Final Laboratorio SO2223q1

Makefile (0,5 puntos)

Crea un **Makefile** para 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)

Todos los programas deben comprobar 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 (4 puntos)

Haz un programa, llamado "**p1.c"**, que reciba 1 o 2 parámetros. El primer parámetro puedes asumir que siempre es un número. Éste indica la cantidad de elementos que se escribirá en formato interno. Si la invocación del programa tiene un segundo parámetro de entrada, éste indica el tipo de elementos que escribirá. Este parámetro es un carácter con los posibles valores 'c' (los elementos serán chars) o 'i' (los elementos serán integers). En caso de no tener este segundo parámetro, el programa decide el formato de los elementos mediante la comprobación del nombre del ejecutable que se ha usado para la invocación. En caso de ser "./p1-char.exe" escribirá chars, mientras que si es "./p1-integer.exe" serán integers (**NOTA:** *para comparar strings mira en el man cómo usar la rutina "strcmp", ya utilizada en el laboratorio*). Si se ha invocado mediante otro nombre y sólo tiene un único parámetro de entrada, debe mostrar el mensaje de la función Usage y finalizar.

El comportamiento del programa debe ser el siguiente. Una vez comprobados los parámetros de entrada, realiza una espera pasiva hasta la recepción de un signal SIGUSR1 (no es necesario bloquear el signal al comienzo de la ejecución). Cuando lo reciba, muestra un mensaje por la salida estándar con el formato "El proceso <PID> comienza", donde "<PID>" es el PID del proceso. A continuación escribe, a través del canal (file descriptor) número 10, los elementos indicados mediante la invocación del programa (es decir, chars o integers). Serán valores consecutivos crecientes, es decir, desde cero hasta el valor que alcance en función de la cantidad de elementos escritos. Una vez completado, escribe el mensaje "El proceso <PID> ha terminado", por la salida estándar, y realiza la llamada a sistema para finalizar, utilizando como código de finalización el número de elementos escritos.

GUIA: *aconsejamos seguir los siguientes pasos para implementar el código: (1) implementar el caso de un único parámetro escribiendo chars; (2) añadir la opción de uno o dos parámetros escribiendo los dos tipos de datos; (3) añadir el resto de características del enunciado.*

NOTA: *para comprobar el correcto funcionamiento del código, puedes ejecutar la siguiente línea de código, donde se redirecciona el canal 10 para que apunte a un fichero llamado "datos.dat". Esta línea de comandos SOLO funciona bien con el intérprete de comandos "bash".* ***Ejecuta "bash" si estás en otro intérprete, como por ejemplo "tcsh".*** *Este redireccionamiento no lo necesitas en la ejecución normal, SOLO PARA HACER UN TEST. En esta ejecución se invoca al programa "./p1" para que escriba 5 chars (recuerda enviar un SIGUSR1 para que empiece a escribir):*

./p1 5 c 10> datos.dat

PREGUNTA: crea un fichero de texto, llamado "**respuestas.txt**" y contesta a la siguiente pregunta. ¿Qué línea/s de comandos has de ejecutar para conseguir tener un único ejecutable, pero que se pueden usar los nombres "p1-char.exe" y "p1-integer.exe" para invocarlo?

Final Laboratorio SO2223q1

Ejercicio 2 (5 puntos)

Implementa un programa llamado "**p2.c"** que espera parámetros de entrada en forma de duplas "X Y", donde "X" es un número e "Y" es un carácter ('c' o 'i'). El proceso padre creará una pipe con nombre, llamada "MYPIPE", que comunicará los procesos hijo con su padre. De tal forma que se crearán tantos procesos hijo como duplas haya en la invocación del programa. Cada uno de ellos mutará para ejecutar el código desarrollado en el ejercicio anterior, utilizando cualquiera de las dos formas de invocación indicadas.

El comportamiento del código es el siguiente. El proceso padre, una vez revisados los parámetros de entrada, crea los procesos hijo siguiendo un esquema secuencial. Cada uno ha de bloquear el signal SIGUSR1 antes de mutar para ejecutar el programa desarrollado en el ejercicio anterior. Además, su tabla de canales debe estar preparada para que los elementos que escribirá, por el canal 10, lo hará a la pipe. **NOTA:** *en caso que no hayas sido capaz de realizar el ejercicio anterior, te proporcionamos un ejecutable, llamado "p1-original.exe", correspondiente a la implementación correcta.*

Por su parte, el proceso padre después de crear cada proceso hijo realizará una espera activa durante 2 segundos. Después de este tiempo, envía un signal SIGUSR1 al proceso hijo y solicita memoria dinámica para un array de tantos elementos, y del tipo correspondiente, como los que leerá del proceso hijo que acaba de crear. Lee **los valores** (chars o integers) de la pipe y los guarda en el array. Cuando el proceso hijo finalice, si es voluntariamente, el proceso padre crea un fichero, con el nombre "salida-XXX.dat", donde "XXX" será el PID del proceso hijo, y con permisos de lectura y escritura para el usuario creador y para el grupo, pero ningún permiso para el resto de usuarios. Si ya existía, borrará previamente su contenido. En él escribirá, en formato interno, todos los valores del array mediante un único write y liberará la memoria que había solicitado para el array. Cuando el proceso padre haya terminado de realizar las tareas relacionadas con el último proceso, mostrará un mensaje por pantalla indicando el mensaje "Final del proceso principal" y terminará.

NOTA: *Puedes confirmar si el contenido de cualquiera de los ficheros ".dat" es correcto utilizando el comando "xxd". El comando muestra, en formato hexadecimal, el valor de cada byte* del fichero.

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
- El fichero respuestas.txt

Qué hay que entregar

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

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