Administración y Organización de Computadores

Curso 2016-2017

<u>Práctica</u>: *Procesamiento de imágenes en ensamblador*

El principal objetivo de esta práctica es completar una aplicación de procesamiento de imágenes, escrita en *C++* sobre Linux, a través de la implementación en ensamblador de las diferentes operaciones que debe proporcionar el programa. La integración de los dos lenguajes se llevará a cabo mediante las facilidades de ensamblado en línea proporcionadas por el compilador *gcc*. La programación en ensamblador se realizará considerando la arquitectura de un procesador Intel o compatible de 32 bits.

A continuación, se detallan diferentes aspectos a tener en cuenta para el desarrollo de esta práctica.

<u>Descripción de la aplicación</u>

El código que deberá ser completado está incluido en una aplicación C++, disponible como proyecto de Qt. La siguiente figura muestra una captura de pantalla en un instante de ejecución de la aplicación:

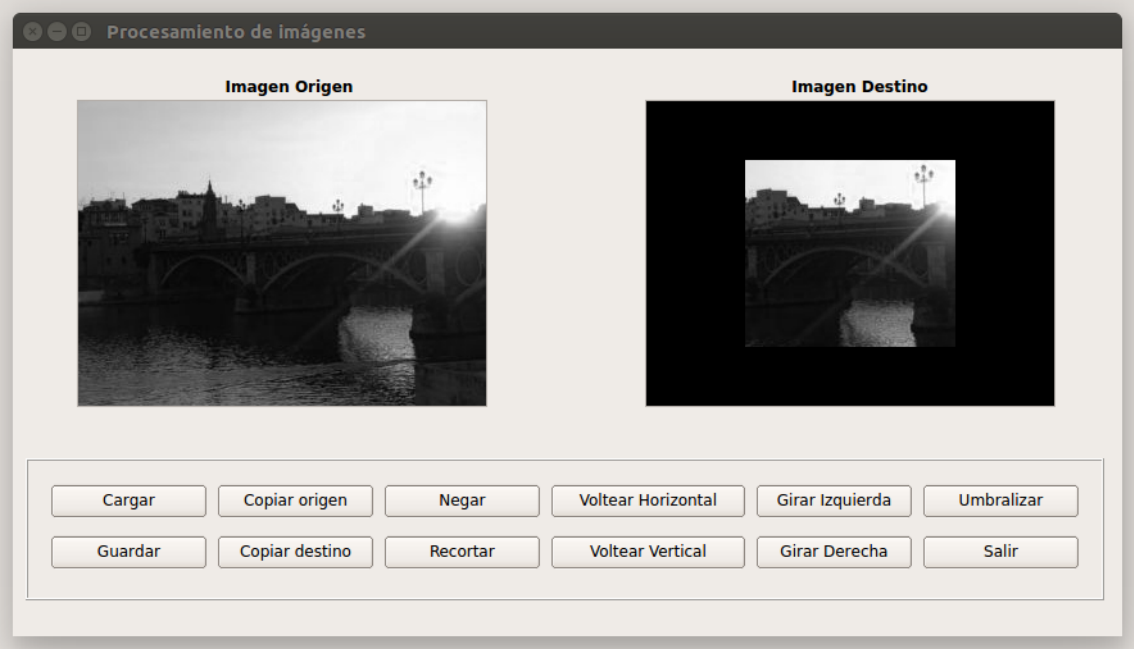

Figura 1: interfaz principal de la aplicación

¿Qué es Qt?

Qt es un marco de desarrollo de aplicaciones que, entre otras aportaciones, proporciona herramientas y librerías de clases para la creación de interfaces de usuario en entornos de escritorio.

Puesta en marcha del proyecto

El fichero que contiene la descripción del proyecto (*pracaoc.pro*) se encuentra disponible en la carpeta principal de la aplicación. Dicho fichero puede ser utilizado para importar el proyecto desde diferentes entornos de desarrollo. No obstante, se recomienda trabajar con Qt Creator (paquete *qtcreator*). Es necesario instalar además los paquetes *libqt4-dev* y *qt4-dev-tools*, en sistemas de 32 bits, o *libqt4-dev:i386, qt4-dev-tools:i386, gcc-multilib* y *g++-multilib* en sistemas de 64 bits.

