C:

cuadrados\_4.c:

/\*

Programa: cuadrados\_4.c

Descripción: Genera cuatro rectangulos anidados bitmap en formato BMP y guarda la imagen en el fichero test.bmp.

Compilación: gcc --32 -o cuadrados\_4 cuadrados\_4.c

Ejecución: $./cuadrados\_4

Visualización: comando "display" -> $display test.bmp -> salir con Ctrl-C

\*/

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#define top 512

#define xcoor top/8

#define ycoor top/8

// Definición de tipos

typedef int LONG;

typedef unsigned char BYTE; // tipo de cada byte de cada pixel. Rango de valores entre 0x00 y 0xFF. Indican intensidad de cada color RGB

typedef unsigned int DWORD;

typedef unsigned short WORD;

// definición del tipo de la cabecera FILE del fichero BMP

typedef struct tagBITMAPFILEHEADER

{

WORD bfType; // 2 /\* Magic identifier \*/

DWORD bfSize; // 4 /\* File size in bytes \*/

WORD bfReserved1; // 2

WORD bfReserved2; // 2

DWORD bfOffBits; // 4 /\* Offset to image data, bytes \*/

} \_\_attribute\_\_((packed)) BITMAPFILEHEADER;

// definición del tipo de la cabecera INFO del fichero BMP

typedef struct tagBITMAPINFOHEADER

{

DWORD biSize; // 4 /\* Header size in bytes \*/

LONG biWidth; // 4 /\* Width of image \*/

LONG biHeight; // 4 /\* Height of image \*/

WORD biPlanes; // 2 /\* Number of colour planes \*/

WORD biBitCount; // 2 /\* Bits per pixel \*/

DWORD biCompress; // 4 /\* Compression type \*/

DWORD biSizeImage; // 4 /\* Image size in bytes \*/

LONG biXPelsPerMeter; // 4

LONG biYPelsPerMeter; // 4 /\* Pixels per meter \*/

DWORD biClrUsed; // 4 /\* Number of colours \*/

DWORD biClrImportant; // 4 /\* Important colours \*/

} \_\_attribute\_\_((packed)) BITMAPINFOHEADER;

// definición del tipo de cada pixel. Cada pixel son tres bytes . Cada byte es tipo BYTE.

typedef struct

{

BYTE b;

BYTE g;

BYTE r;

} RGB\_data;

int bmp\_generator(char \*filename, int width, int height, unsigned char \*data)

{

BITMAPFILEHEADER bmp\_head;

BITMAPINFOHEADER bmp\_info;

int size = width \* height \* 3;

bmp\_head.bfType = 0x4D42; // 'BM'

bmp\_head.bfSize= size + sizeof(BITMAPFILEHEADER) + sizeof(BITMAPINFOHEADER); // 24 + head + info no quad

bmp\_head.bfReserved1 = bmp\_head.bfReserved2 = 0;

bmp\_head.bfOffBits = bmp\_head.bfSize - size;

// finish the initial of head

bmp\_info.biSize = 40;

bmp\_info.biWidth = width;

bmp\_info.biHeight = height;

bmp\_info.biPlanes = 1;

bmp\_info.biBitCount = 24; // bit(s) per pixel, 24 is true color

bmp\_info.biCompress = 0;

bmp\_info.biSizeImage = size;

bmp\_info.biXPelsPerMeter = 0;

bmp\_info.biYPelsPerMeter = 0;

bmp\_info.biClrUsed = 0 ;

bmp\_info.biClrImportant = 0;

// finish the initial of infohead;

// copy the data

FILE \*fp; // declaro stream pointer

if (!(fp = fopen(filename,"wb"))) return 0; // inicializo stream pointer abriendo el fichero

fwrite(&bmp\_head, 1, sizeof(BITMAPFILEHEADER), fp); //escritura la cabecera FILE

fwrite(&bmp\_info, 1, sizeof(BITMAPINFOHEADER), fp); //escritura la cabecera INFO

//escritura de pixels

fwrite(data, 1, size, fp);// 1ºarg: buffer a escribir , 2ºarg: long en bytes de cada elemento, 3ºarg:nº de elementos, 4ºarg: stream pointer

fclose(fp);

return 1;

}

// Función de entrada

int main(int argc, char \*\*argv)

{

int i,j;

RGB\_data buffer[top][top]; // Array de pixels

memset(buffer, 0, sizeof(buffer)); // inicializa el buffer con el valor cero. Librería libc.

