n$  el número de hijo, usando como separador de columnas el tabulador. El primer número de cada hijo (para diferenciar la secuencia) será el  $(100 * n)$ . En la figura aparece un ejemplo de la salida que generará cada hijo.

Hijo 0
000
001
002
...

Hijo 1
100
101
102
...

Hijo 2
200
201
202
...

Hijo 3
300
301
302
...

Ejecutar el proceso varias veces. La salida no debe ser la misma ¿Por qué?

### Ejercicios Hilos

**Conc\_3.** Repetir el ejercicio **Conc\_2** pero ahora usando hilos.

**Conc\_4.** Vamos ahora a practicar la comunicación entre hilos usando la memoria compartida, es decir, las variables globales. Vamos a usar dos variables globales (compartidas) una de tipo entero que se inicializará a 0 y un array de 3 cadenas de caracteres de tamaño 80 caracteres. El programa generará 3 hilos donde cada uno hará lo siguiente: mirará la variable entera, si vale 0 la inicializará a 8, si su valor es distinto de 0 (ya la ha inicializado un hilo anterior) incrementará en uno su valor. Una vez hecho esto almacenará en la posición del array de cadenas correspondiente a su número de hilo (el 0 en la posición 0, el 1 en la 1 y el 2 en la 2) el siguiente mensaje (cadena): "Mensaje del hilo  $n$ . Valor anterior de la variable  $x$ . Valor nuevo  $y$ ", donde  $n$  es el número de hilo,  $x$  el valor que tenía la variable entera antes de que la modificara e  $y$  el valor de la variable entera tras ser modificada. Con estos mensajes podremos seguir el orden de modificación. Finalmente el hilo principal, tras la finalización de cada uno de los hilos mostrará por la salida estándar el valor final de la variable entera y los mensajes almacenados en el array de cadenas.

## Ejercicios Semáforos

- Conc\_5.** Repetir el ejercicio **Conc\_4** sobre hilos, pero ahora haciendo, mediante semáforos, que el acceso a las variables compartidas (las globales) sea mutuamente excluyente en los hilos hijos, es decir, que mientras uno las esté modificando el resto no puedan acceder a ellas. Para hacer esto sólo hay que seguir el ejemplo mostrado en la última transparencia, donde se muestra el uso de semáforos para exclusión mutua.
- Conc\_6.** Vamos a probar ahora el uso de semáforos genéricos para coordinar dos hilos, en un ejemplo típico y simple de productor-consumidor. Crear como variable externa (global) un array de 5 números de tipo entero. El hilo 0, por ejemplo, almacenará en ese array de manera consecutiva y cíclica valores enteros entre 1 y 50 (será el productor). El hilo 1 los leerá en el orden en que fueron introducidos, mostrándolos por pantalla (será el consumidor). El hilo 0 escribirá datos, de la manera indicada, mientras tenga espacio en el array, hasta un máximo de 50. El hilo 1 leerá datos del array mientras haya datos, hasta el máximo de 50, "liberando", mediante el uso de semáforos, las posiciones leídas para que el hilo 0 pueda seguir almacenando datos.

**Pista:** necesitamos dos semáforos genéricos uno inicializado a 5 (número máximo de datos que caben en el array) que será usado por el productor y que llamaremos, para identificarle aquí, *sem\_prod*, y otro inicializado a 0 que será usado por el consumidor, que llamaremos aquí *sem\_con*. El productor antes de almacenar un dato consultará *sem\_prod* (operación *sem\_wait*), si no vale 0 quiere decir que hay espacio escribiendo en la posición que corresponda el valor que corresponda; una vez hecho esto "avisará" al consumidor que hay datos incrementando su semáforo de control *sem\_con* (operación *sem\_post*). El consumidor por su parte, antes de leer un dato tendrá que ver si hay consultando *sem\_con* (operación *sem\_wait*); si hay leerá el dato correspondiente a la posición que toque y "avisará" al producto que tiene un hueco libre incrementando el semáforo *sem\_prod* (operación *sem\_post*).

Si todo funciona correctamente tenemos que ver por pantalla una secuencia de enteros entre 1 y 50. Añadir, para seguir ejecución, mensajes de inicializado y finalizado cada hilo, por ejemplo.