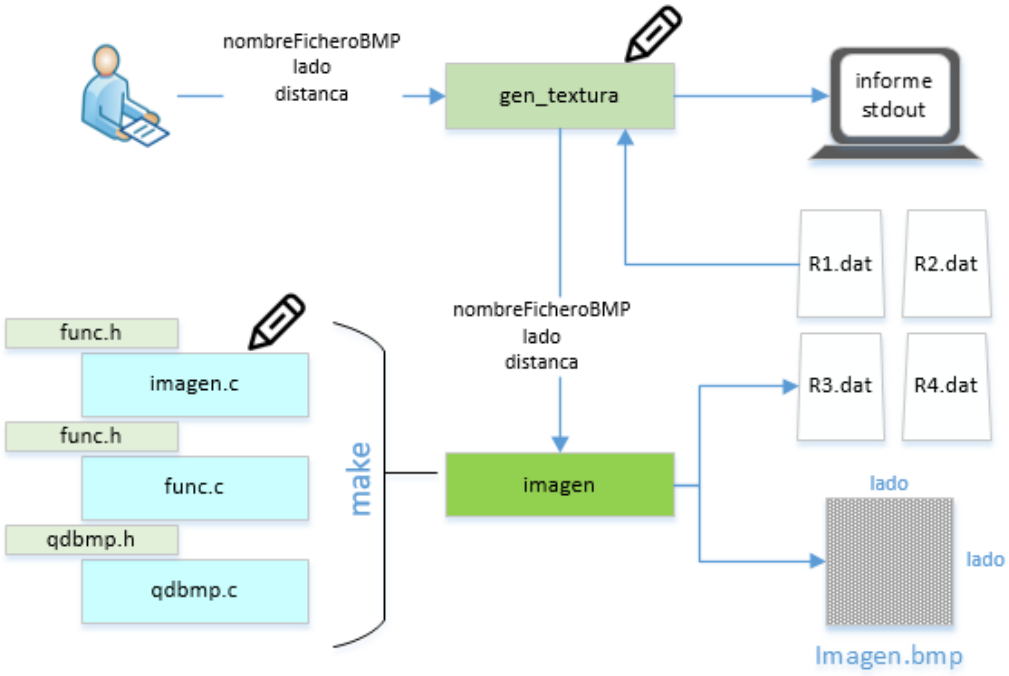
Sistemas Operativos

Examen Práctico
Convocatoria Ordinaria – 10 Enero 2019



Disponemos de un conjunto de ficheros en **C** con sus respectivas cabeceras **.h** que una vez compilados mediante **make** generan un ejecutable llamado **imagen**.

Este ejecutable genera por un lado una imagen **BMP** cuadrada de tamaño **lado x lado** y con una textura que está regulada por el parámeto **distancia**. El ejecutable **imagen** recibe estos parametros (lado y distancia) junto con el **nombre** que se va a dar al fichero .bmp. Por otro lado también genera cuatro ficheros de resultados **Rn.dat** con datos de la ejecución.

Para lanzar este ejecutable se dispone de un script **bash** llamado **gen_textura** que recibe los mismos parametros del usuario y que realiza el lanzamiento del ejecutable **imagen.** Cuando los ficheros de resultados están generados, los lee y compone un **informe** que muestra por **pantalla**.

Se pide:

Editar correctamente los esqueletos de ficheros **imagen.c** y **gen_textura** para que realicen esta tarea cumpliendo las siguientes especificaciones.

Para aprobar es necesario aprobar ambas partes por separado.

Parte C: 6 Puntos. Parte Bash: 4 Puntos

Especificaciones:

gen_textura:

- Debe realizar la comprobación de argumentos que viene comentada en el esqueleto del script.
- Si el fichero .bmp existe deberá preguntar al ususario si quiere sobreescribirlo, en cuyo caso lanza el ejecutable **imagen** con los parametros adecuados. Si no desea sobreescribirlo, simplemente termina. Si el fichero .bmp no existe lanza el ejecutable igualmente.
- Utilizar las variables y arrays definidos en el código, puesto que la función **ShowResults()** los utiliza para mostrar el resultado. Esta función no hay que modificarla.
- Las variables y arrays se deben rellenar al ir leyendo cada uno de los ficheros .dat generados por **imagen** (en total cuatro). Estos ficheros son generados por cuatro procesos hijos del proceso imagen y se numeran con el número de hijo, **Rn.dat**. Estos ficheros contienen un dato por linea, siendo estos datos los siguientes y en este orden:

  lado
  distancia
  carga
  ixIni
  ixEnd
  numPares
  numImpares
  numClaros
  numOscuros

donde: **carga** es el número de pixels de la imagen que le tocan a cada hijo
ixIni es el indice del array de pixels por el que ha comenzado a trabajar el hijo
ixEnd es el indice del array de pixels en el que ha terminado de trabajar el hijo
numPares es el número de pares que ha contado el hijo en su bloque de pixels
numImpares es el número de impares que ha contado el hijo en su bloque
numClaros es el número de pixeles claros (>=128) que ha contado en su bloque
numOscuros es el número de pixeles oscuros que ha contado en su bloque

imagen:

El proceso imagen generaraá una imagen **.bmp** y cuatro ficheros de resultados.

