Makefile (0,5 punts)

Crea un fitxer Makefile que permeti generar tots els programes de l'enunciat alhora i cadascun per separat. Afegeix una regla (clean) per esborrar tots els binaris i/o fitxers objecte, i deixar només els fitxers font. Els programes s'han de generar si, i només si, hi ha hagut canvis als fitxers font.

Control d'errors (0,5 punts)

Per a tots els programes que es demanen a continuació cal comprovar els errors de TOTES les crides a sistema (excepte el write per stdout/stderr), controlar els arguments d'entrada i definir la funció Usage() si cal.

Exercici 1 (1,5 punts)

Fes un programa anomenat **parser.c** que buscarà els salts de línia dins d'un text. Si el programa rep el nom d'un fitxer com paràmetre d'entrada, *nom_fitxer*, l'obrirà per llegir-ho. En cas de no rebre cap paràmetre, llegirà el text des de l'entrada estàndard. Cada vegada que trobi un salt de línia (és a dir, el caràcter '\n') es guardarà la seva posició dins del fitxer (és a dir, l'offset) en un vector d'enters. Aquest vector d'enters es guardarà en la regió de memòria heap (es valorarà que s'utilitzi un únic vector amb tants elements com nombre de salts de línia trobi, però sense reservar més memòria del compte en el heap). Una vegada arribat al final del fitxer, s'escriurà amb una única crida al sistema (es valorarà que només s'utilitzi una única crida per fer-ho), tot el contingut del vector d'enters (en format intern) pel canal número 4. Al final, abans de finalitzar, s'escriurà un missatge pel canal d'error que indiqui el nombre total de salts trobats i la quantitat de memòria heap, en Bytes, utilitzada per guardar aquest vector.

Per validar aquest exercici, hem proporcionat uns fitxers de mostra ("text1", "text2", "text3" que són còpies dels fitxers originals "text1_orig", "text2_orig", "text3_orig", respectivament). Es tracta de fitxers de text que tenen els salts de línia en els offsets indicats en els fitxers "text1_orig.out", "text2_orig.out", "text3_orig.out", respectivament. El contingut d'aquest segon conjunt de fitxers es pot analitzar amb la línia de comandes "xxd file". Aquesta comanda mostra, en format hexadecimal, el valor dels bytes del fitxer indicat com paràmetre d'entrada.

Exemples de sortida1:

```
$ ./parser  text1 4> text1.out
El text te 5 salts de línia i shan utilitzat 20 Bytes del heap

$ ./parser < text1 4> text1.out
El text te 5 salts de línia i shan utilitzat 20 Bytes del heap

$ cat text1 | ./parser 4> text1.out
El text te 5 salts de línia i shan utilitzat 20 Bytes del heap
```

NOTA: La línia de comandes "xxd text1.out" hauria de mostrar la següent sortida:

```
00000000: 0100 0000 0400 0000 0800 0000 0d00 0000  ................
00000010: 1800 0000                                ....
```

Aquests valors en hexadecimal corresponen als nombres en enter (format intern de la màquina) les posicions "1", "4", "8", "13" i "24", respectivament. El contingut del fitxer "text1.out" hauria de ser el mateix que "text1_orig.out". La columna de l'esquerra (abans dels ':') indica, en hexadecimal, la posició dins del fitxer que mostra la comanda "xxd" i la columna de la dreta (els punts) la representació en ASCII si és possible (en cas contrari, posa un punt) de cada Byte.

Exemples de sortida2 i 3:

```
$ ./parser text2 4> text2.out
El text te 7 salts de línia i shan utilitzat 28 Bytes del heap

$ ./parser text3 4> text3.out
El text te 18 salts de línia i shan utilitzat 72 Bytes del heap
```

ATENCIÓ! Aquest exercici s'avaluarà en funció de l'execució correcta del programa segons se indica a continuació: 1 punt si funciona com s'espera; 0,25 punts addicionals si, funcionant com s'espera, la quantitat de memòria heap que es reserva i s'assigna és la justa i necessària per guardar els salts de línia que es trobin; i 0,25 punts addicionals, si funcionant com s'espera, s'utilitza una única invocació a la crida al sistema corresponent per escriure tot el contingut del vector d'enters pel file descriptor "4".

