EXAMEN DE PRÁCTICAS (CONVOCATORIA EXTRAORDINARIA)

Escribe un programa filter.c para linux que, dado un fichero de entrada (primer argumento), una expresión regular (segundo argumento) y un fichero de salida (tercer argumento), deje el fichero de salida todas las líneas del fichero que tienen encajan con la expresión regular, pero en mayúsculas. Si hay algún error o no se proporcionan los argumentos indicados, se debe terminar avisando y con el estatus adecuado.

No se puede ejecutar una shell ni usar funciones como system(3).

Por ejemplo:

$ cat f
patatin
epepe
patatona
adios
$ grep '^pata' f | tr a-z A-Z > f2
$ cat f2
PATATIN
PATATONA
$ ./filter f "pata" f3
$ cat f3
PATATIN
PATATONA
$./filter f pata
usage: filter file regex fileout
$ echo $?
1
$

EXAMEN ENTREGA DE PRÁCTICAS (CONVOCATORIA EXTRAORDINARIA)

Añade un comando built in cdup a tu shell.

Vas a entregar sólo la versión con la modificación y sólo el código fuente, un .tgz con todos los ficheros.

El comando cdup recibe un argumento obligatorio que tiene que ser un número positivo o 0. El argumento cambia al directorio padre tantas veces como dice el argumento (o lo deja igual si es cero).

Ejemplo:

$ pwd
/tmp/hola/otro/mas
$ cdup 2
$ pwd
/tmp/hola
$ cdup 0

$ pwd
/tmp/hola
$ cdup 1
$pwd
/tmp
$ cdup 17
$pwd
/
$ cdup
usage: cdup N

EXAMEN DE ENTREGA DE PRÁCTICAS (CONVOCATORIA ORDINARIA)

Añade un comando builtin slay a tu shell.

Vas a entregar sólo la versión con la modificación y sólo el código fuente, un .tgz con todos los ficheros.

El comando slay recibe un argumento obligatorio que tiene que ser un número diferente a 0. El argumento representa un pid.
El comando mandará la señal SIGKILL dicho pid.
Si hay cualquier problema con los argumentos, escribirá su uso por la salida de error y saldrá con error.
Si hay algún problema mandando la señal, escribirá la razón por la salida de error y saldrá con error.

Ejemplo:

$ slay
usage: slay pid
$ slay 0
usage: slay pid
$ slay hola
usage: slay pid

 # el proceso 32 es de root y no tengo permisos
$ slay 32
Operation not permitted
$ slay 2056

EXAMEN DE PRÁCTICAS (CONVOCATORIA ORDINARIA)

Escribe un programa llamado ifaces en C para linux que escriba por su salida el número de interfaces de red que hay configuradas en el sistema. No se permite ejecutar un shell ni usar funciones como system(3). Si hay algún problema ejecutando los programas externos o de otro tipo, el programa debe escribir una descripción del error como corresponda y salir con fallo.

Por ejemplo:

$ ifconfig | egrep '^[^\t]' | wc -l

4
$ ifaces
4
$

EJERCICIO 0: C

Escribe un programa sortstr.c en C para Linux que reciba cómo argumentos palabras (un número indeterminado mayor que 0) y escriba por su salida la lista de palabras ordenadas. El programa debe ignorar las palabras que no tengan ninguna vocal.

Para ordenar, se debe implementar el algoritmo insertion sort.

Ejemplo:

$ ./sortstr ffhfhhfhf hola adios pepe ffffff pepin pepe
adios
hola
pepe
pepe
pepin
$

EJERCICIO 1: C CHKVAR

Escribe un programa chkvar.c en C para Linux que reciba un número par de parámetros, obligatoriamente mayor o igual que dos. Cada par de parámetros será un nombre de variable de entorno y su contenido. Si el contenido coincide con el que hay en el entorno en esa variable de entorno saldrá con éxito. En caso contrario (la variable no existe o ese no es su contenido) saldrá con estado de error, mostrando un mensaje por la salida de error como el que se muestra a continuación:

