# CLAB2 SO1819Q2

## Makefile (0,5 puntos)

Crea un **Makefile** que permita 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 solo los ficheros fuentes. Los programas deben generarse si, y solo si, ha habido cambios en los ficheros fuentes.

## Control de Errores y Usage (0,5 puntos)

Todos los programas deben incluir un control adecuado de los argumentos usando la función `Usage()` y deben controlar los errores en todas las llamadas al sistema.

## Ejercicio 1 (2,25 puntos)

Implementa un programa llamado **aleatorios.c** que acepta un parámetro de entrada (*N* de aquí en adelante). Este parámetro de entrada *N* indica la cantidad de números aleatorios que generará el programa, que lo hará mediante (1) inicializar la aleatoriedad del sistema random (con la línea "srand(getpid());") y luego (2) generando los números aleatorios con la línea de código:

```
num = rand()  %  1048576;
```

Esta línea asigna a la variable "num" un número aleatorio que va desde "0" a menor que 1048576. Dentro del bucle de creación de números aleatorios, en cada iteración, tenemos que comprobar que el número no haya sido generado anteriormente. Para ello, tendremos un vector de tantas posiciones como posibles valores aleatorios podamos generar. Por cada número aleatorio que se genera, "X", primero se comprueba si ya ha sido generado anteriormente comprobando si la posición "X" del vector está a 1 (ya ha sido generado) o 0 (aún no ha sido generado). Si ya había sido generado en una iteración anterior, éste no sirve y generaremos otro hasta conseguir uno nuevo, en cuyo caso ponemos a 1 la posición "X" del vector, escribimos el valor "X" en formato interno (es decir, int) por el canal 57 y pasamos a la siguiente iteración del bucle.

**NOTA:** *Para poder lanzar una prueba ejecuta el programa con la siguiente línea de comandos que generará 25 números enteros aleatorios y los pondrá en el fichero "file.dat". Puedes revisar el contenido del fichero "file.dat" mediante la línea de comando "#>xxd  file.dat" que muestra el valor hexadecimal de cada byte del fichero:*

```
#> ./aleatorios  25  57>  file.dat
```

A continuación, contesta en el fichero **respuestas.txt**:

a) Explica brevemente por qué, si ejecutamos "#>cat  file.dat" aparecen caracteres extraños en lugar del valor de los números por pantalla.
b) Si "file.dat" fuera una named pipe, ¿cuál sería el comportamiento del programa?

## Ejercicio 2 (4 puntos)

Implementa un código, llamado **todos.c**. El programa acepta un único parámetro de entrada (*N* de aquí en adelante). El proceso padre creará N procesos hijo que se ejecutarán de forma secuencial y que mutarán para ejecutar el programa cuyo código has desarrollado en el ejercicio anterior, poniendo el valor "100" como parámetro de entrada. Es decir, cada hijo generará 100 números aleatorios. Si no has podido realizar el código puedes utilizar el binario que adjuntamos en el control, llamado *"aleatorios_original.exe"*. El padre se comunicará con cada hijo mediante una **UNICA** pipe sin nombre.

De tal forma que, cada hijo, antes de mutar, configurará convenientemente la tabla de canales para que cuando escriba por el canal 57 lo haga sobre la pipe sin nombre. Por su parte, el padre leerá 100 números enteros, procedentes de "aleatorios" de cada hijo, y los escribirá en el mismo formato, es decir "int", en un fichero llamado *"salida.bin"*. Si el fichero no existe, lo creará con permisos de lectura y escritura para el propietario, lectura para el grupo y ninguno para el resto. Si el fichero ya existe, el contenido se trunca y por tanto se pierde. Además, el padre ha de liberar el PCB de cada hijo tras la muerte de cada uno de ellos.
Antes de finalizar el proceso padre, ha de posicionarse en la mitad del fichero, leer el número entero que ocupa esa posición y mostrar un mensaje por pantalla que indique en qué posición está y el valor del número que está en ese lugar.
Además, indica en el fichero **respuestas.txt** qué línea/s de comandos has de ejecutar para mostrar la traza de llamadas al sistema "read" que hace el programa.
**NOTA:** *Puedes comprobar el resultado usando el comando "#>xxd fichero", que muestra los offsets y valores en hexadecimal del contenido de un fichero. Por ejemplo, lanzando a ejecutar la línea de comandos "#>./todos 5" el fichero "salida.bin" debería ocupar 2000 Bytes, ya que tiene 500 números enteros, siendo el entero 250º, que está en el offset 1000 Bytes el que está en la mitad. Por lo que el mensaje será "La mitad la ocupa la posición 250 que contiene el numero XXX".*

## Ejercicio 3 (1,25 puntos)

Copia el código del ejercicio anterior y llámalo **todos2.c**. Haz los cambios que consideres oportunos para sustituir la pipe sin nombre por una pipe con nombre, donde además el código ha de asegurarse que la pipe exista y en caso contrario, la crea.

## Ejercicio 4 (1,5 puntos)

Responde, en el fichero **respuestas.txt**, razonando brevemente tus respuestas a las siguientes preguntas:

A) **(0,5 puntos)** Indica qué líneas de comando has de ejecutar y qué campo indica el número de hardlinks que tiene el inodo correspondiente al directorio $HOME de tu cuenta.

B) **(0,5 puntos)** Dispón de dos terminales. Indica qué líneas de comando has de ejecutar para crear un soft-link en el directorio $HOME de tu cuenta que apunte al dispositivo "terminal" que utiliza una de las dos ventanas de terminal. Luego, indica la línea de comandos necesaria para redirigir la salida de un comando (el que quieras) a esa otra terminal.

C) **(0,5 puntos)** Indica qué pasos has de seguir y líneas de comando a ejecutar para comprobar si el major number del driver que quieres instalar, de un dispositivo a nivel de carácter, está actualmente en uso o no.

## Qué se tiene que hacer

- El Makefile
- Los códigos de los programas en C
- La función Usage() para cada programa
- El fichero respuestas.txt con las respuestas a las preguntas

# CLAB2 SO1819Q2

## Qué se valora

- Que sigas las especificaciones del enunciado
- Que el uso de las llamadas a sistema sea el correcto
- Que se comprueben los errores de **todas** las llamadas al sistema
- Código claro y correctamente indentado
- Que el Makefile tenga bien definidas las dependencias y los objetivos
- La función Usage() que muestre en pantalla cómo debe invocarse correctamente al programa en caso que los argumentos recibidos no sean adecuados.
- El fichero respuestas.txt

## Qué hay que entregar

Un único fichero tar.gz con el código fuente de los programas en C, el Makefile, y las respuestas en respuestas.txt:

```
tar zcvf clab2.tar.gz Makefile respuestas.txt aleatorios.c todos.c todos2.c
```