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: Suma de enteros (1,5 punto)

Escribe un programa llamado suma_enteros.c que recibe como parámetro el nombre de un fichero de texto. El fichero contiene números enteros en formato interno de la maquina (formato entero) sin ninguna separación. El programa deberá leer los números del fichero y sumarlos todos. El resultado de la suma se debe imprimir por el terminal, y además se debe almacenar en un fichero llamado suma.txt. Si suma.txt no existe, debe crearse con permisos de lectura y escritura para el usuario, y si existe queremos que el contenido final incluya solamente el resultado de la última ejecución. Os proporcionamos un fichero de entrada de prueba llamado numeros.txt. Por ejemplo:

```
$ ./suma_enteros numeros.txt
55
$ cat suma.txt
55
```

Ejercicio 2: Cuenta líneas (3 puntos)

Crea un programa que se llame cuenta_lineas.c. El programa debe recrear el comportamiento del siguiente comando:

```
$ cat <parametro1> | grep <parametro2> | wc –l
```

Donde <parametro1> y <parametro2> son parámetros del programa. Para conseguirlo el programa debe crear un total de tres hijos y tantas pipes sin nombre como sean necesarias para el envío de datos entre los hijos. El primer hijo muta para ejecutar el comando cat sobre el fichero especificado en el primer parámetro del programa. El segundo hijo muta para ejecutar el comando grep que busca, en la salida de cat, la palabra que el programa recibe como segundo parámetro. El tercer hijo muta para ejecutar el comando wc **–**l, sobre la salida de grep, y escribe su salida por la salida estándar. El padre espera a que todos los hijos terminen su ejecución y luego acaba. Un ejemplo de ejecución que cuenta cuantas líneas contienen la palabra execlp en el fichero cuenta_lineas.c seria:

```
$ ./cuenta_lineas cuenta_lineas.c execlp
3
```

Ejercicio 3: Ordenar números (3 puntos)

Crea un programa que se llame ordenar.c. Este programa recibe como parámetro el nombre de un fichero que os proporcionamos (para-ordenar.txt), que contiene caracteres numéricos del 0 al 9 sin espacios entre caracteres y sin carácter de final de línea. El programa debe ordenar ascendentemente los números leyendo y escribiendo en el mismo fichero de entrada usando el siguiente algoritmo. El algoritmo lee caracteres adyacentes de dos en dos empezando en la posición 0, y si no están en el orden correcto los intercambia. Luego lee otra vez dos caracteres adyacentes

des de la posición 1, y si no están en el orden correcto los intercambia. Así sucesivamente hasta que no se puedan leer 2 caracteres. Si ha habido algún intercambio hay que volver a realizar el proceso hasta que el algoritmo detecte que todos los números están ordenados. Por ejemplo, con la entrada que os proporcionamos:

Primera pasada:
"31142" → "13142" Lee los dos primeros caracteres y los intercambia ya que 3 > 1
"13142" → "11342" Lee los dos siguientes des de la posición 1 y los intercambia ya que 3 > 1
"11342" → "11342" No intercambia
"11342" → "11324" Intercambia ya que 4 > 2

Segunda pasada ya que han habido intercambios en la primera:
"11324" → "11324" No intercambia
"11324" → "11324" No intercambia
"11324" → "11234" Intercambia ya que 3>2
"11234" → "11234" No intercambia

Aunque ya esta ordenada la secuencia, hay que hacer una tercera pasada para que el algoritmo lo compruebe y acabar la ejecución. El fichero recibido como parámetro debe actualizarse con cada intercambio y terminar con la secuencia ordenada.

Ejercicio 4: Contesta las siguientes preguntas en el fichero respuestas.txt, debes justificar todas las respuestas (1,5 puntos) (0,75 cada una)

1. ¿Qué dirección de memoria devuelve la llamada a sbrk(0)? ¿Qué pasa si intentamos acceder a esta dirección?

2. Crea un soft-link a numeros.txt y llámalo numeros.txt.sl. ¿Están ambos ficheros relacionados con el mismo inodo? ¿Por qué? ¿Qué comando/s has usado para comprobarlo?

Qué se tiene que hacer

- El Makefile
- Los códigos de los programas en C
- El fichero respuestas.txt con las respuestas a las preguntas

Qué se valora

- Que sigas las especificaciones del enunciado
- Que el uso de las llamadas a sistema sea el correcto y comprobación de errores
- 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 el programa en caso que los argumentos recibidos no sean adecuados.

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 *.c
```