Final Laboratorio SO2122q2

Makefile (0.5 puntos)

Crea un **Makefile** que permita generar todos los programas del enunciado a la vez y cada uno de ellos por separado. Añade una regla (clean) para borrar todos los binarios y/o ficheros objeto, y dejar sólo los ficheros fuente. Los programas deben generarse si, y solo si, ha habido cambios en los ficheros fuente.

Control de errores (0.5 puntos)

Para todos los programas que se piden a continuación deben comprobarse los errores de TODAS las llamadas a sistema (excepto el write por pantalla), controlar los argumentos de entrada y definir la función Usage().

Ejercicio 1: Gestión procesos (3 puntos)

Haz un programa p1.c que reciba 2 parámetros: un número de procesos y un entero (que llamaremos semilla). El programa debe crear tantos procesos como indique el primer argumento. Los procesos se crearán de forma secuencial. Para cada nuevo proceso, se mutará a programa dummy (que os damos implementado) pasándole un entero como argumento. Ese entero será para el primero proceso creado la semilla recibida por p1 y para los otros el valor actual de semilla más el valor que el proceso anterior devuelva en el exit (de forma que la semilla se va incrementando). El programa p1 escribirá por la salida std un mensaje indicando el valor de semilla al final de la ejecución. Por ejemplo

```
[user]$ ./p1 2 1
El total es 7
[user]$ ./p1 3 1
El total es 15
[user]$ ./p1 2 2
El total es 11
```

El programa dummy tiene un retardo por lo que el tiempo de ejecución se va incrementando.

Ejercicio 2: Gestión procesos + signals (1,5 puntos)

Copia p1.c en p2.c. Modifica p2 de forma que limitemos el tiempo que p2 estará ejecutándose. Para ello, haremos las siguientes modificaciones. Añadiremos un tercer parámetro que será un número máximo de segundos. El programa debe utilizar el signal SIGALRM para controlar el tiempo de ejecución y limitarlo al indicado en el parámetro. Si pasan los X segundos indicados, el programa aún está ejecutándose, escribirá el mensaje “Tiempo límite” y terminará. Se valorará que que durante la configuración del signal los signals estén bloqueados. El programa no puede utilizar sleep para implementarlo, ha de hacerse con las llamadas trabajadas en el curso.

Por ejemplo, si ponemos 5 segundos de límite y usamos el comando time para comprobar el tiempo de ejecución veremos lo siguiente.

```
[user]$ time ./p2 3 1 5
 Limite tiempo

real    0m5.004s
user    0m0.001s
sys     0m0.010s
```

Si ponemos 50 segundos sí que dará tiempo a ejecutarse el programa

```
[user]$ time ./p2 3 1 50
El total es 15

real    0m11.016s
user    0m0.000s
sys     0m0.015s
```

Ejercicio 3: Entrada/Salida (1,5 puntos)

Copia p2.c en p3.c. Modifica p3.c de forma que los mensajes que en p2 iban a la salida std ahora vayan a un fichero de datos. Para ellos añadiremos un nuevo argumento que será un nombre de fichero. El programa p3 debe crear el fichero si no existe y si existe debe asegurarse que solo contiene le resultado de cada ejecución. El fichero ha de tener permisos de lectura y escritura para el propietario. El programa debe conservar las escrituras en la salida std, de forma que debéis hacer los cambios necesarios para que vuestro fichero corresponda con la salida std.

Por ejemplo:

```
[user]$ ./p3 4 1 50 salida4.1
[user]$ cat salida4.1
El total es 31
[user]$ ./p3 4 1 5 salida4.1
[user]$ cat salida4.1
 Limite tiempo
```

Ejercicio 4: Comunicación procesos (3 puntos)

Copia dummy,c en dummy2.c. Modifica dummy2.c para que antes del exit escriba el parámetro del exit por su salida std en formato interno de la máquina.

Copia p1.c en p4.c. Modifica p4.c para que los procesos creados originalmente muten a dummy2 en lugar de a dummy. Además crearemos un nuevo proceso adicional . El objectivo es que este nuevo proceso se comunique con una pipe sin nombre con los procesos que mutan a dummy, de forma que lea los números que estos escriben y los sume todos. Este nuevo proceso debe leer los números que habrán en la pipe y acumularlos a la semilla inicial . Cuando no haya más números escribirá un mensaje por la salida std indicando el total “Total de la pipe es x”. Si vuestro programa está bien, esta X debe coincidir con el mismo valor que el proceso incial también imprimirá (que viene de p1.c).

- Ten en cuenta que dummy2 escribe por su salida std.
- Piensa cuando debes crear este nuevo proceso, si antes o después de los procesos que mutarán a dummy2.

Por ejemplo:

```
[user]$ ./p4 3 2
El total es 23
Total de la pipe es 23
```

Final Laboratorio SO2122q2

Qué hay que hacer

- El Makefile
- Los códigos de los programas en C
- La función Usage() para cada programa

Qué se valora

- Que sigas las especificaciones del enunciado
- Que el uso de las llamadas al sistema sea el correcto
- Que se comprueben los errores de **todas** las llamadas al sistema
- Que el código sea claro y correctamente indentado
- Que el Makefile tenga bien definidas las dependencias y objetivos
- Que la función Usage() muestre por pantalla como debe invocarse correctamente el programa en el caso que los argumentos recibidos no sean los adecuados

Qué hay que entregar

Un único fichero tar.gz con el código de todos los programas y el Makefile,:

```
tar zcvf final.lab.tar.gz Makefile *.c
```