$ echo $USER $HOME $LANG
paurea /home/paurea C
$ ./chkvar USER paurea && echo bien
bien
$ ./chkvar NOEXISTE paurea && echo bien
error: NO EXISTE != paurea
$ ./chkvar USER paurea HOME /home/paurea && echo bien
bien
$ ./chkvar USER paurea HOME /home/pepe LANG es LANG C && echo bien
error: HOME != /home/pepe, LANG != es
$ ./chkvar
usage: ch varname varcontent [varname varcontent] …
$ ./chkvar USER
usage: ch varname varcontent [varname varcontent] …

EJERCICIO 2: C EXECARGS

Escribe un programa en C llamado execargs.c que ejecute todos los comandos que le pasen como argumentos. El primer argumento debe ser el número de segundos que hay que esperar entre la finalización de uno y la ejecución del siguiente. El resto de argumentos que se le pasan al programa serán rutas de ejecutables, opcionalmente con argumentos para ese comando si la string contiene espacios. No se puede ejecutar ejecutar una shell ni utilizar system(3).

Se considerará error si el programa no recibe al menos dos argumentos o el primero no es un entero.

En caso de error,
se debe avisar como corresponda y acabar la ejecución (no se deben ejecutar los programas restantes).

Para esperar un número de segundos, convertir de string a entero y partir una string en trozos usando un separador, se recomienda la lectura de:

man 3 sleep
man 3 atoi
man 3 strtoi
man 3 strtok_r

Un ejemplo de ejecución es el siguiente:

$ ./execargs 2 '/bin/echo hola' 'bin/echo adios'
hola
adios
$ ./execargs
usage execargs secs command [command …]
$ ls
execargs
execargs.c
execargs.o
$ ./execargs 1 /bin/ls '/bin/echo ya está'
execargs
execargs.c
execargs.o
ya está
$ ./execargs 1 /xx/ls '/bin/echo ya está'
/xx/ls: No such file or directory
error: /xx/ls
$ ./execargs 1 '/bin/echo hola' '/basdfo ya esta'
/basdfo: No such file or directory
error: /basdfo ya esta
$./execargs 1 '/usr/bin/ls /noexiste' ' /bin/echo ya esta'
/usr/bin/ls: cannot access '/noexiste' : No such file or directory
error: /usr/bin/ls /noexiste

EJERCICIO 3: GREPMATRIX

Escribe un programa en C para GNU/LINUX llamado grepmatrix.c que reciba como argumentos los siguientes datos:

- Una lista de palabras a buscar en ficheros.
- Una lista de rutas de ficheros en los que buscar las palabras.

La separación entre estas dos listas se delimitará con el modificador -f. Al menos debe darse una palabra y un fichero. La aparición de más de un modificador -f (o de cualquier otro que no sea -f) resultará en un error.

La salida del programa debe indicar en una matriz qué palabras aparecen en cada fichero, siguiendo el formato exacto de los ejemplos que se pondrán a continuación. Se indicará con “x” cuando aparece la palabra en el fichero, y con ‘o’ cuando no aparece. Observa que las columnas de la matriz están separadas por un único tabulador.

Para ver si un fichero contiene una palabra, el programa debe ejecutar fgrep(1). Se deben proporcionar los modificadores necesarios para que fgrep no escriba nada por su salida (ni datos ni errores).

Si hay error buscando alguna palabra en un fichero, el programa no debe escribir la matriz por su salida, debe escribir un error indicando qué fichero lo ha provocado y terminar la ejecución, pero sin dejar ningún proceso huérfano.