Funcionamiento de la aplicación

La interfaz principal (figura 1) está compuesta por dos ventanas de imagen y una serie de botones a través de los cuales es posible ejecutar las diferentes operaciones que proporciona la aplicación. Las dos ventanas de imagen se corresponden con la imagen origen (ventana de la izquierda) y la imagen destino (ventana de la derecha). La imagen origen es la imagen sobre la que se realizan la mayoría de las operaciones de procesamiento. El resultado de cada una de estas operaciones se almacena en la imagen destino. Las imágenes procesadas por el programa tienen una resolución de 320x240 (320 columnas y 240 filas) y cada píxel está representado por un nivel de gris (0-negro, 255-blanco). Se describen a continuación las opciones incluidas en la aplicación:

- Cargar: carga, en la imagen origen, la imagen del fichero indicado. Si la imagen del fichero tiene una resolución superior a la soportada por el programa, ésta es recortada. Además, es posible cargar imágenes en color. En tal caso, esta opción se encarga de transformar la imagen a escala de grises.
- Guardar: guarda, en el fichero indicado, la imagen destino.
- Copiar origen: copia la imagen origen en la imagen destino.
- Copiar destino: copia la imagen destino en la imagen origen.
- Negar: El nivel de gris de cada píxel de la imagen origen es invertido (255 – nivel de gris) y almacenado en la imagen destino.

imagen origen

imagen destino

- Recortar: copia el trozo seleccionado de la imagen origen en la imagen destino. Al activarse la opción, el programa entra en modo de selección (el icono del puntero del ratón se transforma en una cruz al situarse sobre la imagen origen). En ese instante, el usuario selecciona con el ratón la porción de la imagen origen que deberá copiarse en la imagen destino. La ventana de imagen seleccionada se copiará centrada en la imagen

destino. Suponiendo que la ventana seleccionada comienza en la posición (fI, cI) y tiene un tamaño de WxH, cada píxel de la ventana, situado en la posición (f, c), se copiará en la imagen destino siguiendo la siguiente transformación:
imgD[f-fI+(240-H)/2, c-cI+(320-W)/2] = imgO[f, c]

El desplazamiento ((240-H)/2, (320-W)/2) permite centrar la ventana en la imagen destino.


imagen origen


imagen destino

- Voltear horizontal: voltea en horizontal la imagen origen y almacena el resultado en la imagen destino:

  imgD[f, c]=imgO[f, 320-c]


imagen origen


imagen destino

- Voltear vertical: voltea en vertical la imagen origen y almacena el resultado en el imagen destino:

  imgD[f, c]=imgO[240-f, c]

imagen origen

imagen destino

- Girar izquierda: gira 90º hacia la izquierda la imagen origen y almacena el resultado en la imagen destino. El eje de giro estará situado en el punto central de la imagen, por lo que, suponiendo un rango en fR de [-120, 120] y en cR de [-160, 160], la transformación de cada píxel sería la siguiente:

  imgD[fR+120, cR+160] = imgO[cR+120,-fR+160]

  Puesto que las imágenes no son cuadradas, no todos los píxeles de la imagen destino tienen un píxel asociado en la imagen origen. El proceso de giro en estos puntos asignará un valor 0 (negro).

imagen origen

imagen destino

- Girar derecha: gira 90º hacia la derecha la imagen origen y almacena el resultado en la imagen destino. El eje de giro estará situado en el punto central de la imagen, por lo que, suponiendo un rango en fR de [-120, 120] y en cR de [-160, 160], la transformación de cada píxel sería la siguiente:

  imgD[fR+120, cR+160] = imgO[-cR+120, fR+160]

