Prácticas de Ampliación de Sistemas Operativos • Tarea Semana 4

En esta tarea se pide implementar dos herramientas de forma separada. Deberéis demostrar que sabéis trabajar con ficheros usando *buffers* para evitar hacer llamadas al sistema para leer o escribir un único byte del fichero, ya que una cantidad excesiva de llamadas al sistema enlentecerá nuestro programa. Estas dos utilidades se usarán junto con la desarrollada en la semana 3 (`merge_files`), para ejecutarlas unidas por tuberías en un programa (`merge_mystrings_split`). **En todos los programas se deberán tratar las lecturas y escrituras parciales adecuadamente, y evitar a toda costa las lecturas o escrituras byte a byte**.

Descripción de la primera herramienta: `mystrings`

Escribe un programa llamado `mystrings` que lea un flujo de bytes por la entrada estándar y saque por la salida estándar toda secuencia de caracteres imprimibles incluyendo el espacio ( ), el tabulador (`\t`) y el retorno de línea (`\n`) de una longitud mayor o igual que un cierto valor configurable, que vaya seguida de un carácter no imprimible. Esto nos permite extraer cadenas imprimibles de un fichero binario. **Nota**: no confundáis la representación de un carácter especial como es `\t`, que es el carácter tabulador (código ASCII 9), con los caracteres `\` y `t` por separado.

El programa tiene que aceptar las siguientes opciones:

```
./mystrings [-t BUFSIZE] [-n MINLENGTH]
```

donde `MINLENGTH` tomará el valor 4 por defecto y es el número mínimo de caracteres imprimibles necesarios para que una cadena sea sacada por la salida estándar; su valor está comprendido entre 1 y 255. `BUFSIZE`, por su parte tendrá el valor 1024 por defecto, pero puede ser cualquier valor positivo mayor o igual que MINLENGTH y menor o igual que 1048576.

El proceso tiene que utilizar como *buffer* de lectura una zona de memoria del tamaño indicado por la opción `BUFSIZE`, y las escrituras parciales deberán tratarse correctamente, tal y como se explicó en las sesiones de prácticas.

Para determinar si un carácter es válido para nuestra cadena, podemos ayudarnos de la siguiente función de `ctype.h`:

- `isprint()`: indica si el carácter es imprimible o un espacio.

Por supuesto, se deberán comprobar los argumentos y sacar los errores correspondientes. La herramienta `genera_bytes.py` es un programa en Python que se os proporciona junto con el enunciado.

Importante: Para simplificar la implementación de las escrituras, se permite que el buffer de escritura no esté completamente lleno en cada llamada a `write()`. No obstante, se valorará positivamente aquella implementación que no haga uso de esta simplificación.

