Makefile (0,5 puntos)

Crea un fichero 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 sólo los ficheros fuente. Los programas deben generarse si, y solo si, ha habido cambios en los ficheros fuente.

Control de errores (0,5 puntos)

Para todos los programas que se piden a continuación deben comprobarse los errores de TODAS las llamadas a sistema (excepto el write por pantalla), controlar los argumentos de entrada y definir la función Usage() si es necesario.

Ejercicio 1 (2,5 puntos)

Haz un programa que llamaremos "busqueda_parcial.c". Este programa recibirá tres argumentos, el primero será un número que corresponderá con un PID (que no usaremos en este ejercicio), el segundo será una palabra y el tercero un nombre de fichero. El programa creará un proceso nuevo que mutará al programa grep recibiendo como parámetro la palabra y el nombre de fichero (grep palabra fichero). El programa grep termina con un exit(0) cuando encuentra la palabra, exit(1) si no la encuentra y exit(2) si hay un error. Haz que el proceso inicial controle como ha terminado su hijo y escriba un mensaje por pantalla indicando la palabra, el fichero y el valor devuelto en el exit.

Ejemplo de salida:

```
[clab1]$ ./busqueda_parcial 100 main busqueda_parcial.c
void main(int argc, char *argv[])
Termina grep palabra main fichero busqueda_parcial.c : exit 0
[clab1]$ ./busqueda_parcial 100 palabra_que_no_esta busqueda_parcial.c
Termina grep palabra palabra_que_no_esta fichero busqueda_parcial.c : exit 1
```

Ejercicio 2 (2,5 puntos)

Copia busqueda_parcial.c en busqueda_parcial_con_espera.c. Modifícalo para que el programa espere la recepción de los signals SIGUSR1 y/o SIGUSR2 antes de crear el nuevo proceso y hacer nada (se recibirá uno de los dos). La espera ha de ser sin consumir CPU. Se pide esperar a los dos signals con una única función. La función de atención a los signals escribirá un mensaje indicando el signal recibido.

- Si se ha recibido un SIGUSR1 el programa continuará su ejecución.
- Si se recibe un SIGUSR2 no se continuará la ejecución. Además, en caso que el PID recibido sea diferente de cero se enviará el signal SIGUSR2 a ese proceso.

En caso de recibir un SIGUSR1 y continuar el programa, el proceso inicial, después de tratar el valor del exit del hijo, enviará un SIGUSR1 al proceso con el PID indicado por parámetro si el valor del exit de su hijo era 1 y un SIGUSR2 si es 0. Esta última parte la haremos únicamente si el PID recibido es distinto de cero. Después de procesar el valor del exit del hijo el proceso terminará con un exit(0).

Ejemplo de salida (en el primer caso se recibe un SIGUSR1 y en el segundo un SIGUSR2):

```
[clab1]$ ./busqueda_parcial_con_espera 0 main busqueda_parcial.c
SIGNAL recibido 10
void main(int argc, char *argv[])
Termina grep palabra main fichero busqueda_parcial.c : exit 0
[clab1]$ ./busqueda_parcial_con_espera 0 main busqueda_parcial.c
SIGNAL recibido 12
```

Ejercicio 3 (3,5 puntos)

Haz un programa que llamaremos “busca_multiple.c”. Este programa recibe un número variable de parámetros, entre 2 y 10. El primer parámetro, obligatorio, será una palabra. El resto se corresponderán con nombres de ficheros (mínimo 1).

El programa creará tantos procesos como nombres de ficheros recibidos. Antes de crear los procesos, el programa ha de bloquear los signals SIGUSR1 y SIGUSR2 para evitar que sus hijos reciban algún signal antes de tenerlo configurado.

Para cada hijo, el proceso mutará al programa “busqueda_parcial_con_espera“, pasándole como parámetros el PID de su último hermano, la palabra y un fichero (uno para cada uno). En el caso del primer hijo le pasaremos un 0 como PID.

Después de crear los procesos de forma concurrente, enviaremos un SIGUSR1 al último hijo creado y esperaremos a que terminen.

Ejemplo de salida (el primer proceso busca la palabra y como la encuentra y envía un SIGUSR2 al segundo este ya no la busca):

```
[clab1]$ ./busca_multiple main busca_multiple.c busca_multiple.c
SIGNAL recibido 10
void main(int argc, char *argv[])
Termina grep palabra main fichero busca_multiple.c : exit 0
SIGNAL recibido 12
```

¿Cómo probarlo? (0,5)

Indica en el fichero respuesta.txt que comandos ejecutarías para testear el ejercicio 2 ya que en ese caso nadie envía ningún signal automáticamente.

Qué hay que hacer

- El makefile
- Los códigos de los programas en C
- La función Usage() para cada programa que sea necesario

Qué se valora

- Que sigas las especificaciones del enunciado
- Que el uso de las llamadas al sistema sea el correcto
- Que se comprueben los errores de todas las llamadas al sistema
- Que el código sea claro y correctamente indentado
- Que el Makefile tenga bien definidas las dependencias y objetivos
- Que la función Usage() muestre por pantalla como debe invocarse correctamente el

programa en el caso que los argumentos recibidos no sean los adecuados

- El fichero respuesta.txt

Qué hay que entregar

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

```
tar zcvf clab1.tar.gz makefile *.c respuesta.txt
```