  Como en la transformación anterior, para aquellos píxeles que no tengan un píxel asociado en la imagen origen, el proceso asignará un valor 0.

imagen origen

imagen destino

- Umbralizar: la umbralización es una técnica de segmentación utilizada para separar en una imagen los objetos del fondo. El umbral se establece de manera automática utilizando el método Otsu. Los píxeles de la imagen origen con valor inferior al umbral se ponen a 0 (negro) en la imagen destino y aquellos que superan el umbral tomarán el valor 255 (blanco). Para calcular el umbral de manera automática, el método Otsu utiliza el histograma de la imagen. El histograma de una imagen representa la frecuencia de aparición de cada nivel de gris. Se trata de un vector de 256 entradas, una por cada nivel de gris. Cada elemento $h[i]$ contiene el número de píxeles de imagen con nivel de gris i.

  A partir del histograma de la imagen origen, el umbral se obtiene como el nivel de gris que "*mejor*" permite separar los píxeles de la imagen en 2 clases (fondo y objeto). Se demuestra que dicho umbral es aquel que permite maximizar la dispersión entre las dos clases. La dispersión entre 2 clases se define como:

$$\sigma^2 = \frac{n_1\left(\mu_1 - \mu_T\right)^2 + n_2\left(\mu_2 - \mu_T\right)^2}{n_T}$$

  siendo $n1$ y $n2$ el número de píxels de cada clase $\mu 1$ y $\mu 2$ el nivel de gris medio de cada clase, μT el nivel de gris medio de toda la imagen y nT el número total de píxels. Para un determinado valor de umbral k, las medias $\mu 1$ y $\mu 2$ se calculan como:

$$\mu_1 = \frac{\sum_{i=0}^{k} h[i] * i}{n_1} \qquad \mu_2 = \frac{\sum_{i=k+1}^{255} h[i] * i}{n_2}$$

  donde $n1$ y $n2$ se obtienen como:

$$n_1 = \sum_{i=0}^{k} h[i] \qquad n_2 = \sum_{i=k+1}^{255} h[i]$$

imagen origen

imagen destino

Componentes de la aplicación

El código fuente de la aplicación está formado por 5 ficheros: *main.cpp*, *pracaoc.cpp*, *pracaoc.h*, *imageprocess.cpp* e *imageprocess.h*. Además, se incluyen un formulario de Qt (*mainForm.ui*) y el fichero de descripción del proyecto (*pracaoc.pro*). El contenido de cada fichero fuente es el siguiente:

- *main.cpp*: contiene el procedimiento principal que permite lanzar la aplicación, así como crear y mostrar la ventana principal que actúa como interfaz entre el usuario y la aplicación.
- *pracaoc.h*: fichero que contiene la definición de la clase principal de la aplicación (*pracAOC*). Esta clase contiene los elementos principales de gestión de la aplicación. Entre los atributos, se encuentran las definiciones de las interfaces de usuario incluidas en el programa, así como de las variables que permiten almacenar la información de las imágenes origen y destino utilizadas en las opciones de procesamiento de la aplicación. Estas variables son *imgO* e *imgD*. Ambas están definidas como un array de tipo uchar (unsigned char) de 320x240 elementos. La imagen dentro de cada uno de estos arrays se encuentra almacenada por filas. Esto implica que el acceso a un determinado píxel situado en una fila *f* y una columna *c* de imagen se realiza a través de la posición del array *f*320+c*.
- *pracaoc.cpp*: Incluye la implementación de los métodos de la clase *pracAOC*. En su mayoría, estos métodos se encargan de responder a los distintos eventos de la interfaz de usuario incluida en el programa y de llamar a las funciones de procesamiento de imagen que correspondan en cada caso (disponibles en *imageprocess.cpp* e *imageprocess.h*).
- *imageprocess.h*: contiene la definición de las funciones implementadas en el fichero *imageprocess.cpp*.
- *imageprocess.cpp*: implementación de las funciones de procesamiento de imagen que se ejecutan a través de las distintas opciones de la aplicación. La mayoría de estas funciones contienen una implementación vacía. El objetivo de esta práctica es completarlas para que el funcionamiento de la aplicación sea el descrito anteriormente.

