Makefile (0,5 puntos)

Crea un Makefile para compilar todos los programas del examen, utilizando una regla para todos y utilizando reglas individuales para cada uno. Incluye una regla llamada "clean" para eliminar todos los archivos binarios y objetos, pero conservando los archivos fuente. Cada binario debe compilarse si y solo si sus archivos fuente se han actualizado.

Control de errores (0.5 puntos)

Para todos los siguientes programas, debes verificar los errores de TODAS las llamadas al sistema (excepto la escritura en pantalla), controlar los argumentos de entrada y definir la función Usage() si es necesario.

Ejercicio 1 (2,25 puntos)

Crea un programa en C, llamado e1.c, que reciba un único argumento (n). Comprueba[1] que sea un número natural, donde: 0<n<5. El programa crea tantos procesos concurrentes como indique su argumento n. Cada proceso hijo cambia su imagen (muta) a la del programa **clab** (que adjuntamos con el enunciado), sin argumentos. El proceso padre espera a todos sus hijos, muestra por salida estándar el PID (*process identifier*) del hijo muerto y termina. Ejemplo de ejecución (los PIDs pueden variar):

```
$ ./e1 2
Hijo con PID 78146 ha finalizado
Hijo con PID 78147 ha finalizado
$ ./e1
Error: ús: ./e1 <num_proc>
```

Ejercicio 2 (3,5 puntos)

Modifica el programa anterior, nómbralo `e2.c`. Como antes, este programa recibe un único argumento n y crea tantos procesos concurrentes como indique n. Ahora es importante guardar el orden de creación de los hijos, que va de 0 a n-1, donde n<5. Como antes, cada proceso hijo cambia su imagen (muta) a la del programa clab, pero ahora clab se invoca con el argumento entero 255. El proceso padre debe esperar a todos sus hijos sin bloquearse en ningún momento, realizando una espera activa y capturando el signal SIGCHLD. Para cada hijo muerto, el padre muestra el PID del hijo, el orden en el que ha terminado, su índice de creación y el valor que ha devuelto mediante exit. Desde otro terminal, debes enviar una señal SIGUSR1 a cada uno de los hijos para que terminen.

```
$ ./e2 2
Hijo con PID 96558 (creado el segundo, ha acabado el primero) ha finalizado con codigo 11
Hijo con PID 96557 (creado el primero, ha acabado el segundo) ha finalizado con codigo 19
```

¿Como lo has probado? (0,5 puntos)

Indica en el fichero resposta.txt que comandos has ejecutado para testear el ejercicio 2.

Ejercicio 3 (2,75 puntos)

Implementa tu versión del programa `clab.c` que has usado en los ejercicios anteriores. Este programa acepta un argumento que indica el número de segundos, entre 0 y 255, que

---

[1] Recomendamos usar la funcion `strlol` en lugar de `atoi` para convertir cadenas de caracteres a enteros, ya que facilita la gestión de errores cuando la cadena de caracteres no es un número válido.

transcurren antes de enviar un SIGALRM a sí mismo. Si no recibe ningún argumento, el número de segundos es 1. En cualquier caso, al recibir el signal SIGUSR1, devuelve (a través de exit) el número de segundos que ha estado ejecutándose, si este es menor que 255. Si no recibe el signal SIGUSR1 antes de 255 segundos o recibe un SIGALRM, devuelve 0. Mientras espera la llegada de los signals (ALRM o USR1), no consume CPU. Este programa, si los argumentos son correctos, no escribe nada por pantalla.

Que se valora:

- Que se sigan las especificaciones del enunciado
- Que el uso de las llamadas al sistema sea correcto
- Que todas las llamadas al sistema sean verificadas para detectar errores
- Que el código sea claro y esté correctamente indentado
- Que el Makefile tenga dependencias y objetivos bien definidos
- Que la función Usage() muestre en pantalla cómo invocar correctamente el programa en caso de que los argumentos recibidos no sean adecuados.

Que entregar:

Un unico fichero tar.gz con el codigo de todos los programas, el Makefile, y el fichero resposta.txt:

```
tar zcvf clab.tar.gz Makefile *.c resposta.txt
```