MPI PRACTICA 2

SISTEMA DISTRIBUIDO DE RENDERIZADO DE GRÁFICOS:

1. Enunciado.

Utilizaremos las primitivas pertinentes MPI2 como acceso paralelo a disco y gestión de procesos dinámico:
Inicialmente el usuario lanzará un solo proceso mediante **mpirun -np 1 ./pract2.** Con ello MPI lanza un primer proceso que será el que tiene acceso a la pantalla de gráficos, pero no a disco. Él mismo será el encargado de levantar N procesos (con N definido en tiempo de compilación como una constante) que tendrán acceso a disco, pero no a gráficos directamente.
Los nuevos procesos lanzados se encargarán de leer de forma paralela los datos del archivo *foto.dat*. Después, se encargarán de ir enviando los pixels al primer elemento de proceso para que éste se encargue de representarlo en pantalla.
Usaremos la plantilla pract2.c para comenzar a desarrollar la práctica. En ella debemos completar el código que ejecuta el proceso con acceso a la ventana de gráficos (rank 0 inicial) y la de los procesos "trabajadores".
Se proporciona el archivo foto.dat. La estructura interna de este archivo es 400 filas por 400 columnas de puntos. Cada punto está formado por una tripleta de tres "unsigned char" correspondiendo al valor R,G y B de cada uno de los colores primarios. Estos valores se pueden usar para la función dibujaPunto

2. Planteamiento de la solución.

El usuario ejecutará el programa solo creando 1 único proceso que será el proceso principal. Este después creará una cantidad de procesos trabajadores que vendrá definido de la siguiente forma:

```
#define NUM  4
```

Podemos distinguir dos roles en nuestro programa:

- **Rank 0**:
  - Es el proceso que creará a los procesos trabajadores.
  - Es el único que tiene acceso a gráficos.
  - Es el encargado de crear la ventana donde se dibujaran los puntos de la imagen.
  - Es el encargado de recibir todos los puntos que enviaran los trabajadores y pintarlos en pantalla.

- **Trabajadores:**
  - Son los nodos creados por el proceso principal.
  - Son los únicos que tienen acceso a disco.
  - Leeran el fichero "foto.dat" y procesaran su información.
  - Aplicaran el filtro a los pixeles que lean.
  - Enviaran el pixel procesado a el proceso principal.

3. Diseño de la solución.

El proceso principal creará tantos procesos hijos como se haya indicado y esperará a que estos manden todos los puntos de la imagen que les haya tocado procesar.
Los procesos trabajadores primero calcularan sobre cuantas filas de la imagen trabajaran. Una vez calculadas cuantas filas les tocan leerá del fichero "foto.dat" pixel a pixel (tripleta de 3 unsigned char que representan los colores RGB) y aplicara un filtro sobre estos valores y lo enviará al proceso principal.
Cada proceso trabajador tendrá una vista sobre el fichero "foto.dat", de esta forma se posicionará en la primera posición que le toque leer el fichero.

- **Espera de puntos (Proceso maestro).**

Después de crear a los procesos trabajadores el proceso principal se quedara esperando a que se le envíen todos los puntos y una vez que los reciba lo dibujará en la ventana.

```
void esperarPuntos(MPI_Comm intercomm)
{
	int buff[5];
	int i;
	long size_imagen = FILA_IMAGEN*FILA_IMAGEN;
	MPI_Status status;
	for(i=0;i<size_imagen;i++)
	{
		MPI_Recv(&buff,5,MPI_INT,MPI_ANY_SOURCE,1,intercomm,&status);
		dibujaPunto(buff[0],buff[1],buff[2],buff[3],buff[4]);
	}
}
```

- **Reparto de la imagen (procesos trabajadores).**

Esta función la ejecutaran los procesos trabajadores y se les devolverá un array de 2 posiciones. La primera posición será la primera fila del reparto y la segunda posición la última línea que deberán tratar.

```
int *RepartirTarea(int rank,int size)
{
	/* Reparto */
	int filaReparto = FILA_IMAGEN/NUM;
	static int reparto[2];
	reparto[INICIO] = rank * filaReparto;
	reparto[FINAL] = (rank+1) * filaReparto;
	if(rank==size-1)
	{
		reparto[FINAL] = FILA_IMAGEN;
	}


	return reparto;
}
```

- **Apertura del fichero (procesos trabajadores)**

Cada proceso abrirá una vista de la imagen correspondiente a la primera posición que les toca leer del fichero.

```
long bytesLeer = filaReparto*FILA_IMAGEN*3*sizeof(unsigned char);

MPI_File imagen;
MPI_File_open(MPI_COMM_WORLD, FOTO, MPI_MODE_RDONLY, MPI_INFO_NULL,&imagen);
MPI_File_set_view(imagen, rank*bytesLeer,MPI_UNSIGNED_CHAR, MPI_UNSIGNED_CHAR,
       "native", MPI_INFO_NULL);
```

- **Leer pixel del fichero "foto.dat" (procesos trabajadores)**

Cada proceso leera la tripleta de unsigned char y llamara a la función de aplciar filtro.

```
void tratarImagen(int rank,int *reparto, long bytesLeer, MPI_Comm
commPadre,MPI_File imagen,char *flag)
{
      int i,j;
      unsigned char pixel[3];
      int mensaje[5];
      MPI_Status status;

      /* Recorremos cada fila que nos ha tocado */
      for(i=reparto[INICIO]; i<reparto[FINAL]; i++)
      {
            /* Recorremos cada columna */
            for(j=0; j<COLUMNA_IMAGEN; j++)
            {
                  MPI_File_read(imagen, pixel, 3, MPI_UNSIGNED_CHAR, &status);
                  aplicarFiltro(j,i,pixel,commPadre,flag);
            }
      }
      MPI_File_close(&imagen);

}
```