Dentro del fichero *imageprocess.cpp*, se ha incluido la implementación de la primera función (*copiar*) a modo de ejemplo. Dicha implementación se detalla a continuación. La función *copiar* es la única función de *imageprocess.cpp* que se encuentra implementada. Como ya se ha comentado anteriormente, la implementación de las restantes es objeto de esta práctica.

```
void imageprocess::copiar(uchar * imgO, uchar * imgD)
{
        asm volatile(
                "movl %0, %%esi \n\t"
                "movl %1, %%edi \n\t"
```

```
            "movl $320*240, %%ecx \n\t"
        "bcopia: \n\t"
            "movb (%%esi), %%al \n\t"
            "movb %%al, (%%edi) \n\t"
            "incl %%esi \n\t"
            "incl %%edi \n\t"
        "loop bcopia \n\t"
            :
            : "m" (imgO), "m" (imgD)
            : "%eax", "%ecx", "%esi", "%edi", "memory"

        );
}
```

La función *copiar* es invocada en la ejecución de las opciones “Copiar Origen” y “Copiar Destino” para copiar la imagen origen en la destino y viceversa. Esta función incluye como parámetros un array que contiene la imagen a copiar (*imgO*) y un segundo array que especifica la imagen donde se debe realizar la copia (*imgD*). Ambos parámetros son tratados como operandos de entrada del bloque de código en ensamblador (la lista de operandos de salida está vacía). El motivo para que esto sea así es que en ningún caso se van a modificar estos parámetros, puesto que su contenido es la dirección de comienzo de los bloques de memoria donde se encuentran las dos imágenes. Lo que sí se va a modificar es el contenido del bloque de memoria apuntado por *imgD*, pero esto no afecta a la dirección almacenada en dicho parámetro.

Tras la definición de los operandos, se incluye la lista de registros utilizados dentro del código. Además de los registros indicados, dado que la memoria es modificada, la lista incluye también la palabra “memory”.

Una vez aclaradas estas definiciones, analicemos a continuación paso a paso el código ensamblador incluido en el procedimiento:

- Las dos primeras instrucciones se encargan de copiar las direcciones iniciales de memoria de las dos imágenes, indicadas por los dos operandos (%0=imgO, %1=imgD), en dos registros, *esi* y *edi*, para su posterior direccionamiento. Así, a través del registro *esi* podremos acceder a cada uno de los píxels de *imgO* y, mediante el registro *edi*, tendremos acceso a los píxeles de *imgD*.
- La copia de cada píxel de *imgO* en la imagen *imgD* se realiza mediante un bucle con tantas iteraciones como indica el tamaño de las imágenes (320x240). Este bucle se lleva a cabo mediante la instrucción *loop*, por lo que, lo siguiente que hace el procedimiento es inicializar el registro *ecx* con el total de iteraciones.
- Una vez inicializado *ecx*, comienza el bucle de copia. Cada iteración consiste en la copia del píxel actual de *imgO* – (*%%esi*) – en el píxel correspondiente de *imgD* – (*%%edi*) –. Dicha copia se lleva a cabo a través del registro *al*, puesto que no es posible indicar los dos elementos de memoria como operandos de la instrucción *mov*. Tras esta operación y antes de pasar a la siguiente iteración del bucle, los índices *esi* y *edi* son incrementados para que apunten a los siguientes elementos de *imgO* e *imgD*.

