`Makefile` (0,5 puntos)

Cread un `Makefile` que permita generar todos los programas del enunciado a la vez y cada uno de ellos por separado. Añadid una regla (`clean`) para borrar todos los binarios y/o ficheros objeto. Los programas deben generarse si y sólo si ha habido cambios en los ficheros fuentes.

`Usage()` y tratamiento de errores (0,5 puntos)

Los códigos que debéis desarrollar han de verificar si los parámetros recibidos son los esperados. Si no lo son, deben invocar a una rutina llamada `usage()` que muestre cómo debe ser invocado el programa. Los programas deben realizar el tratamiento de errores en todas las llamadas al sistema.

Procesos, memoria, entrada/salida y *signals* (8,0 puntos)

A) [4,0 puntos] Implementad un programa (llamadlo `calc1.c`) que espere dos parámetros: el número máximo de procesos hijos que debe crear (`N`) y un nombre de fichero (`file`). El proceso debe crear `N` procesos hijos que ejecutarán concurrentemente el programa `hijo1.c` (os facilitamos su código fuente, **no tenéis que modificarlo**) pasándole como parámetro el nombre del fichero `file`. Antes de la creación de cada uno de los procesos restantes para completar `N`, el proceso original debe verificar si ha acabado algún proceso de los ya creados, en cuyo caso si ha finalizado correctamente (con código de finalización igual a 0), debe indicar su pid y el mensaje "ended correctly", matar al resto y finalizar el proceso de creación de hijos. En caso que algún proceso haya finalizado con código incorrecto, debe imprimir su pid y la causa de finalización y continuar con la creación de hijos. El proceso original y todos los hijos utilizarán el mismo dispositivo como salida estándar.
Una vez acabado el proceso de creación de hijos, `calc1` esperará la muerte de los hijos creados que queden. Para cada hijo finalizado debe ahora aplicar mismo tratamiento que durante el proceso de creación de hijos.

Si ningún proceso hijo finaliza correctamente, imprimirá `Incalculable`. Todos estos mensajes se mostrarán por el canal de error.

Observaciones:

- Si examináis el código de `hijo1.c` veréis que el fichero recibido de entrada debe contener números enteros en formato ASCII separados por un salto de línea. El programa escribe por su salida estándar su pid y la suma de dichos números y finaliza con `exit(0)`. El código incorpora un retardo de tiempo (aleatorio, hasta 6 segundos) para conseguir una cierta variabilidad en el tiempo de ejecución del programa. Además, aleatoriamente, el programa puede finalizar con un código de finalización diferente de 0 o por la recepción de un signal no tratado; ésto simula que el programa no ha podido realizar su cometido correctamente.

- Dado el comportamiento aleatorio del programa `hijo1.c`, pruebas sucesivas pueden mostrar diferencias en cuanto a los motivos de finalización errónea y a si el resultado es calculable.

- Os proporcionamos varios ficheros de prueba (`test*.txt`).

- No podéis asumir un valor máximo de `N`.

- En las pruebas de funcionamiento, se aconseja que realicéis ejecuciones con `N`=10.

- En la última página disponéis de un ejemplo orientativo del resultado esperado.

B) [1,0 puntos, dependiente de A)] Para el caso que al menos un proceso hijo finaliza correctamente, ¿podéis garantizar que el último mensaje será "`ended correctly`"? **Justificad** la respuesta en el fichero `respuestas.txt`; en caso que vuestro `calc1.c` no lo garantice, modificad el código de `calc1.c` (llamadle `calc2.c`) para garantizarlo (la solución debe mantener la ejecución concurrente de los procesos hijos).

C) [1,5 puntos, dependiente de A) e independiente de B)] Modificad el programa `calc1.c` (llamadle `calc3.c`) para incorporar el tratamiento de signals:

- Si recibe `SIGINT`, debe imprimir el mensaje `SIGINT` por su canal de error, matar a todos los procesos hijos creados hasta el momento y finalizar.
- Además, cada segundo ha de mostrar un mensaje por el canal de error indicando el número de segundos que el programa lleva en ejecución y cuántos hijos ha creado hasta el momento.

Observaciones:

- Todas las esperas que realice `calc3.c` han de ser bloqueantes.
- Para realizar las pruebas, podéis aumentar el valor de la constante `MAXDELAY_S` en el programa `hijo1.c`

D) [1,5 puntos, dependiente de A) e independiente de B) y C)] Modificad el programa `calc1.c` (llamadle `calc4.c`) para permitir que los números de entrada se reciban del canal de entrada estándar. En este programa `calc4.c` esperará un único parámetro (`N`, el número de procesos hijos).

Los procesos hijos tendrán que ejecutar el programa `hijo2.c`, prácticamente idéntico a `hijo1.c` excepto en lo referente a la lectura de los números: `hijo2.c` los lee directamente del canal de entrada estándar. Consecuentemente, `hijo2.c` no espera parámetros.

Como `calc1.c`, `calc4.c` deberá indicar cuál de los hijos es el primero en finalizar correctamente y matar al resto de los hijos.

Observaciones:

- Añadid un comentario al inicio de `calc4.c` describiendo brevemente vuestra solución.
- No podéis modificar el código del programa `hijo2.c`

Pregunta (1,0 puntos)

Ejecutad el script `dir.sh` que os hemos facilitado. Si examináis su código veréis que crea una estructura de directorios y muestra el resultado de ejecutar `ls -la`. Explicad justificadamente en el fichero `respuestas.txt` los valores numéricos mostrados en la segunda columna del resultado (correspondiente al número de enlaces físicos a cada fichero).

Qué se valora

- Que sigáis las especificaciones del enunciado.
- Que el uso de las llamadas a sistema sea el correcto y 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 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 `calc*.c`, `Makefile` y `respuestas.txt`:

```
tar zcvf final.tar.gz Makefile respuestas.txt calc*.c
```

Ejemplos orientativos del resultado de `calc1`

```
prompt$ ./calc1 10 test01.txt
26631 wrong execution (exit code 2)
26623 wrong execution (exit code 7)
26630 wrong execution (exit code 7)
26628 wrong execution (signal 10: User defined signal 1)
26629 wrong execution (exit code 1)
26627 wrong execution (exit code 5)
26625 wrong execution (signal 14: Alarm clock)
26632 wrong execution (exit code 8)
26626 wrong execution (exit code 6)
26624 : 9999999999
26624 ended correctly
prompt$ ./calc1 4 test00.txt
8666 wrong execution (exit code 6)
8663 wrong execution (exit code 9)
8664 wrong execution (signal 11: Segmentation fault)
8665 wrong execution (exit code 3)
Incalculable
prompt$
```