Exercici 2 (2 punts)

Fes un programa anomenat **modifier.c** que acceptarà al menys dos paràmetres d'entrada: un nom i, al menys, un número (pots assumir que són números positius). L'objectiu d'aquest programa és canviar els salts de línia del fitxer de text per un guió (és a dir, '-'). El programa obrirà el fitxer passat com paràmetre d'entrada, *nom_fitxer*, i el seu corresponent ".out" creat en l'exercici anterior. Els números indicaran els salts de línia que es volen canviar. Es valorarà mostrar missatges d'error per pantalla per aquells números passats com paràmetre d'entrada que no ocupen posicions vàlides dins del fitxer ".out", així com que el programa NOMÉS accedeixi a les posicions indicades, tant en el fitxer ".out" com en el fitxer de text. El canvi s'actualitzarà en el mateix fitxer de text. El fitxer ".out" NO s'ha de modificar. **Per lo que abans de fer una prova s'aconsella utilitzar una còpia del fitxer original.**

Exemple de sortida 1:

```
$ ./modifier  text1 2 20
El salt de línia 20 no esta en el rang daquest fitxer
```

El fitxer "text1" passaria a tenir:

```
1
23-456
7890
1234567890
```

Exemple de sortida 2:

```
$ ./modifier  text2 1 2 3 8 4 5 6 7 9
El salt de línia 8 no esta en el rang daquest fitxer
El salt de línia 9 no esta en el rang daquest fitxer
```

El fitxer "text2" passaria a tenir:

```
ABCD--EFGHIJKLMNO-PQ-RSTUVW-XYZ-1234567890-
```

Exercici 3 (5,5 punts)

Fes un programa anomenat **multiparser.c** que acceptarà tants paràmetres d'entrada com noms de fitxers de text que es vulguin processar. ***NOTA:*** *es recomana utilitzar fitxers d'entrada amb noms diferents.* Abans de res, el procés principal ha d'assegurar-se de bloquejar tots els signals.

Per cada paràmetre d'entrada, es crearà un procés child, seguint un esquema de creació i execució seqüencial. Per cada procés creat, el procés principal activarà una alarma d'un segon i esperarà, sense consumir CPU, fins rebre el signal corresponent. Al rebre'l el procés principal ha de "desbloquejar" al procés child utilitzant una pipe amb nom, però sense enviar dades. A continuació, el procés principal escriurà el contingut del fitxer i-èssim a una pipe sense nom connectada al procés child corresponent. Una vegada processat tot el fitxer, el procés principal monitoritzarà la finalització del procés fill i mostrarà per pantalla si ha finalitzat correctament o no. Per últim, el procés principal ha de finalitzar mostrant un missatge pel canal d'error que indiqui "Final del proces principal".

Per la seva part, el procés child i-èssim ha de crear un fitxer de sortida que tindrà el mateix nom que el fitxer que processarà, però afegint el sufix ".out". Per exemple, el fitxer "file" implicaria crear "file.out". Si no existeix, es crea amb permisos de lectura i escriptura pel usuari creador,

lectura pel grup i res pels altres. Si el fitxer ja existeix, s'esborra el seu contingut anterior. Per últim, es quedarà esperant, sense consumir CPU, a que el procés principal el desbloquegi, mitjançant la pipe amb nom, per poder continuar i mutar per executar el programa "parser" de l'exercici 1. Si no has pogut completar-ho amb èxit, et proporcionem un correctament implementat.

Per comprovar que el programa s'ha executat correctament, hauran de sortir tants missatges del programa "parser" (un per segon) i creat tants fitxers ".out" com diferents de fitxers diferents s'han introduït com paràmetres d'entrada.

Exemple de sortida:

```
$ ./multiparser  text1 text2 text3
Proces principal despertant proces child 1
Proces child 1 despertat
El text te 5 salts de línia i shan utilitzat 20 Bytes del heap
El proces child 1 ha finalitzat voluntàriament
Proces principal despertant proces child 2
Proces child 2 despertat
El text te 7 salts de línia i shan utilitzat 28 Bytes del heap
El proces child 2 ha finalitzat voluntariament
Proces principal despertant proces child 3
Proces child 3 despertat
El text te 18 salts de línia i shan utilitzat 72 Bytes del heap
El proces child 3 ha finalitzat voluntàriament
Final del proces principal
```

NOTA: En aquest exemple es generarien els fitxers de sortida "text1.out", "text2.out" i "text3.out". A més, hem afegit missatges per aclarir el comportament que han de tenir els processos.

Què s'ha de fer

- El Makefile
- Els codis dels programes en C
- La funció Usage() per a cada programa que sigui necessari

Què es valora

- Que es segueixin les especificacions de l'enunciat
- Que l'ús de les crides al sistema sigui correcte
- Que es comprovin els errors de totes les crides al sistema
- Que el codi sigui clar i correctament indentat
- Que el Makefile tingui ben definides les dependències i objectius
- Que la funció Usage() mostri per pantalla com s'ha d'invocar correctament el programa en cas que els arguments rebuts no siguin els adequats

Què cal lliurar

Un únic fitxer tar.gz amb el codi de tots els programes i el Makefile:

```
tar zcvf examenlab.tar.gz Makefile *.c
```