Makefile (0,5)

Crea un archivo *Makefile* para compilar todos los programas del examen, usando reglas independientes para cada uno. Incluye una regla llamada *clean* para eliminar todos los binarios, archivos objeto y/o ficheros temporales. Los programas deben compilarse si y sólo si se han actualizado los archivos de código fuente que los componen.

Control de errores y función Usage (0,75)

Todos los programas deben incluir un control adecuado de los argumentos usando la función usage y deben controlar los errores en TODAS las llamadas al sistema (a excepción del write por pantalla).

1: Formato entrada/salida (*1,75* puntos)

Escribe un programa en C llamado **arith.c** para realizar operaciones aritméticas sobre un conjunto de números enteros. El programa recibirá como argumento un número, *n*, que debe ser mayor que 0 y menor que 3, y leerá de la entrada estándar enteros (en formato binario). A cada número leído le aplicará una operación aritmética concreta y escribirá el resultado por la salida estándar (en formato binario). Este proceso se repetirá mientras haya valores en la entrada estándar. La operación aritmética a aplicar dependerá del valor de *n*:

- Si *n* es 1, restará 1 a cada número leído.
- Si *n* es 2, calculará 2^M para cada número leído M (puedes usar un desplazamiento de bits para hacer el cálculo (1<<M)).

Ejemplo de uso usando el fichero **entrada** que os proporcionamos, que incluye los primeros nueve números naturales en formato binario (123456789), y el programa **printints** que también os damos, que escribe por pantalla números enteros en formato texto:

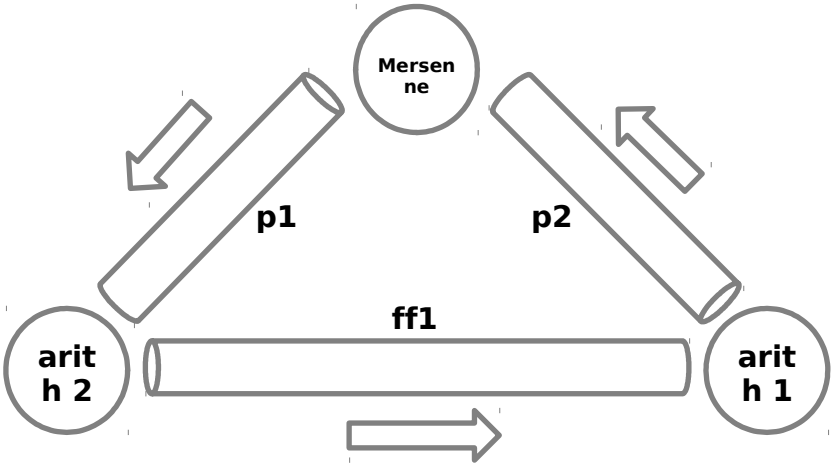
```
% ./arith 1 < entrada | ./printints
0 1 2 3 4 5 6 7 8

% ./arith 2 < entrada | ./printints
2 4 8 16 32 64 128 256 512
```

2: Pipes + gestión de ficheros (*3,75* puntos)

Escribe un programa en C llamado **Mersenne.c** que calcule los números de Mersenne que corresponden a un conjunto de números enteros. El número de Mersenne de un valor N se calcula como 2^N – 1. El programa recibirá como argumento el nombre de un fichero que contendrá el conjunto de números enteros (en formato binario) a calcular. Si el fichero no existe, el programa debe devolver un error. Para hacer los cálculos se crearán dos procesos, cuya

comunicación se tiene que hacer mediante varias pipes con nombre (*ff1*) y sin nombre (*p1 y p2*), tal como se detalla en la figura. El proceso inicial creará todas las pipes. La pipe con nombre se creará con permisos de lectura y escritura para el usuario y el grupo y ninguno para el resto, pero hay que asumir que pueden haber sido creadas previamente por otro programa. El proceso inicial leerá los enteros del fichero pasado como parámetro y los escribirá en la pipe *p1*. Cada proceso se encargará de una parte del cálculo para cada número, siguiendo el orden indicado. Para hacerlo, mutará al programa **arith** (pasando el argumento que corresponda) tras haber redireccionado su entrada y salida estándar a los canales correspondientes de las pipes. Al finalizar el cálculo, el proceso inicial leerá los resultados de la pipe *p2* y los escribirá en un fichero, llamado "salida". Si el fichero ya existía, se debe devolver un error. El fichero se debe crear con permisos de lectura y escritura para el usuario y ningún permiso para el grupo y el resto.




```
% ./Mersenne entrada
% cat salida | ./printints
1 3 7 15 31 63 127 255 511
```

3: Puntero L/E + gestión de ficheros (*3,25* puntos)

Escribe un programa en C llamado **primemap.c** que identifique números primos. El programa recibirá como parámetro el nombre de un fichero que contendrá el conjunto de números enteros (en formato binario) a evaluar. Si el fichero no existe, el programa debe devolver un error. El programa generará como resultado un fichero de texto con el mismo nombre que el fichero de entrada, pero añadiendo el sufijo "-prime-map". Si el fichero ya existía, se trunca el contenido. El fichero se debe crear con permisos de lectura y escritura para el usuario y el grupo y lectura para el resto. Para cada número entero del fichero de entrada, el fichero de salida contendrá un carácter indicando si este número es primo (carácter 'p') o no (carácter '-').

El programa debe crear tantos procesos como números enteros haya en el fichero de entrada. Puedes calcular la cantidad de hijos necesarios a partir del tamaño del fichero. Los procesos hijos deben ejecutarse de manera concurrente. Cada uno de ellos se encargará de comprobar un número entero, que vendrá determinado inversamente al orden en que se han creado. Así, el primer hijo creado comprobará el último entero del fichero, el segundo hijo comprobará el penúltimo entero y así sucesivamente. Para hacer la comprobación, cada proceso hijo abrirá el fichero, leerá el número entero que le toca y comprobará si es primo usando el programa **is_prime** (adjunto al enunciado) que recibe como parámetro un número. Para ello deberá crear otro proceso que mute a este programa, y recoger su código de finalización cuando éste acabe. Este código será 1 cuando el número recibido como parámetro sea primo, y 0 cuando no lo sea. Cuando sepa si el número es primo o no, el proceso escribirá el carácter que toque (‘p’ o ‘-‘) en la posición correspondiente del fichero de salida y acabará con un código de finalización igual al valor obtenido de la finalización del programa **is_prime**. Cuando todos los hijos hayan acabado, el proceso inicial calculará cuántos números primos hay en el fichero de salida (sumando los códigos de finalización de los procesos hijos) y escribirá el resultado al final del fichero de salida. Ejemplo de uso usando el fichero **salida** generado por el programa del apartado anterior:

```
%./primemap salida

% cat salida-prime-map

-pp-p-p-- 4
```

Se valorará en los programas

- Seguir las especificaciones de los ejercicios.
- El uso correcto de las llamadas al sistema.
- Control de errores en las llamadas al sistema.
- La claridad del código y una indentación adecuada.
- Que las reglas del Makefile tengan las dependencias y los objetivos requeridos correctos.
- La función Usage() muestra por pantalla, en caso de invocación incorrecta, la línea de comandos correcta para ejecutar el programa.

Qué se debe entregar

Un único fichero *tar.gz* con el código fuente y el Makefile. Para generarlo podéis usar el comando siguiente:

```
tar zcvf clab2.tar.gz *.c Makefile
```