// Los colores se expresan en formato RGB donde la intensidad de cada color se codifica con un byte

// 0x00: ausencia de color ; 0xFF: intensidad máxima

// La ausencia de los tres colores primarios R=G=B=0x00 es el negro

// R=G=B=0xFF es el blanco

//va generando cada vez cuadrados con menos área restando un octavo por cada lado

for (i = xcoor; ((xcoor <= i) && (i < (top-xcoor))); i++)

{

for (j = ycoor; ((ycoor <= j) && (j < (top-ycoor))); j++)

{

// intensidad de rojo

buffer[i][j].r = 0xff;

// intensidad de verde

buffer[i][j].g = 0x00;

// intensidad de azul

buffer[i][j].b = 0xff;

}

}

for (i = 2\*xcoor; ((2\*xcoor <= i) && (i < (top-2\*xcoor))); i++)

{

for (j = 2\*ycoor; ((2\*ycoor <= j) && (j < (top-2\*ycoor))); j++)

{

// intensidad de rojo

buffer[i][j].r = 0x00;

// intensidad de verde

buffer[i][j].g = 0xff;

// intensidad de azul

buffer[i][j].b = 0xff;

}

}

for (i = 3\*xcoor; ((3\*xcoor <= i) && (i < (top-3\*xcoor))); i++)

{

for (j = 3\*ycoor; ((3\*ycoor <= j) && (j < (top-3\*ycoor))); j++)

{

// intensidad de rojo

buffer[i][j].r = 0xFF;

// intensidad de verde

buffer[i][j].g = 0xff;

// intensidad de azul

buffer[i][j].b = 0x00;

}

}

bmp\_generator("./test.bmp", top, top, (BYTE\*)buffer);

return EXIT\_SUCCESS;

}

comp\_ejec\_vis.sh:

#! /bin/bash

make #compila bmp\_imagen

./bmp\_imagen #ejecuta bmp\_imagen

display test.bmp & #muestra la imagen que se ha generado en test.bmp con el comando

#anterior

makefile:

CC=gcc

CFLAGS=-Wall -g -m32

bmp\_imagen: bmp\_imagen.c

$(CC) $(CFLAGS) -o bmp\_imagen bmp\_imagen.c #compila bmp\_imagen.c

clean:

$(RM) test.bmp bmp\_imagen #borra los modulos ejecutables test.bmp

#y bmp\_imagen

ENSAMBLADOR:

pixels.c:

/\*

Programa: array\_pixels\_2.s

Compilación: as --32 -gstabs -o array\_pixels\_2.o array\_pixels\_2.s

Descripción: Generar un array de 2D (coordenadas de los pixels ) para almacenar una estructura de datos RGB (3ª dimensión) e inicializarlo.

Metodogía: Desarrollar el algoritmo en lenguaje C

Relacionar las estructuras de C con las de Ensamblador:

Primero generamos la segunda dimensión

Después generamos la primera dimensión con un array de direcciones de memoria sin inicializar:

A continuación inicializamos el array de punteros " " con las direcciones de los pixels

\*/

## variables globales

## .global buffer ## buffer es o una variable global o un argumento

## Sección Macros

.equ DIMENSION\_Z, 3 #tamaño de la tercera dimensión: estructura pixel

## El origen de coordenadas está en la esquina inferior izquierda.

## El contenido del buffer comienza por el origen de coordenadas y contiene filas secuenciales

## La estructura del pixel sigue la seguencia B-G-R

## Sección variables scope fichero inicializadas

.section .data

rojo: .byte 0

verde: .byte 0

azul: .byte 0

origen\_x: .4byte 0 ## Origen coordenada X del cuadrado en pixels

origen\_y: .4byte 0 ## Origen coordenada Y del cuadrado en pisels

lado: .4byte 0 ## Lado del cuadrado en pixels

proporcion: .4byte 0 ## tanto por cien de los tres colores primarios RGB

dimension: .4byte 0 ## Dimension del cuadrado background en pixels

buffer\_ptr: .4byte 0 ## variable puntero donde guardaré el argumento referencia que le pasa main desde C

fin\_x: .4byte 0 ## Limite coordenada X del cuadrado en pixels=origen\_x+lado-1

fin\_y: .4byte 0 ## Limite coordenada Y del cuadrado en pixels=origen\_y+lado-1

## Sección Instrucciones

.section .text

.global pixels\_generator\_2

## .global buffer

.type pixels\_generator\_2, @function

pixels\_generator\_2:

/\*

Correspondencia tuple 3D (Fila x\_coor ,Columna y\_coor, Offset RGB) -> buffer lineal

Coordenada Filas :(0,Nx-1)

Coordenada Columnas :(0,Ny-1)

Offset RGB (0,1,2)

Buffer (direcciona bytes):

Elemento del buffer: cada elemento son 3 bytes.

Espacio de elementos - Espacio bytes: la dirección de un elemento es el byte AZUL de dicho elemento.