Ejemplos de ejecuciones serían las siguientes:

```
$ ./mystrings -h
Uso: ./mystrings [-t BUFSIZE] [-n MINLENGTH]
Lee de la entrada estándar el flujo de bytes recibido y escribe en la
salida estándar
las cadenas compuestas por caracteres imprimibles incluyendo espacios,
tabuladores y
saltos de línea, que tengan una longitud mayor o igual a un tamaño dado.
-t BUFSIZE     Tamaño de buffer donde MINLENGTH <= BUFSIZE <= 1MB (por
defecto 1024).
-n MINLENGTH   Longitud mínima de la cadena. Mayor que 0 y menor que 256
(por defecto 4).

$ ./mystrings -n
./mystrings: option requires an argument -- 'n'
Uso: ./mystrings [-t BUFSIZE] [-n MINLENGTH]
Lee de la entrada estándar el flujo de bytes recibido y escribe en la
salida estándar
las cadenas compuestas por caracteres imprimibles incluyendo espacios,
tabuladores y
saltos de línea, que tengan una longitud mayor o igual a un tamaño dado.
-t BUFSIZE     Tamaño de buffer donde MINLENGTH <= BUFSIZE <= 1MB (por
defecto 1024).
-n MINLENGTH   Longitud mínima de la cadena. Mayor que 0 y menor que 256
(por defecto 4).

$ ./mystrings -n 0
La longitud mínima de cadena tiene que ser mayor que 0 y menor de 256.
Uso: ./mystrings [-t BUFSIZE] [-n MINLENGTH]
Lee de la entrada estándar el flujo de bytes recibido y escribe en la
salida estándar
las cadenas compuestas por caracteres imprimibles incluyendo espacios,
tabuladores y
saltos de línea, que tengan una longitud mayor o igual a un tamaño dado.
-t BUFSIZE     Tamaño de buffer donde MINLENGTH <= BUFSIZE <= 1MB (por
defecto 1024).
-n MINLENGTH   Longitud mínima de la cadena. Mayor que 0 y menor que 256
(por defecto 4).

$ echo "abcd" | ./mystrings -n 10
$ echo "abcd" | ./mystrings -n 5
abcd
$ echo -e "abcd\nefg" | ./mystrings -n 3
abcd
efg
$ echo -e "\b\bABAB\b\b" | ./mystrings -t 4 -n 4 | hexdump -C
00000000  41 42 41 42                                       |ABAB|
00000004
$ echo -e "\b\bAABBCCDD\b\b" | ./mystrings -t 6 -n 4 | hexdump -C
00000000  41 41 42 42 43 43 44 44                           |AABBCCDD|
00000008
$ echo -e "\b\bAABB\bCCDD\b" | ./mystrings -t 6 -n 4 | hexdump -C
00000000  41 41 42 42 43 43 44 44                           |AABBCCDD|
```

```
00000008
$ echo -e "ABCDEFGHI\bJKLMNOPQRSTUVWXYZ" | ./mystrings -n 8
ABCDEFGHIJKLMNOPQRSTUVWXYZ
$ echo -e "ABCDEFGHI\bJKLMNOPQRSTUVWXYZ" | ./mystrings -n 10
JKLMNOPQRSTUVWXYZ
$ python3 genera_bytes.py 20000 | ./mystrings -n 10
k#oX6Da @fI
$ python3 genera_bytes.py 20000 | ./mystrings -n 10 | hexdump -C
00000000  6b 23 6f 58 36 44 61 20  40 66 49                 |k#oX6Da @fI|
0000000b
```

Descripción de la segunda herramienta: `split_files`

Escribe un programa llamado `split_files` que acepte las siguientes opciones:

```
./split_files [-t BUFSIZE] FILEOUT1 [FILEOUT2 ... FILEOUTn]
```

Por ejemplo:

```
$ ./split_files -t 2048 salida1 salida2
```

El programa deberá leer la entrada estándar e ir escribiendo en los ficheros de salida los bytes de entrada, de manera que el primer byte vaya al primer fichero, el segundo byte al segundo fichero, y así hasta el n-ésimo, repitiendo este proceso cíclicamente hasta que se alcanza el final de la entrada.

Si los ficheros de salida no existen, deberán crearse, y si ya existen, se truncarán al abrirlos. Admitirá un máximo de 16 ficheros de salida.

Para conseguir un buen rendimiento, hay que utilizar buffers para reducir el número de llamadas a `read()` y `write()`. Las lecturas y escrituras parciales deben tratarse correctamente, tal y como se explicó en las sesiones de prácticas.