El proceso creará una imagen de tamaño **lado x lado** en un segmento de memoria compartida con tamaño **lado x lado** enteros. Este segmento no es más que un array consecutivo de enteros.

El proceso padre (imagen), creará 4 hijos para repartir el trabajo. A cada hijo se le asigna un conjunto de enteros del array, definido por los indices **ixIni** e **ixEnd**. Son bloques consecutivos de enteros.

Cada uno de los hijos a su vez generara 2 hijos, los nietos de padre. Estos nietos realizarán tareas específicas, uno contará pares/impares y el otro contará claros/oscuros. Estas cuentas las realizan sobre el bloque de enteros asignados a su padre (el hijo en curso).

- El proceso padre no realiza comprobación de argumentos, se su pone que vienen correctos pues gen_textura lo realiza.
- Ultilizar las variables definidas en el programa para realiza la tarea, si se quiere se pueden definir variables adicionales para trabajar.
- La comunicación entre padre e hijos y entre hijos y nietos se realizará mediante pipes. Para comunicar padre e hijo crear y utilizar el pipe **hijosPipe** y para comunicar un hijo con los nietos crear y utilizar **nietosPipe**. No es necesario cerrar los descriptores no usados.
- Para escribir y leer de los pipes existen funciones ya desarrolladas al efecto, podeis usarlas o desarrollar la lectura/escritura del pipe vosotros.
- El proceso padre creará 4 hijos sin esperar a que los hijos terminen para crear al siguiente. Deberá controlar los posibles errores en la creación de los hijos mediante fork. Deberá esperar la terminación de los hijos para continuar y terminar.
- Cada hijo habrá escrito información en el **pipeHijos**, esta será leida por el padre en el array **arrayDatos**. Este array será pasado a la función (ya desarrollada) **GenerarFicheroResultados** junto con el resto de parametros necesarios. Esta función genera el fichero **Rn.dat** para cada hijo, siendo **n** el número de orden de cada hijo.
- Una vez leidos todos los **arrayDatos** (uno por cada hijo) el padre esperará (o capturará) a la terminación de los hijos usando **wait** o **waitpid**. No se realiza nada con el estado de terminación de los hijos.
- Tras la correcta terminación de los hijos el padre llamará a la función (ya realizada) **SaveAsBMP** pasándole el array de memoria compartida y el resto de parametros para generar la imagen .bmp. *Esta imagen puede ser vista haciendo doble click sobre ella en el explorador de archivos.*
- Respecto al segmento de memoria compartida, el padre, lo creaará, se vinculará, desvinculará y eliminará correctamente donde proceda.
- Respecto a los indices para que cada hijo tome el conjunto de enteros apropiado, estos se basan en la variable **carga** que ya está calculada al inicio del código del padre. Su valor depende del valor de **carga** y del número de orden del hijo. Este número de orden puede ser un iterador **i** del bucle de creación de los hijos). Su valor se calcula asi:

```
ixIni = i * carga;
ixEnd = i * carga + (carga-1);
```

- El proceso padre llamará a la función **CodigoHijo(ixIni,ixEnd)** que realizará la tarea de los hijos.
- Los hijos tendrán su código en la función **CodigoHijo** que deberá también ser completada.
- Los hijos, tras las inicializaciones y la creación del **nietosPipe**, deberán crear dos nietos dada uno. Cada nieto llamará a una función para realizar su tarea, uno a **ContarPares** y otro a **ContarClaros**. Estas funciones ya están desarrolladas. Se les pasa los indices y un **arrayCuentas** donde colocarán las cuentas que realicen.

- Al igual que el padre, los hijos comprobarán posibles errores en la llamada a fork.
- Cada nieto escribirá el **arrayCuentas** en el **nietosPipe** para comunicarselo a su padre (cada hijo)
- Al igual que el padre, los hijos deberán esperar/capturar la terminación de sus hijos (los nietos)
- Tras la terminación de los nietos, los hijos leerán del **nietosPipe** los arrays de cuentas de cada uno de los nietos y compondrán el **arrayDatos** que cada hijo enviará al padre. El codigo que lee del **nietosPipe** y que compone el **arrayDatos** ya está desarrollado.
- Cada hijo deberá no obstante escribir su **arrayDatos** en el **hijosPipe** para que sea leido por el padre.
- Los hijos están vinculados automáticamente al segmento de memoria compartida por estarlo su padre, pero deberán desvincularse antes de terminar.

El siguiente esquema muestra la jerarquía padre/hijos/nietos y los canales de comunicación entre ellos y con el segmento de memoria compartida.