1º elemento de cada fila i : 1Fi -> x\_coor \* DIMENSION\_Y \* DIMENSION\_Z

1º elemento de cada columna j : 1Cj -> y\_coor \* DIMENSION\_Z

Posición del color: PC (B->0, G->1, R->2)

Función correspondencia del pixel (i,j) con el elemento (Offset Pixel (i,j)) en el buffer: OPij -> 1Fi + 1Cj

Ejemplo: Array XY 512x512, al pixel (3,3) le corresponde el siguiente Offset:

1F=3\*512\*3=4608

1C=3\*3 =9

OP= 4608+9=4617

\*/

## Nuevo frame en la pila

push %ebp

mov %esp, %ebp

#################################################

### SALVO EL CONTEXTO ANTERIOR A LA SUBRUTINA ###

push %eax

push %ebx

push %ecx

push %edx

push %edi

push %esi

#################################################

## Capturo el argumento origen\_x

mov 8(%ebp),%eax

mov %eax,origen\_x

## Capturo el argumento origen\_y

mov 12(%ebp),%eax

mov %eax,origen\_y

## Capturo el argumento lado

mov 16(%ebp),%eax

mov %eax,lado

## Capturo el argumento proporcion

mov 20(%ebp),%eax

mov %eax,proporcion

## Capturo el argumento dimension

mov 24(%ebp),%eax

mov %eax,dimension

## Capturo el argumento buffer

mov 28(%ebp),%eax

mov %eax,buffer\_ptr #inicializo el puntero con el argumento

## intensidad colores: color INICIAL

## color background ubuntu RGB is (48, 10, 36).

movb proporcion,%al

movb $0x00,rojo

movb $0xff,verde

movb $0x00,azul

## Cálculo de los limites

mov origen\_x,%eax

add lado,%eax

dec %eax

mov %eax,fin\_x

mov origen\_y,%eax

add lado,%eax

dec %eax

mov %eax,fin\_y

## Bucle de inicialización del array

mov origen\_x,%edi #inicio Filas

fila:

mov origen\_y,%esi #inicio Columnas

columna:

## control columna

cmp fin\_y,%esi

jz col\_exit

## actualización posición columna

inc %esi

## Reset registros aritmética

xor %eax,%eax

xor %ebx,%ebx

xor %ecx,%ecx

## Aritmética Fila

movw %di,%bx #posición fila

imul dimension,%ebx

imul $DIMENSION\_Z,%ebx

## Aritmética Columna

movw %si,%cx #posición columna

imul $DIMENSION\_Z,%ecx

## Correspondencia array\_pixel -> posición buffer

xor %eax,%eax

add %ebx,%eax

add %ecx,%eax

mov %eax,%ebx #EBX contiene el offset del pixel en el buffer

## Actualizar colores en el pixel

xor %ecx,%ecx #índice color

movb azul,%al #intensidad azul

mov buffer\_ptr,%edx

lea (%edx,%ebx),%edx

mov %eax,(%edx,%ecx) # dirección efectiva = M[buffer\_ptr] + offset + posi\_color

inc %ecx

movb verde,%al #intensidad verde

mov buffer\_ptr,%edx

lea (%edx,%ebx),%edx

mov %eax,(%edx,%ecx) # dirección efectiva = M[buffer\_ptr] + offset + posi\_color

inc %ecx

movb rojo,%al #intensidad rojo

mov buffer\_ptr,%edx

lea (%edx,%ebx),%edx

mov %eax,(%edx,%ecx) # dirección efectiva = M[buffer\_ptr] + offset + posi\_color

## siguiente columna

jmp columna

## actualización posición fila

col\_exit:

cmp fin\_x,%edi

jz fil\_exit

##siguiente fila

inc %edi

## Reset registros aritmética

xor %eax,%eax

xor %ebx,%ebx

xor %ecx,%ecx

## Aritmética Fila

movw %di,%bx #posición fila

imul dimension,%ebx

imul $DIMENSION\_Z,%ebx

## Aritmética Columna

movw %si,%cx #posición columna

imul $DIMENSION\_Z,%ecx

## Correspondencia array\_pixel -> posición buffer

xor %eax,%eax

add %ebx,%eax

add %ecx,%eax

mov %eax,%ebx #EBX contiene el offset del pixel en el buffer

## Actualizar colores en el pixel

xor %ecx,%ecx #índice color

movb azul,%al #intensidad azul

mov buffer\_ptr,%edx

lea (%edx,%ebx),%edx

mov %eax,(%edx,%ecx) # dirección efectiva = M[buffer\_ptr] + offset + posi\_color

inc %ecx

movb verde,%al #intensidad verde

mov buffer\_ptr,%edx

lea (%edx,%ebx),%edx

mov %eax,(%edx,%ecx) # dirección efectiva = M[buffer\_ptr] + offset + posi\_color

inc %ecx

movb rojo,%al #intensidad rojo

mov buffer\_ptr,%edx

lea (%edx,%ebx),%edx

mov %eax,(%edx,%ecx) # dirección efectiva = M[buffer\_ptr] + offset + posi\_color

## siguiente fila

jmp fila

fil\_exit:

#################################################

### RESTAURO EL CONTEXTO ANTERIOR A LA SUBRUTINA ###

pop %esi

pop %edi

pop %edx

pop %ecx

pop %ebx

pop %eax

#################################################

## Recuperar el antiguo frame

mov %ebp,%esp

pop %ebp

ret

.end

comp\_ejec\_vis\_as.sh:

#! /bin/bash

make clean #borra los ejecutables creados anteriormente

make #compila la subrutina pixels.s y el programa bmp\_as.c pasandole

#pixels.o

./bmp\_imagen #crea el bmp

display test\_as.bmp & #printea el bmp en pantalla