Makefile (0.5 puntos)

Crea un fichero Makefile que compile automáticamente todos los ejercicios del exàmen. El Makefile debe incluir las reglas *all* y *clean*. Debe contener además un target por ejecutable.

Función Usage (0.5 puntos)

Todos los programas deben incluir la función **usage.**

Control de errores (1 punto)

Todos los programas deben incluir el **control de errores** para las llamadas a sistema (except escrituras en la salida std)para asegurar que son lo más robusto posibles.

Contesta en el fichero answers.txt (2 puntos)

Contesta las siguientes preguntas, en el fichero answers.txt, indicando el comando utilizado para cada caso.

- ¿Cuantos inodos libres hay en el sistema de ficheros?
- ¿Cuál es el número de inodo del fichero / ?
- ¿Cuántos enlaces (links) tiene el inodo raiz?
- ¿Cuál es la lista de la llamada a sistema "read" ejecutadas durante la ejecución del comando "ls –la /"?

set_holes.c (3 puntos)

Crea un nuevo programa llamado set_holes.c. Este programa recibe 2 parámetros, los dos corresponden a nombres de ficheros. El primero el un fichero con números en formato entero. Estos números corresponden con posiciones (bytes) del segundo fichero q deben rellenarse con el carácter '0'. El programa set_holes debe comprobar que todas las posiciones indicadas en el primer fichero son correctas. Por ejemplo, si el contenido del fichero "positions" fuera 0,4,5,9 . Os damos un fichero con números para q podáis probar.

$ cat 0_to_9.txt
0123456789
$set_holes positions 0_to_9.txt
$ cat 0_to_9.txt
0123006780

Condición de carrera (race condition) (3 puntos)

Lee atentamente el siguiente código:

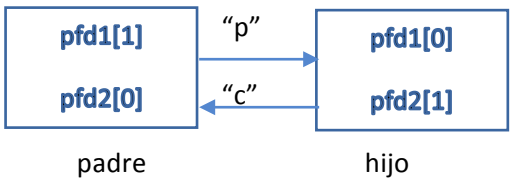
```
/* race_cond.c */
char parent[] = "123456789";
char child[] = "ABCDEFGHIJ";
int main(void) {
```

```
    pid_t pid;int j;
    if ((pid = fork())<0) {
        perror ("error->fork"); exit(1);
    }
    else if (pid==0) {
        for (j=0;j<strlen(child);j++) {
            write(1,(child+j),1);
        }
    }
    else if (pid>0) {
        for (int j=0;j<strlen(parent);j++) {
            write(1,(parent+j),1);
        }
    }
    exit(0);
}
```

Este código contiene una condición de carrera ya que la salida depende del orden en el cual los procesos se ejecutan por el kernel y durante cuanto tiempo. Queremos modificar este código de forma que la salida de padre e hijo se intercale de forma controlada. Mediante un parámetro indicaremos cuantos caracteres de cada uno escribiremos antes de "dar paso" al siguiente (alternando padre-hijo). Por ejemplo:

$ ./race_cond 1
1A2B3C4D5E6F7G8H9I
$ ./race_cond 2
12AB34CD56EF78GH9I

Para evitar la aleatoriedad y controlar los turnos, se pide crear cinco funciones nuevas y utilizarlas en el código. Estas funciones crearán dos pipes sin nombre y ofrecerán la funcionalidad para comunicar/sincronizar padre-hijo. Cada proceso le dirá al otro (vía pipe) cuando ha terminado su turno (y por lo tanto el otro puede empezar) y esperará a que sea su turno otra vez. Las funciones son:



TELL_WAIT(); /* Crea dos pipes: pfd1 y pfd2 */

TELL_PARENT(); /* El hijo escribe el carácter "c" en la segunda pipe */

TELL_CHILD(); /* El padre escribe el carácter "p" en la primera pipe*/

WAIT_PARENT(); /*El hijo lee el carácter "p" de la primera pipe */

WAIT_CHILD(); /* El padre lee el carácter "c" de la segunda pipe */

Implementa las funciones y inserta las llamadas en el código de forma que cumpla con la funcionalidad esperada.

A tener en cuenta…..

- Debes seguir las especificaciones
- El uso correcto de las llamadas a sistema
- Genera un código que sea comprensible utilizando las reglas de indentación

A entregar

Un único fichero clab2.tar.gz con el código fuente de todos los ejercicios, el Makefile y el fichero answers.txt