Filtrar: filtra por contenido los archivos del directorio indicado.

Código de Apoyo

El código de apoyo del ejercicio puede descargarse de la aquí: filtrar.2015a.tgz

También se encuentra disponible en triqui en la ruta:
`~ssoo/pub/filtrar.2015a.tgz`

```
[usuario@triqui ??] cd
[usuario@triqui ~] cp ~ssoo/pub/filtrar.2015a.tgz .
```

El código de apoyo debe descomprimirse en la ruta `DATSI/so/filtrar.2015a`. Para ello se puede usar el siguiente mandato. Compruebe previamente que en su cuenta este directorio no existe o está vacío.

```
[usuario@triqui ??] cd
[usuario@triqui ~] tar xvfz filtrar.2015a.tgz
[usuario@triqui ~] cd DATSI/so/filtrar.2015a
[usuario@triqui ~/DATSI/so/filtrar.2015a]
```

Compilación

El código de apoyo incluye un *Makefile* que controla los detalles de compilación todos los fuentes de apoyo.

```
[usuario@maquina ~/DATSI/so/filtrar.2015a] make
gcc -Wall  -ldl  filtrar.c filtrar.h   -o filtrar
gcc -Wall -fPIC -shared libfiltra_alfa.c  -o libfiltra_alfa.so
gcc -Wall -fPIC -shared libfiltra_delay.c -o libfiltra_delay.so
gcc -Wall -fPIC -shared libfiltra_void.c  -o libfiltra_void.so
```

El código inicial, tal y como se proporciona, contiene algunos errores y *warnings* que deben ser corregidos para hacer que compile.
No debería ser difícil. **Esta será su primera tarea!**

Corrección

La puntuación que se obtendrá en este ejercicio será la alcanzada por medio de un corrector. El corrector se pasará sobre las prácticas entregadas.

Entrega

En cualquier momento se podrá entregar el código desarrollado, en el estado en el que se encuentre. La única entrega válida será la última. La entrega se podrá realizar directamente mediante el mandato:

```
[usuario@triqui ~/DATSI/so/filtrar.2015a/] make entrega
...
entrega.ssoo Versión 2004.3, Recolector de fuentes de practicas
entrega.ssoo Copyright (c) 1995..2004 Francisco Rosales (frosal@fi.upm.es)

entrega.ssoo
```

```
   El alumno "XXXXX"
   desea entregar la practica "filtrar.2015a".

entrega.ssoo
   Departamento: Arquitectura y Tecnología de Sistemas Informáticos
   Asignatura: Sistemas Operativos
   Encargado: Francisco Rosales <frosal@fi.upm.es>
   Titulo: Filtrar: Presencial de Diseño
   Directorio origen: ~/DATSI/so/filtrar.2015a
   Ficheros: filtrar.c filtrar.h ...
....
```

Mensajes Generados

Para su conveniencia, a continuación se muestran los formatos de los mensajes generados por este programa.

Mensajes de error

```
"Uso: %s directorio [filtro...]\n"
"Error al abrir el directorio '%s'\n"
"AVISO: No se puede stat el fichero '%s'!\n"
"AVISO: No se puede abrir el fichero '%s'!\n"
"Error al emitir el fichero '%s'\n"
"Error al leer el directorio '%s'\n"
"Error al crear el pipe\n"
"Error al crear proceso %d\n"
"Error al ejecutar el mandato '%s'\n"
"Error al ejecutar el filtro '%s'\n"
"Error al esperar proceso %d\n"
"%s: %d\n"
"%s: senyal %d\n"
"Error al abrir la biblioteca '%s'\n"
"Error al buscar el simbolo '%s' en '%s'\n"
"Error al intentar matar proceso %d\n"
"AVISO: La alarma ha saltado!\n"
"Error FILTRAR_TIMEOUT no es entero positivo: '%s'\n"
"AVISO: La alarma vencera tras %d segundos!\n"
```

Funcionamiento

```
filtrar directorio [filtro...]
```

El mandato a desarrollar (*filtrar*) tiene como objetivo aplicar los filtros indicados a la concatenación de los archivos contenidos en el directorio especificado. Es decir, su comportamiento sería similar a:

```
cat directorio/* | filtro1 | filtro2 | ... | filtroN
```

Pero con las siguientes diferencias:

- Los filtros (indicados como argumentos opcionales) podrán ser de dos tipos:
  - Biblioteca dinámica, si su nombre termina en ".so".
  - Mandato estándar sin argumentos, para cualquier otro nombre.
- Sólo se procesarán las entradas de `directorio` que no empiecen por punto y no sean a su vez directorios.
- Se implementará un control de tiempo máximo de ejecución.