Aquí vemos unos ejemplos de ejecución. El `$` al final de una línea significa que no hay un retorno de carro al final de la salida de la orden y el *prompt* del sistema sale a continuación, pero en vuestro caso particular sería el *prompt* que tengáis activo. Esto sucede con el primer ejemplo que usa `echo`, que no añade ningún retorno de línea al usar la opción `-n`, y se puede ver cómo el segundo `cat` se escribe a continuación. Además vamos a usar el programa `genera_bytes.py` para generar valores binarios y así comprobar que el programa también funciona con dichos valores:

```
$ ./split_files -h
Uso: ./split_files [-t BUFSIZE] FILEOUT1 [FILEOUT2 ... FILEOUTn]
Divide en ficheros el flujo de bytes que recibe por la entrada estandar
El numero maximo de de ficheros de salida es 16.
-t BUFSIZE  Tamaño de buffer donde 1 <= BUFSIZE <= 1MB (por defecto 1024).

$ ./split_files
```

```
Error: No hay ficheros de salida.
Uso: ./split_files [-t BUFSIZE] FILEOUT1 [FILEOUT2 ... FILEOUTn]
Divide en ficheros el flujo de bytes que recibe por la entrada estandar
El numero maximo de de ficheros de salida es 16.
-t BUFSIZE  Tamaño de buffer donde 1 <= BUFSIZE <= 1MB (por defecto 1024).

$ ./split_files f1 f2 f3 f4 f5 f6 f7 f8 f9 f10 f11 f12 f13 f14 f15 f16 f17
Error: Demasiados ficheros de salida.
Uso: ./split_files [-t BUFSIZE] FILEOUT1 [FILEOUT2 ... FILEOUTn]
Divide en ficheros el flujo de bytes que recibe por la entrada estandar
El numero maximo de de ficheros de salida es 16.
-t BUFSIZE  Tamaño de buffer donde 1 <= BUFSIZE <= 1MB (por defecto 1024).

$ ./split_files -t 0
Error: Tamaño de buffer incorrecto.
Uso: ./split_files [-t BUFSIZE] FILEOUT1 [FILEOUT2 ... FILEOUTn]
Divide en ficheros el flujo de bytes que recibe por la entrada estandar
El numero maximo de de ficheros de salida es 16.
-t BUFSIZE  Tamaño de buffer donde 1 <= BUFSIZE <= 1MB (por defecto 1024).

$ echo "texto" > f1; chmod -w f1; ./split_files f1 f2
Error: No se puede abrir/crear 'f1': Permission denied

$ echo -n "ABCDEFGHIJKLMNOPQRSTUVWXYZ" | ./split_files -t 1 s1 s2
$ wc -c s1
13 s1
$ wc -c s2
13 s2
$ cat s1
ACEGIKMOQSUWY$ cat s2
BDFHJLNPRTVXZ$

$ python3 ./genera_bytes.py 20 | ./split_files -t 8 s1 s2
$ wc -c s1
10 s1
$ wc -c s2
10 s2
$ hexdump -C s1
00000000  39 8c 72 34 d8 0f 6f 0d  d6 e5                    |9.r4..o...|
0000000a
$ hexdump -C s2
00000000  0c 7d 47 2c 10 2f 77 65  70 8e                    |.}G,./wep.|
0000000a
```

Ejecución de las tres utilidades

Finalmente, vamos a hacer un pequeño programa que ejecute las tres utilidades conectadas por tuberías. Un ejemplo de línea de órdenes que ejecutaríamos sería la siguiente:

```
$ ./merge_mystrings_split -t 1024 -n 8 -i fi1,fi2,fi3 fo1 fo2
```

que debe ejecutar internamente la línea siguiente:

```
$ ./merge_files -t 1024 f1 f2 f3 | ./mystrings -t 1024 -n 8 | ./split_files
-t 1024 fo1 fo2
```

El programa deberá crear tres procesos hijos, cada uno de los cuales ejecutará (con alguna variante de `exec()`) una de las utilidades externas, es decir, que cada utilidad estará compilada por separado y estos procesos hijos usarán los respectivos ejecutables. Estos procesos estarán conectados entre sí por tuberías, de manera que cada uno pueda proporcionar al siguiente los datos que genere. El proceso padre esperará la terminación de los hijos sin un orden determinado.

Como podéis ver, habrá que tomar los parámetros que se le pasan a esta nueva orden y, a partir de ellos, configurar los parámetros que recibirá cada una de las tres órdenes que se ejecutarán conectadas por tuberías. En el caso concreto del argumento `-i`, la lista de ficheros de entrada está separada por comas, por lo que habrá que extraer los nombres de los ficheros de esa cadena.

