`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 y, en caso que no lo sean, 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 y *signals* (5,0 puntos)

A) [2,0 puntos] Implementad un programa (llamadlo `spawnA.c`) que espere un parámetro entero: el número de procesos hijos que debe crear (`N`). El proceso debe crear `N` procesos hijos que ejecutarán el programa `hijo1.c` (os facilitamos su código fuente).

Una vez creados, el programa entrará en un bucle infinito donde, sin realizar espera activa, monitorizará la muerte de sus procesos hijos. Cuando detecte la muerte de alguno de sus hijos, creará un nuevo proceso hijo que también ejecutará el programa `hijo1.c`. De esta forma, el proceso intentará tener siempre `N` procesos hijos vivos ejecutando `hijo1.c`.

Observaciones:

- `hijo1.c` espera como parámetro un entero que representa el orden de creación de los hijos (el primer hijo tendrá el orden 0, el segundo tendrá el orden 1, …).
- Se aconseja ejecutar `spawnA` desde un terminal y desde otro provocar la muerte de procesos `hijo1`. Haced que `spawnA` imprima qué está haciendo.
- Para matar todos los procesos `hijo1` podéis utilizar la orden `killall hijo1`

B) [1,0 puntos, dependiente de A)] Modificad el programa del apartado A) (llamadlo `spawnB.c`) para que cada vez que reciba el *signal* `SIGUSR2` añada un nuevo proceso al conjunto de procesos hijos que ejecutan `hijo1`. Consecuentemente, a partir de este momento, se incrementará en una unidad el número de procesos hijos que deben mantenerse vivos.

C) [1,0 puntos, dependiente de A) e independiente de B)] Modificad el programa del apartado A) (llamadlo `spawnC.c`) de forma que cada vez que reciba el *signal* `SIGUSR1` escriba por su salida estándar los *pids* de los `N` procesos hijos vivos actualmente.

Observación:

- No podéis asumir un valor máximo de `N`. Tendréis que resolver el problema utilizando memoria dinámica.

D) [1,0 puntos, dependiente de A), B) y C)] Combinad las funcionalidades B) y C) en un mismo código (llamadlo `spawnD.c`). Debéis tener en cuenta que el apartado B) permite incrementar el valor de `N` y cómo afectará ésto a la funcionalidad del apartado C).

Observaciones:

- En relación a la memoria dinámica, sólo podéis solicitar memoria utilizando `malloc()`. No podéis utilizar otras rutinas como `realloc()`, `reallocarray()` o similares.
- Mientras se esté tratando un *signal*, el resto de *signals* debe estar bloqueado.

Comunicación y entrada/salida (2,5 puntos)

E) [1 punto, independiente del ejercicio anterior] Modificad el código de `hijo1.c` (llamadlo `hijo2.c`) de forma que en el cuerpo del bucle infinito substituyáis su contenido actual por:

- la lectura por entrada estándar de un número natural (en formato binario). Dicho natural lo interpretaremos como la posición en la que debemos posicionarnos en el fichero `input.txt` (os subministramos uno de ejemplo)
- la lectura del carácter almacenado en dicha posición del fichero `input.txt`
- la escritura por salida estándar del carácter leído (aprovechad el `sprintf` que también muestra el *pid* y el número de orden del hijo).

  Observación: notad que `hijo2` debe seguir realizando un bucle infinito.

F) [1,5 puntos, dependiente de A) y E)] Modificad el código de `spawnA.c` (llamadlo `spawnF.c`) de forma que invoque a `hijo2` en vez de a `hijo1`. Además, preparad que todos los hijos creados tengan su entrada estándar redireccionada a una *pipe* con nombre llamada `NP` (podéis crearla desde la línea de órdenes).

Observación:

- Para enviar datos a la *pipe* con nombre podéis utilizar el programa `echo_bin.c` (os subministramos dicho código) . Analizad cómo está implementado antes de utilizarlo.

Preguntas (1,5 puntos)

Contestad **justificadamente** en el fichero `respuestas.txt` a las siguientes preguntas:

A) **[0,5 puntos]** En el apartado F), ¿sería factible que el fichero `input.txt` fuese abierto una única vez por `spawnF.c` y que los procesos `hijo2` heredasen el puntero de lectura/escritura sobre dicho fichero?

B) **[1,0 puntos]** En el apartado F), indicad qué órdenes habéis ejecutado para crear la *pipe* con nombre y probar el funcionamiento del programa. Adjuntad la salida obtenida.

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 `spawn*.c`, `hijo2.c`, `Makefile` y `respuestas.txt`:

```
tar zcvf final.tar.gz Makefile respuestas.txt spawn*.c hijo2.c
```