A continuación se procede a describir el comportamiento esperado de este mandato, en el orden en que el

corrector de esta práctica irá pasando las pruebas.

Pruebas

En general, toda detección de error que realice el programa deberá emitir un mensaje por la salida de error estándar. Si el error impide continuar, se deberá concluir el proceso con un estado de terminación 1.

Compilación correcta

El código inicial proporcionado contiene algunos errores que impiden que compile.

- Prueba de que el código compila correctamente.

Chequeo de argumentos

Hay un sólo argumento obligatorio.

- Pruebas sobre el número correcto de argumentos.

Recorrido de directorio

- Prueba de que el directorio indicado se puede abrir y recorrer.
- Prueba de que se saltan las entradas que empiecen por punto.
- Prueba de que se saltan las entradas que sean directorio.
- Prueba de que se saltan las entradas que no se puedan leer.

Emitir el contenido de los archivos del directorio

El proceso principal, al recorrer el directorio y para cada archivo que abre, se debe emitir su contenido completo por la salida estándar.

El proceso principal deberá tolerar la terminación anticipada del primer filtro (su lector), del mismo modo que deberá permitir que este termine y filtre la información recibida.

- Prueba de que se emite el contenido de los archivos del directorio.

Conexión de un filtro en secuencia

Al principio no se realizarán pruebas con más de un filtro.

Tras el nombre del directorio podrá venir indicado por su nombre (sólo su nombre sin más argumentos) un filtro, lo que permitirá distinguir entre los dos tipos de filtro.

Todo filtro deberá filtrar la información presente a su entrada estándar y sacar el resultado por su salida estándar. Todo filtro deberá ser ejecutado como un proceso hijo del proceso principal.

Estos procesos deben quedar comunicados por tuberías, para formar una secuencia o *pipeline*. La entrada estándar del primer filtro vendrá de la salida estándar del proceso principal. La salida estándar del último filtro será la salida del programa.

NOTA: Para facilitar su comprensión se recomienda dibujar como quedaría la secuencia de procesos.

Filtro de tipo mandato estándar

El argumento correspondiente será el nombre del mandato a ejecutar, sin argumentos.

- Prueba de que se detecta fallo al intentar ejecutar el mandato.
- Prueba de que el mandato se ejecuta correctamente.
- Prueba de que el mandato filtra la entrada hacia su salida.

Estado de terminación de los filtros

Se debe esperar la terminación de cada uno de los procesos filtro, en el orden en que aparecen en la invocación. Se debe visualizar un mensaje con el nombre del filtro y su estado de terminación.

- Prueba de que se espera la terminación del filtro.
- Prueba de que se detecta y muestra su estado de terminación.
- Prueba de que se distinguen los procesos muertos por una señal.

Filtro de tipo biblioteca dinámica

El procedimiento de filtrado en el caso de los filtros de tipo **biblioteca dinámica** consistirá en localizar en la biblioteca el símbolo **`tratar`** y utilizarlo para procesar los datos leídos (en bloques de 4KiB) de la entrada estándar y escribir por la salida estándar los datos que superen el filtrado.

```
/*
 * Esta es la función de filtrado que debe estar
 * implementada en cada biblioteca de filtrado.
 */
    int tratar(char*buff_in, char*buff_out, int tam);
/*
 * Procesa <tam> caracteres de <buff_in> copiando
 * en <buff_out> aquellos que pasan el filtro.
 * Devuelve el número de caracteres que se han copiado.
 */
```

- Prueba de que se intenta abrir la biblioteca como tal.
- Prueba de que se intenta localizar el símbolo `tratar`.
- Prueba de que se ejecuta correctamente la biblioteca.