Una forma sencilla de extraer estos nombres de fichero es usando la función `strtok_r()`, que podéis consultar en la ayuda de `man`. Esta función toma en su primera llamada una cadena a descomponer, otra cadena con el separador o separadores, y un tercer parámetro con una variable donde irá guardando por dónde se quedó analizando la cadena original. En las siguientes llamadas, se debe pasar `NULL` como cadena a descomponer. Cada llamada devolverá un puntero a carácter con la dirección de comienzo de cada cadena. Hay que tener mucho cuidado con esta función, porque lo que hace es modificar la cadena original, buscando el separador y sustituyéndolo por un carácter `\0`, que indica final de cadena. Es por esto que podemos usar los punteros que nos devuelve directamente como cadenas, pero si queremos no perder la cadena original, tendríamos que trabajar con una copia. Normalmente, al analizar los argumentos no necesitamos conservarlos, así que no nos preocupa que se altere la cadena.

Un ejemplo de código que extraería los elementos de una cadena separada por comas sería el siguiente:

```c
#define _POSIX_C_SOURCE 200809L
#include <stdio.h>
#include <string.h>

int main()
{
   char cadena[] = "file1,file2,file3";
   char *ptrToken;
   char *saveptr;

   ptrToken = strtok_r(cadena, ",", &saveptr);
   while (ptrToken != NULL)
   {
      printf("%s\n", ptrToken);
      ptrToken = strtok_r(NULL, ",", &saveptr);
   }
   return 0;
}
```

Si visualizáramos los bytes que componen ahora la cadena, podríamos comprobar que donde antes había comas, ahora hay ceros indicando fin de una subcadena.

