SISTEMAS OPERATIVOS: Informe práctica 3

Ejercicio 2: memoria compartida

En este ejercicio, se pedía realizar un programa que crease un cierto número de procesos hijos pasado como argumento de entrada. Cada hijo, pide por pantalla que se introduzca un nombre, lo escribirá en la memoria compartida creada por el padre e incrementará en una unidad el contador de dicha memoria. Posteriormente, el padre lee el nombre y el identificador escrito por el hijo y lo imprime por pantalla.

En nuestra implementación, realizamos un bucle en el que el padre crea la memoria compartida en la primera iteración y crea los procesos hijo. Cada hijo espera un tiempo aleatorio usando la función random, se une a la memoria compartida y después de realizar la petición, escribe el nombre introducido en la memoria, lee el último identificador y lo incrementa en una unidad. Finalmente, envía la señal SIGUSR1 al padre y termina. Por su parte, el padre espera a recibir dicha señal mediante la función pause y la maneja imprimiendo el nombre y el identificador por pantalla. Por este motivo, el puntero a la memoria compartida es una variable global.

Si el padre no esperase a cada hijo, estos procesos se ejecutarían al azar. De manera que, tanto los hijos como el padre accederían a la memoria compartida aleatoriamente y el padre imprimiría por pantalla datos sin sentido.

Salida de la ejecución:

```
homerjr:~/workspace/soper/Práctica 3 $ ./ejercicio2
Se debe pasar un número entero positivo como único argumento
homerjr:~/workspace/soper/Práctica 3 $ ./ejercicio2 d
El argumento de entrada debe ser un entero positivo
homerjr:~/workspace/soper/Práctica 3 $ ./ejercicio2 2 d
Se debe pasar un número entero positivo como único argumento
homerjr:~/workspace/soper/Práctica 3 $ ./ejercicio2 5
Introduzca el nombre de usuario:
Andres
Nombre de usuario: Andres
Identificador: 6202
Introduzca el nombre de usuario:
Antonio
Nombre de usuario: Antonio
Identificador: 6203
Introduzca el nombre de usuario:
Alejandro
Nombre de usuario: Alejandro
Identificador: 6204
Introduzca el nombre de usuario:
Aitor
Nombre de usuario: Aitor
Identificador: 6205
Introduzca el nombre de usuario:
Arturo
Nombre de usuario: Arturo
Identificador: 6206
```

Ejercicio 4: biblioteca de semáforos

En el ejercicio cuatro, se pedía implementar las funciones de la biblioteca de semáforos. En el fichero semáforos.h, a parte de la declaración de las funciones, hemos añadidos tres macros

que definen lo que devuelven las funciones: OK si no se ha producido ningún error, ERROR si se ha producido un error al llamar al sistema operativo y ERROR_SIN_REESTABLECER si ha habido algún error y no se sabe el estado del semáforo. Esta última macro se utiliza en las funciones que modifican varios semáforos en una misma llamada (DownMultiple_Semaforo y UpMultiple_Semaforo). Estas funciones llaman a Down_Semaforo y Up_Semaforo, respectivamente, por cada número de semáforo pasado como argumento. En caso de que alguna de estas llamadas devuelva un error, se intentará deshacer los cambios llamando a la otra función por cada número de semáforo previo al error y se devolverá ERROR. Pero, si al intentar deshacer los cambios, la otra función también devuelve un error, el estado del semáforo es desconocido y se devolverá ERROR_SIN_REESTABLECER.

Ejercicios 5: test de la biblioteca de semáforos

En el ejercicio cinco, se pedía realizar un test comprobando el correcto funcionamiento de la biblioteca de semáforos implementada en el apartado anterior. Para ello, hemos realizado dos programas. En el primero, ejercicio5.c, se comprueba su validez para semáforos binarios y en el segundo, ejercicio5b.c, se comprueba en el caso más general. En ambos, se llaman a todas las funciones de la biblioteca, comprobando que sus retornos son OK. En caso de que alguno no lo sea, se imprime un mensaje de error por pantalla y se termina el programa liberando los semáforos. En caso de correcto funcionamiento, se van imprimiendo el nombre de las funciones que han retornado correctamente.

Salida del test:

```
homerjr:~/workspace/soper/Práctica 3 $ ./ejercicio5
Crear
Crear con semaforos ya existentes
Inicializar
Down semaforo 0
Up semaforo 0
DownMultiple_Semaforo
UpMultiple_Semaforo
Borrar_Semaforo
Ejecución correcta
homerjr:~/workspace/soper/Práctica 3 $ ./ejercicio5b
Crear
Crear con semaforos ya existentes
Inicializar
DownMultiple_Semaforo
Down semaforo 1
Down semaforo 2
Down semaforo 2
Up semaforo 2
Up semaforo 2
Up semaforo 1
UpMultiple_Semaforo
Borrar_Semaforo
Ejecución correcta
```

Ejercicio 6: problema del productor-consumidor

En este ejercicio se pedía implementar el problema del productor-consumidor, en el que el productor escribe letras en una zona de memoria compartida, a la cual accede el consumidor para leer las letras e imprimirlas por pantalla.

Dado que el enunciado era muy general, hemos decidido implementarlo mediante dos procesos hijos. El padre crea la memoria compartida y tres semáforos que se inicializan a uno, tamaño de la memoria y a cero (respectivamente). El primero, es un semáforo binario que bloquea al proceso que quiere acceder a la memoria, en caso de que el otro proceso esté accediendo. El segundo indica el número de espacios libres donde el productor puede escribir. El tercero indica el número de caracteres que puede leer el consumidor. Posteriormente, el padre crea tanto el proceso productor como el proceso consumidor que se unen a la memoria compartida. Después, entra en un bucle hasta que escribe la última letra, se separa la memoria y termina su ejecución. En cada iteración, se realiza un down sobre los dos primeros semáforos para indicar que se va a reducir en uno el espacio libre y que se va a acceder a la memoria compartida. Una vez escrito el carácter, se realiza un up sobre el primero y el tercer semáforo, indicando que se deja de utilizar la memoria y para despertar al proceso consumidor para que pueda leer. Por su parte, el proceso consumidor realiza una secuencia de acciones similar sobre los semáforos, pero se realiza un down sobre el tercer semáforo antes de leer y un up sobre el segundo semáforo después para indicar si puede leer y despertar al proceso productor al final. Cuando lee la última letra del abecedario, se separa de la memoria y termina. Finalmente, el padre espera a ambos hijos y libera todos los recursos.

Otra decisión que hemos tomado ha sido la de limitar el tamaño del buffer a 5 para que se vea con mayor claridad la alternancia que se produce entre consumidor y productor.

A la derecha se adjunta una posible salida de la ejecución. Como se puede observar, productor y consumidor son ejecutados de manera aleatoria y ambos terminan cuando escriben o leen la letra z.

```
./ejercicio6
Produce a
Produce b
Produce c
Produce d
b consumido
b consumido
c consumido
d consumido
Produce e
Produce f
Produce g
Produce h
e consumido
f consumido
g consumido
Produce i
h consumido
Produce j
i consumido
Produce k
j consumido
Produce l
k consumido
Produce m
l consumido
Produce n
m consumido
Produce o
n consumido
Produce p
o consumido
Produce q
p consumido
Produce r
q consumido
Produce s
r consumido
Produce t
s consumido
Produce u
t consumido
Produce v
u consumido
Produce w
v consumido
Produce x
w consumido
Produce y
x consumido
Produce z
y consumido
z consumido
```