Todas las búsquedas deben ser concurrentes.
Ejemplos:

$ ./grepmatrix
usage: grepmatrix word [words …] -f file [file …]
$ ./grepmatrix hola -f
usage: grepmatrix word [words …] -f file [file …]
$ ./grepmatrix -f f1
usage: grepmatrix word [words …] -f file [file …]
$ echo hola adios > f1
$ echo adios > f2
$ ./grepmatrix hola adios -f f1 f2
“hola” “adios”
x x f1
o x f2
$ echo uno dos > f3
$ ./grepmatrix uno dos tres hola adios -f f1 f2 f3
“uno” “dos” “tres” “hola” “adios”
o o o x x f1
o o o o x f2
x x o o o f3
$ rm f2
$ ./grepmatrix uno dos tres hola adios -f f1 f2 f3
error processing file f2
$

EJERCICIO 4: COPYBYTES

Escribe un programa en C llamado copybytes.c que admita dos argumentos obligatorios con dos rutas: un fichero origen y un fichero destino. Opcionalmente se puede dar un tercer argumento con un número entero positivo. Si se pasan sólo dos argumentos, el programa debe copiar todo el fichero origen al fichero destino. Si se pasa un tercer argumento, sólo debe copiar los bytes indicados en el mismo.

Si el fichero destino existe, se debe truncar. En otro caso, se debe crear.

Si la ruta de origen es '-', se deberá leer de la entrada estándar. Si la ruta destino es '-', se deberá escribir en la salida estándar.

Se recomienda probar el programa con ficheros grandes (p. ej. varios MB).

Un ejemplo de ejecución:

$ echo hola adios > /tmp/a
$./copybytes /tmp/a /tmp/b 2
$ cat /tmp/b
ho$ ./copybytes /tmp/a /tmp/b
$ cat /tmp/b
hola adios
$ ./copybytes /tmp/a -
hola adios
$ ./copybytes - - < /tmp/a
hola adios
$

EJERCICIO 5: ZCOUNT

Escriba un programa en C para linux zcount.c cuente el número de bytes a cero en cada uno de los ficheros de un directorio y escriba en el mismo directorio un fichero z.txt con una linea por fichero. Cada línea contendrá el número de bytes a cero, un tabulador y el nombre del fichero. El programa debe recibir un solo parámetro que será el directorio a procesar. Si el fichero z.txt existe, debe ignorarlo para la cuenta, truncar y sobreescribirlo.

Ejemplo:

$ mkdir /tmp/a
$ echo aaa > /tmp/a/nada
$ dd if=/dev/zero of=/tmp/a/megas bs=10M count=1
1+0 records in
1+0 records out
10485760 bytes (10 MB, 10 MiB) copied, 0,0169875, 617 MB/s
$ dd if=/dev/zero of=/tmp/a /diez bs=10 count=1
1+0 records in
1+0 records out
10 bytes copied, 0,000119917 s, 83,4 kB/s

$ echo bbbbbb >> /tmp/a/diez
$ ls /tmp/a
diez megas nada
$ zcount /tmp/a
$ ls /tmp/a
diez megas nada z.txt
$ cat /tmp/z.txt
10 diez
10485760 megas
0 nada

Para leer, se debe usar la llamada al sistema read. Para este ejercicio no está permitido usar las funciones con buffering de stdio (fread,etc).

EJERCICIO 6: SOURCEFILES

Escribe un programa en C llamado sourcefiles.c que lea líneas de su entrada estándar con paths. Por cada path, el programa debe escribir a continuación el número de ficheros fuente que ha encontrado en ese path (en ese directorio y todos los subdirectorios que contiene, esto es, lo debe hacer recursivamente). Por cada path debe escribir:

- El path
- Número de ficheros .c encontrados en ese path.
- Número de ficheros .h encontrados en ese path.
- Suma del tamaño en bytes de todos los ficheros .c y .h que ha encontrado en ese path.

Si procesando algún directorio se encuentran errores, se debe notificar como corresponda, pero se deben seguir procesando líneas hasta el final de la entrada. Si hay algún problema, al final el programa deberá salir con estatus de fallo.

La salida se tiene que ajustar a la del ejemplo:

$ cat paths
/home/pepe
/home/manuel
$ ./sourcefiles < paths
/home/pepe 12 2 33124
/home/manuel 123 14 4234254
$

EJERCICIO 7: PIPELINE

Escribe un programa llamado pipeline.c en C para linux que reciba tres parámetros. Cada parámetro tiene que ser una string con un comando y sus argumentos. El programa debe esperar a todos los procesos y acabar con el estado de salida del último comando del pipeline.