Ejemplos de ejecución

```
$ echo "ABCDEFGHIJKLMNOPQRSTUVWXYZ" > fi1
$ echo "abcdefghijklmnopqrstuvwxyz" > fi2
$ echo "123456789012345678901234567890" > fi3

$ ./merge_mystrings_split -h
Uso: ./merge_mystrings_split [-t BUFSIZE] [-n MIN_LENGTH] -i
FILEIN1[,FILEIN2,...,FILEINn] FILEOUT1 [FILEOUT2 ...]
-t BUFSIZE     Tamaño de buffer donde 1 <= BUFSIZE <= 1MB (por defecto
1024).
-n MINLENGTH   Longitud mínima de la cadena. Mayor que 0 y menor que 256
(por defecto 4).
-i             Lista de ficheros de entrada separados por comas.

$ ./merge_mystrings_split
Error: Deben proporcionarse ficheros de entrada con la opción -i
Uso: ./merge_mystrings_split [-t BUFSIZE] [-n MIN_LENGTH] -i
FILEIN1[,FILEIN2,...,FILEINn] FILEOUT1 [FILEOUT2 ...]
-t BUFSIZE     Tamaño de buffer donde 1 <= BUFSIZE <= 1MB (por defecto
1024).
-n MINLENGTH   Longitud mínima de la cadena. Mayor que 0 y menor que 256
(por defecto 4).
-i             Lista de ficheros de entrada separados por comas.

$ ./merge_mystrings_split fo1 fo2
Error: Deben proporcionarse ficheros de entrada con la opción -i
Uso: ./merge_mystrings_split [-t BUFSIZE] [-n MIN_LENGTH] -i
FILEIN1[,FILEIN2,...,FILEINn] FILEOUT1 [FILEOUT2 ...]
-t BUFSIZE     Tamaño de buffer donde 1 <= BUFSIZE <= 1MB (por defecto
1024).
-n MINLENGTH   Longitud mínima de la cadena. Mayor que 0 y menor que 256
(por defecto 4).
-i             Lista de ficheros de entrada separados por comas.

$ ./merge_mystrings_split -i
./merge_mystrings_split: option requires an argument -- 'i'
Uso: ./merge_mystrings_split [-t BUFSIZE] [-n MIN_LENGTH] -i
FILEIN1[,FILEIN2,...,FILEINn] FILEOUT1 [FILEOUT2 ...]
-t BUFSIZE     Tamaño de buffer donde 1 <= BUFSIZE <= 1MB (por defecto
1024).
-n MINLENGTH   Longitud mínima de la cadena. Mayor que 0 y menor que 256
(por defecto 4).
-i             Lista de ficheros de entrada separados por comas.

$ ./merge_mystrings_split -i fi1,fi2
Error: Debe proporcionarse la lista de ficheros de salida
Uso: ./merge_mystrings_split [-t BUFSIZE] [-n MIN_LENGTH] -i
FILEIN1[,FILEIN2,...,FILEINn] FILEOUT1 [FILEOUT2 ...]
```

```
-t BUFSIZE     Tamaño de buffer donde 1 <= BUFSIZE <= 1MB (por defecto
1024).
-n MINLENGTH   Longitud mínima de la cadena. Mayor que 0 y menor que 256
(por defecto 4).
-i             Lista de ficheros de entrada separados por comas.

$ ./merge_mystrings_split -i f1noexiste,fi2 fo1 fo2
Error: El fichero f1noexiste no existe o no tiene permisos de lectura

$ ./merge_mystrings_split -t 0 -i fi1,fi2 fo1 fo2
Error: El tamaño de buffer debe ser mayor que 0 y menor que 1048576
Uso: ./merge_mystrings_split [-t BUFSIZE] [-n MIN_LENGTH] -i
FILEIN1[,FILEIN2,...,FILEINn] FILEOUT1 [FILEOUT2 ...]
-t BUFSIZE     Tamaño de buffer donde 1 <= BUFSIZE <= 1MB (por defecto
1024).
-n MINLENGTH   Longitud mínima de la cadena. Mayor que 0 y menor que 256
(por defecto 4).
-i             Lista de ficheros de entrada separados por comas.

$ ./merge_mystrings_split -t -i fi1,fi2 fo1 fo2
Error: El tamaño de buffer debe ser mayor que 0 y menor que 1048576
Uso: ./merge_mystrings_split [-t BUFSIZE] [-n MIN_LENGTH] -i
FILEIN1[,FILEIN2,...,FILEINn] FILEOUT1 [FILEOUT2 ...]
-t BUFSIZE     Tamaño de buffer donde 1 <= BUFSIZE <= 1MB (por defecto
1024).
-n MINLENGTH   Longitud mínima de la cadena. Mayor que 0 y menor que 256
(por defecto 4).
-i             Lista de ficheros de entrada separados por comas.

$ ./merge_mystrings_split -t 2000000 -i fi1,fi2 fo1 fo2
Error: El tamaño de buffer debe ser mayor que 0 y menor que 1048576
Uso: ./merge_mystrings_split [-t BUFSIZE] [-n MIN_LENGTH] -i
FILEIN1[,FILEIN2,...,FILEINn] FILEOUT1 [FILEOUT2 ...]
-t BUFSIZE     Tamaño de buffer donde 1 <= BUFSIZE <= 1MB (por defecto
1024).
-n MINLENGTH   Longitud mínima de la cadena. Mayor que 0 y menor que 256
(por defecto 4).
-i             Lista de ficheros de entrada separados por comas.

$ ./merge_mystrings_split -n 0 -i fi1,fi2 fo1 fo2
Error: La longitud mínima de cadena debe ser mayor que 0 y menor de 256
Uso: ./merge_mystrings_split [-t BUFSIZE] [-n MIN_LENGTH] -i
FILEIN1[,FILEIN2,...,FILEINn] FILEOUT1 [FILEOUT2 ...]
-t BUFSIZE     Tamaño de buffer donde 1 <= BUFSIZE <= 1MB (por defecto
1024).
-n MINLENGTH   Longitud mínima de la cadena. Mayor que 0 y menor que 256
(por defecto 4).
-i             Lista de ficheros de entrada separados por comas.

$ ./merge_mystrings_split -n -i fi1,fi2 fo1 fo2
Error: La longitud mínima de cadena debe ser mayor que 0 y menor de 256
Uso: ./merge_mystrings_split [-t BUFSIZE] [-n MIN_LENGTH] -i
FILEIN1[,FILEIN2,...,FILEINn] FILEOUT1 [FILEOUT2 ...]
-t BUFSIZE     Tamaño de buffer donde 1 <= BUFSIZE <= 1MB (por defecto
```

```
1024).
-n MINLENGTH   Longitud mínima de la cadena. Mayor que 0 y menor que 256
(por defecto 4).
-i             Lista de ficheros de entrada separados por comas.

$ ./merge_mystrings_split -n 300 -i fi1,fi2 fo1 fo2
Error: La longitud mínima de cadena debe ser mayor que 0 y menor de 256
Uso: ./merge_mystrings_split [-t BUFSIZE] [-n MIN_LENGTH] -i
FILEIN1[,FILEIN2,...,FILEINn] FILEOUT1 [FILEOUT2 ...]
-t BUFSIZE     Tamaño de buffer donde 1 <= BUFSIZE <= 1MB (por defecto
1024).
-n MINLENGTH   Longitud mínima de la cadena. Mayor que 0 y menor que 256
(por defecto 4).
-i             Lista de ficheros de entrada separados por comas.

$ ./merge_mystrings_split -i fi1,fi2 fo1 fo2
$ cat fo1
ABCDEFGHIJKLMNOPQRSTUVWXYZ
$ cat fo2
abcdefghijklmnopqrstuvwxyz
$ ./merge_mystrings_split -i fi1,fi2 fo1
$ cat fo1
AaBbCcDdEeFfGgHhIiJjKkLlMmNnOoPpQqRrSsTtUuVvWwXxYyZz

$ ./merge_mystrings_split -i fi1,fi2,fi3 fo1 fo2 fo3
$ cat fo1
ABCDEFGHIJKLMNOPQRSTUVWXYZ
$ cat fo2
abcdefghijklmnopqrstuvwxyz
$ cat fo3
123456789012345678901234567890123456
$ ./merge_mystrings_split -i fi1,fi2,fi3 fo1 fo2
$ cat fo1
A1bC3dE5fG7hI9jK1lM3nO5pQ7rS9tU1vW3xY5z

$ cat fo2
aB2cD4eF6gH8iJ0kL2mN4oP6qR8sT0uV2wX4yZ6
$ wc -c fo1
41 fo1
$ wc -c fo2
40 fo2
```