Especificación de los objetivos de la práctica

El principal objetivo de esta práctica es completar el código de la aplicación descrita para que su funcionamiento sea el que se detalla en la sección “*Funcionamiento de la aplicación”*, incluida es esta documentación. Para ello, se deberán implementar, en lenguaje ensamblador, los

procedimientos “vacíos” del módulo “imageprocess.cpp”. Estos procedimientos están asociados con las opciones del programa. Para cada uno de ellos, se proporciona la estructura inicial del bloque ensamblador, en la que se ha incluido la definición de operandos que afectan a la implementación. La lista de registros utilizados incluye únicamente la palabra “memory”, ya que en todos los casos la memoria es modificada. La inclusión de registros dentro de esta lista dependerá de la implementación que se desarrolle en cada caso, por lo que, será necesario completarla para cada uno de los procedimientos.

Se describe a continuación la funcionalidad de cada uno de los procedimientos a completar. Para todos ellos, los parámetros *imgO* e *imgD* contienen, respectivamente, la dirección de los bloques de memoria que almacenan el contenido de las imágenes origen y destino. Para una descripción más detallada de estas funciones, se recomienda revisar la sección “*Funcionamiento de la aplicación”*.

- *void imageprocess::**negar**(uchar * **imgO**, uchar * **imgD**)*: invierte el nivel de gris de cada píxel de *imgO* y devuelve el resultado en *imgD*.
- *void imageprocess::**recortar**(uchar * **imgO**, uchar * **imgD**, int c**I**, int f**I**, int **w**, int **h**)*: copia, centrada en *imgD*, la ventana de *imgO* indicada por *fI*, *cI*, *h* y *w*. Los parámetros *fI* y *cI* son, respectivamente, la fila y la columna de la esquina superior izquierda de la ventana a copiar. El alto y el ancho en píxeles de la ventana viene especificado por *h* y *w*.
- *void imageprocess::**voltearHorizontal**(uchar * **imgO**, uchar * **imgD**)*: voltea, en el eje horizontal, la imagen indicada por *imgO* y almacena el resultado en *imgD*.
- *void imageprocess::**voltearVertical**(uchar * **imgO**, uchar * **imgD**)*: voltea, en el eje vertical, la imagen indicada por *imgO* y almacena el resultado en *imgD*.
- *void imageprocess::**girarIzquierda**(uchar * **imgO**, uchar * **imgD**)*: gira 90º hacia la izquierda la imagen indicada por *imgO* y almacena el resultado en *imgD*.
- *void imageprocess::**girarDerecha**(uchar * **imgO**, uchar * **imgD**)*: gira 90º hacia la derecha la imagen indicada por *imgO* y almacena el resultado en *imgD*.
- *void imageprocess::**calcularHistograma**(uchar * **imgO**, int * **histoO**)*: calcula el histograma de la imagen indicada en *imgO* y lo almacena en *histoO*. Este procedimiento es invocado al ejecutar la opción “*Umbralizar*” para obtener el histograma de imagen sobre el que calcular el umbral con el método Otsu.
- *void imageprocess::**calcularUmbral**(int * **histoO**, uchar & **umbral**)*: calcula el umbral Otsu a partir del histograma indicado en *histoO*. Devuelve el resultado en *umbral*. Este procedimiento es invocado cuando el usuario ejecuta la opción “*Umbralizar*” para calcular el umbral que será aplicado a la imagen mediante el procedimiento *“imageprocess::umbralizar”*.
- *void imageprocess::**umbralizar**(uchar * **imgO,** uchar **umbral,** uchar * **imgD**)*: aplica el umbral indicado a la imagen almacenada en imgO y devuelve el resultado en imgD. Este procedimiento es invocado al seleccionar la opción “*Umbralizar*” para aplicar el umbral obtenido por el procedimiento anterior.

La implementación de los tres últimos procedimientos relacionados con la opción “*Umbralizar*”será opcional. La nota máxima que podrá obtenerse sin la realización de dicho procedimiento será de Notable(8).

Nota: la práctica se realizará de manera individual.
Fecha de entrega de la práctica: La entrega se realizará en enero de 2017. La fecha concreta se comunicará una vez que se publique el calendario de exámenes de la convocatoria de enero.
Entrega: la entrega se realizará a través de la subida de un archivo .zip, que contenga el proyecto completo, al aula virtual de la asignatura.