Implementación de bibliotecas de filtrado

Se debe implementar un filtro sencillo (**`libfiltra_alfa.c`**) que se proporciona vacío en el código de apoyo, y que debe dejar pasar sólo los caracteres **NO alfabéticos**. Para ello se precisará el uso de `isalpha()`.

- Prueba de que `libfiltra_alfa.so` filtra correctamente.

Control de la temporización

Al comenzar la ejecución del programa, si la variable de entorno **`FILTRAR_TIMEOUT`** existe se comprobará que contiene un número entero positivo. En caso afirmativo se preparará una temporización con ese número de segundos y se mostrará el mensaje **`AVISO: La alarma vencera tras x segundos!`**.

Si vence la temporización antes de que el programa termine, se mostrará el mensaje **`AVISO: La alarma ha saltado!`** y se procederá a mandar un `SIGKILL` a cada proceso filtro que siga activo. Para comprobar que un proceso sigue activo se le enviará la señal 0.

El proceso principal deberá terminar, no sin antes pasar a recoger y mostrar el estado de terminación de los filtros, distinguiendo los que murieron por señal.

- Prueba de que la variable es un entero positivo.
- Prueba de temporización que no llega a saltar.
- Prueba de temporización que si llega a saltar.
- Prueba de que se mata a los hijos.

Conexión de múltiples filtros en secuencia

Además, este programa debe ser capaz de filtrar a través de más de un filtro.

Tras el nombre del directorio podrán venir indicados por su nombre (sólo su nombre sin más argumentos) cualquier número de filtros. Los filtros se aplicarán en el orden en que se hayan indicado (de izquierda a derecha).

Todo filtro deberá ser ejecutado como un proceso hijo del proceso principal. El proceso principal deberá tolerar la terminación anticipada de cualquier proceso filtro.

Todos los filtros deben quedar comunicados por tuberías, para formar una secuencia o *pipeline*. La entrada estándar del primer filtro vendrá de la salida estándar del proceso principal. La salida estándar de cada filtro estará conectada con la entrada estándar del siguiente filtro. La salida estándar del último filtro será la salida del programa.

- Pruebas de ejecución correcta con varios filtros.
- Pruebas con errores de ejecución en los filtros.
    - Pruebas con filtros que no funcionan correctamente.
    - Pruebas con filtros que no existan.

NOTA: Para facilitar su comprensión se recomienda dibujar como quedaría la secuencia de procesos.

NOTA: Para facilitar la codificación se recomienda construir la secuencia de filtros arrancando los correspondientes procesos filtro de derecha a izquierda, esto es, del último al primero.

Ejemplos de uso

```
[usuario@triqui ~] mkdir WORK
[usuario@triqui ~] echo "Archivo de 1 linea o 10 palabras o 50 caracteres." > WORK/fichero
[usuario@triqui ~] ./filtrar WORK ./libfiltra_alfa.so
  1   10   50 .
[usuario@triqui ~] ./filtrar WORK
Archivo de 1 linea o 10 palabras o 50 caracteres.
[usuario@triqui ~] ./filtrar WORK rev
.seretcarac 05 o sarbalap 01 o aenil 1 ed ovihcrA
rev: 0
[usuario@triqui ~] ./filtrar WORK wc
      1      10      50
wc: 0
[usuario@triqui ~] ./filtrar WORK rev wc
      1      10      50
rev: 0
wc: 0
[usuario@triqui ~] ./filtrar WORK wc rev
05      01      1
wc: 0
rev: 0
[usuario@triqui ~]
```

Páginas de Manual a Consultar

- `alarm` (2)
- `close` (2)
- `closedir` (3)
- `dlclose` (3)
- `dlerror` (3)
- `dlopen` (3)
- `dlsym` (3)
- `dup` (2)
- `exec` (3)
- `exit` (3)
- `fork` (2)
- `getenv` (3)
- `isalpha` (3)
- `isdigit` (3)
- `kill` (2)
- `open` (2)
- `opendir` (3)
- `pipe` (2)
- `read` (2)
- `readdir` (3)
- `sigaction` (2)
- `stat` (2)
- `strtol` (3)
- `waitpid` (2)
- `write` (2)

docencia/asignaturas/so4/prv/practicas/disenyo/filtrar.2015a.txt · Última modificación: 2015/02/19 16:31 por frosaladm