Es necesario usar la llamada execv para ejecutar. Para tokenizar una string, se recomienda el uso de:

man 3 strtok_r

Los ejecutables de los comandos se deben buscar en los directorios /bin y /usr/bin (en ese orden)

El estado de salida debe ser el del último comando del pipeline.

Por ejemplo:

$ ls -l / | grep u | wc -l
4
$ ./pipeline 'ls -l/' 'grep u' 'wc -l'
4
$./pipeline 'echo hola' ' car 'tr a-z A-Z'
HOLA
$

EJERCICIO 8: TXTSHA2

Escriba en C para Linux un programa llamado txtsha2.c cuyo propósito es crear un resumen hash SHA 256 del resultado de la concatenación de todos los ficheros regulares acabados en .txt de un directorio (ojo, no tienen por qué contener texto). No debe leer subdirectorio, sólo debe considerar ficheros inmediatamente situados en el directorio.

El orden de la concatenación debe ser el orden en el que se encuentran en el directorio (no hay que ordenar los ficheros de ninguna forma). El programa sólo admiteun argumento para indicar el directorio sobre el que se quiera trabajar. SI no se le pasa ningún argumento, debe actuar sobre el directorio de trabajo actual.

El resumen SHA-256 debe calcularse ejecutando el comando sha256sum(1), que debe ejecutarse sin ningún argumento. Este comando lee su entrada estándar hasta que se acaba, para luego calcular el resumen SHA-256 de los datos leídos y escribirlo por su salida estándar.

La salida del programa tiene que ser, simplemente, los 32 bytes expresados en hexadecimal del resumen SHA-256 generado. No se puede escribir nada más, ni antes ni después del SHA-256. En particular, los datos extra que escribe el comando sha1sum a su salida (espacios, guión…), no deben aparecer a la salida del programa.

No se permite la ejecución de un shell ni el uso de la llamada system(), ni la creación de ficheros temporales, ni el uso de otros comandos externos aparte de sha256sum.

A continuación, se muestra un ejemplo:

$ ./txtsha2 /tmp/d
3ee41b364227bcea0b8ee704d718a27da932062cd23fdd851dbf54c8c6181f13
$

EJERCICIO 9: STACKTHREAD

Implementa una pila genérica (stack.c, stack.h y el programa de prueba main.c) con un array que permita meter punteros a cualquier cosa (void *)

Tendrá operaciones públicas para:

Stack, con operaciones públicas para:

- crear una pila vacía con un array de un tamaño (usará realloc si tiene que crecer)
- preguntar si la pila está vacía
- **push** y **pop** de (void *) (si la pila está vacía, pop devuelve NULL)
- ver cuántos elementos hay en la pila
- liberar la pila y recursos asociados

La pila tiene que ser **thread-safe**.

Se debe usr la primitiva **mutex** de la librería pthreads. Múltiples hilos de pthreads deben ser capaces de usar una misma pila de forma concurrente. Cada pila debe usar su propio mutex para asegurar su acceso concurrente (esto es, no se debe usar un mutex global para todas las pilas que se creen usando la biblioteca).

Se debe proporcionar un programa de prueba implementado en un fichero main.c. El programa de prueba debe:

- Tener una estructura **Valor**, con campos **v** (un entero) e **id** (un entero).
- Crear una pila vacía.
- Crear 100 threads. A cada thread se le debe pasar como argumento un entero, que usará como etiqueta (id) en sus valores.
- Cada uno de los threads insertará en la pila 100 puntos con valores correlativos crecientes. A continuación extraerá 40 valores. Cada elemento que extraiga que no sea suyo, lo contará e imprimirá al final el contador por la salida.
- Al acabar los threads, main comprobará que hay 60*100 elementos en la pila y extraerá los valores, comprobando que los valores con el mismo id salen en orden decreciente.

Se debe entregar un único fichero comprimido llamado **stackthreads.tgz** que contenga los tres ficheros **main.c stack.c** y **stack.h.**