Entrega

La entrega consistirá en un único fichero `merge_mystrings_split.zip` que contendrá tanto las tres utilidades separadas (`merge_files.c`, `mystrings.c`, y `split_files.c`) como la que las ejecuta encadenadas (`merge_mystrings_split.c`). Todos los ficheros fuente deben compilar conforme a los ficheros `tasks.json` o `Makefile` incluidos en los ficheros `.tgz`, por ejemplo, con `gcc -ggdb3 -Wall -Werror -Wno-unused -std=c11 merge_mystrings_split.c -o merge_mystrings_split`. La primera línea de cada fichero fuente debe ser la directiva para compilar conforme al estándar POSIX: `#define _POSIX_C_SOURCE 200809L`.

Evaluación

Para aprobar la práctica es imprescindible que los programas pasen las pruebas de ejecución que se muestran en este documento, sin que esto signifique que no se puedan detectar otros errores con pruebas adicionales que el profesor haga durante la evaluación.

Se tendrá en cuenta la correcta indentación del código, su estructuración, el manejo de los posibles errores de las llamadas al sistema o funciones de biblioteca, así como la correcta gestión de la memoria dinámica. Otro punto importante, y por eso lo volvemos a recordar, es que se penalizará severamente el no usar bufferes para las lecturas/escrituras, es decir, leer o escribir byte a byte, pudiendo llegar a suspender la práctica, así como no contemplar lecturas o escrituras parciales si así se pide en el enunciado. Adicionalmente, las ineficiencias que se detecten en el código, también pueden ser objeto de penalizaciones.