Práctica

1

Sistemas Operativos

Repaso de utilización de la shell y del lenguaje C

Enunciado

En esta práctica deberás realizar dos programas en lenguaje “C” y trabajar con ellos con la Shell además de usar las herramientas de desarrollo para su creación y verificación.

Generador

Un programa en leguaje “C” (con el nombre “`generador`”) escribe en STDOUT 1.024 líneas que contienen un número aleatorio en el intervalo [-100.000, 100.000], y un retorno de carro (“\n”). El valor que determina el intervalo se recibe por parámetro (argc, argv), y si no se especifica nada se toma el valor 100.000. El programa debe inicializar el generador de números aleatorios para asegurar que en cada llamada el número obtenido es diferente.

Filtro

Un segundo programa en “C” denominado “`filtro`”, recibe como parámetro obligatorio por la línea de comandos (argc, argv) un valor entero (`LIMITE`), y a continuación lee todos los datos que recibe por `STDIN`; si el valor leído está en el intervalo `[-LIMITE, +LIMITE]` escribe el valor en `STDOUT` y en caso contrario, lo escribe en `STDERR`.

Procesa.sh

Construir un script (denominado “procesa.sh”) que compruebe el valor de la variable de entorno `NUM_ITERACIONES`. Si no está definida, tomará el valor por defecto 10. Llama al programa “generador” durante `NUM_ITERACIONES` veces, redirigiendo la salida de forma incremental al fichero “datos.1”, “datos.2” y así sucesivamente. Cuando acaba una ejecución, espera 1 segundo antes de llamar a la siguiente.

El script llama al programa “filtro” `NUM_ITERACIONES` veces, pasando como parámetro el valor 1.000 y utilizando el fichero correspondiente (“datos.1”, “datos.2”, …) por su entrada estándar. La salida de cada ejecución se va añadiendo a los ficheros “`salida.stdout`” y “`filtrado.stderr`”.

Por último, el script debe imprimir los diez valores más altos del fichero “`salida.stdout`”, así como el número de elementos filtrados en la etapa anterior.

Realización

1. Crea un directorio para esta práctica, denominado “`practica_1`”.
2. Crea “`generador.c`”, compila y depura el programa. Prueba a pasar diferentes valores y comprueba que los valores generados están en el rango especificado. Redirige la salida y crea un fichero de datos “`datos_generador.txt`”.
3. Crea “`filtro.c`”, compila y depura el programa. Utiliza el fichero de datos obtenido del apartado anterior para comprobar que el filtro funciona correctamente y escribe por la salida STDOUT y STDERR los valores adecuados.
4. Enlaza a través de una pipe los comandos generador y filtro. Por ejemplo:
   ```
   generador 200 | filtro 100 > salida.stdout 2> filtrado.stderr
   ```
   Verifica que los ficheros de datos
5. Usa el comando `tee` entre ambos comandos en el ejemplo anterior para ver la información intermedia y guárdalo en el fichero “`generador.stdout`”
6. Generar un fichero `make` para compilar de forma modular este proyecto, de tal forma que pueda compilar ambos programas.
7. Crea el script “`procesa.sh`” y dale permisos de ejecución (utiliza el octal 755 y justifica el uso de este valor). Prueba de forma independiente cada apartado del script antes de verificar que funciona correctamente por completo.

8. Cuando lo tengas debes entregar un único fichero de formato `tar` con lo que has realizado (explicación en comentarios, fuentes, datos, constantes, `make`, entrada, macro, …) a través de Moodle. Se te valorará la calidad del código, que todo funcione y la macro y `make` realizados, así como la documentación / comentarios.

Plazo

En la sesión cuarta se te pedirá que muestres su correcto funcionamiento y cualquier aspecto que el profesor crea oportuno. Deberás entregar lo realizado en el Moodle. La práctica es individual.

Ayudas

1. El canal 0 es `stdin`, el canal 1 es `stdout`, el canal de errores 2 es `stderr`. Los primeros se toman por defecto. Si queremos escribir por el de errores deberemos usar la instrucción:

```
fprintf(stderr, lo mismo que printf);
```

2. El `scanf` nos puede ayudar a comprobar que hemos leído lo adecuado. Devuelve un entero que corresponde al número de datos asignados correctamente, en nuestro caso 1. En otro caso tendremos un error. Se suele incluir en un `while` con <= 0.

3. La función `atoi` convierte un *string* en un entero. Toma un string de argumento y devuelve el entero convertido si el string tiene formato numérico.

```
entero = atoi("numero en string");
```

4. La generación de números aleatorios se puede hacer con llamadas a una librería del ANSI C o con llamadas al estándar POSIX. Si nos limitamos al primer caso (el otro es parecido) debemos:

   a. Usar las cabeceras `#include <stdlib.h>` e `#include <time.h>`.

   b. Inicializar una única vez el generador (semilla) de números aleatorios, por ejemplo, utilizando la fecha del sistema en el momento de hacerlo:

```
srand( time(NULL) );
```

   c. Llamar a la función `rand()` cada vez que necesites un número aleatorio. Esta función genera un número entre `0` y `RAND_MAX` (una constante ya definida en la cabecera). Ejemplos:

```
r = rand() % 100; // un entero entre 0 y 99
r = rand() % 10;   // un entero del 0 al 9
r = rand () % (N-M+1) + M;   // un entero entre M y N
```

5. Para ver cómo funciona el comando `tar` usa el manual. Ejemplos de uso:

   - Crea un tar con cuatro ficheros
     `tar -cf fichero.tar uno dos tres cuatro`
   - Lista el contenido de un tar de forma detallada (verbose)
     `tar -tvf fichero.tar`
   - Extrae en el directorio los ficheros que contiene el tar
     `tar -xf archive.tar`

6. En una macro, guion de Shell o Shell script, para ver si una variable está definida se puede seguir este patrón:

```
if [ -z $VAR ]; then
VAR=1000
fi

if [ -z $VAR ]; then VAR=1000 ; fi   #en una linea
```