- **Crear procesos trabajadores (proceso principal)**

El proceso principal creara tantos procesos trabajadores como este definido:

```
#define NUM  4
```

Para ello hará uso de la primitiva MPI_Comm_spawn

```
MPI_Comm_spawn("pract2",argv,NUM,MPI_INFO_NULL,0,MPI_COMM_WORLD,&intercomm,
        errcodes);
```

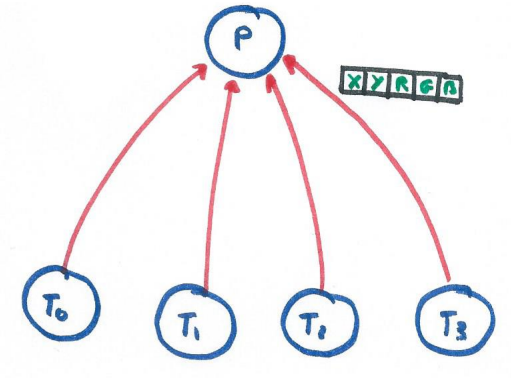
➢ Aplicar filtros (procesos trabajadores)

Una vez se tenga el pixel leído se llama a esta función y sobre los valores RGB aplica el filtro y lo manda al proceso 0.

```
void aplicarFiltro(int x,int y,unsigned char *pixel,MPI_Comm commPadre, char
*flag){
	int mensaje[5];
	MPI_Status status;
	mensaje[0]=x,mensaje[1]=y;
	int i=0;

	switch (*flag){
		case 'R':
			mensaje[2] = (int)pixel[0];
			mensaje[3] = 0;
			mensaje[4] = 0;
			break;
		case 'G':
			mensaje[2] = 0;
			mensaje[3] = (int)pixel[1];
			mensaje[4] = 0;
			break;
		case 'B':
			mensaje[2] = 0;
			mensaje[3] = 0;
			mensaje[4] = (int)pixel[2];
			break;
		case 'W':
mensaje[2] = (int)(pixel[0]*0.2986)+(int)(pixel[1]*0.5870)+(int)(pixel[2]*0.1140);
mensaje[3] = (int)(pixel[0]*0.2986)+(int)(pixel[1]*0.5870)+(int)(pixel[2]*0.1140);
mensaje[4] = (int)(pixel[0]*0.2986)+(int)(pixel[1]*0.5870)+(int)(pixel[2]*0.1140);
			break;
		case 'S':
  mensaje[2] = (int)(pixel[0]*0.393)+(int)(pixel[1]*0.769)+(int)(pixel[2]*0.189);
  mensaje[3] = (int)(pixel[0]*0.349)+(int)(pixel[1]*0.686)+(int)(pixel[2]*0.168);
  mensaje[4] = (int)(pixel[0]*0.272)+(int)(pixel[1]*0.534)+(int)(pixel[2]*0.131);
			break;
		case 'N':
			mensaje[2] = 255-(int)pixel[0];
			mensaje[3] = 255-(int)pixel[1];
			mensaje[4] = 255-(int)pixel[2];
			break;
		default:
			mensaje[2] = (int)pixel[0];
			mensaje[3] = (int)pixel[1];
			mensaje[4] = (int)pixel[2];
			break;
	}
	/* comprobamos que los pixeles no se hayan pasado de valor */
	for(i=2;i<=4;i++){
		if(mensaje[i]>255){
		    mensaje[i] = 255;
		}
		else if (mensaje[i]<0){
			mensaje[i] = 0;
		}
	}
	MPI_Bsend(&mensaje,5,MPI_INT,0,1,commPadre);
}
```

4. Explicación del flujo de datos de la red.



El proceso "P" es el proceso principal. Este es el encargado de crear a los demás procesos mediante la primitiva MPI_Comm_spawn y sera el que reciba los puntos que debe dibujar. Los Procesos T son los procesos trabajadores que trataran los puntos de la imagen y se lo mandaran al principal. El mensaje que mandarán serán 5 enteros. La posición en X, Y y los valores RGB.

5. Instrucciones de cómo compilar y ejecutar:

- ✓ Para compilar, hay dos opciones
  1. Automática con makefile
     - make all
  2. Manual:
     - mpicc pract2.c -o pract2 -lX11

- ✓ Para ejecutar: para elegir lo diferentes filtros hay que cambiar el argumento al ejecutar.
  - mpirun -np 1 ./pract2 <filtro>

Los filtros que se usan:

- Red:
  - make Red
  - mpirun -np 1 ./pract2 R
- Blue
  - make Blue
  - mpirun -np 1 ./pract2 B
- Green
  - make Green
  - mpirun -np 1 ./pract2 G
- BlackWhite
  - make BlackWhite
  - mpirun -np 1 ./pract2 W
- Sepia
  - make Sepia
  - mpirun -np 1 ./pract2 S
- Negativo
  - make Negativo
  - mpirun -np 1 ./pract2 N

6. Fuentes del programa:

```
/* Pract2  RAP 09/10    Javier Ayllon*/

#include <openmpi/mpi.h>
#include <stdio.h>
#include <stdlib.h>
#include <X11/Xlib.h>
#include <assert.h>
#include <unistd.h>
#define NIL (0)

#define NUM 4
#define FOTO "foto.dat"
#define FILA_IMAGEN 400
#define COLUMNA_IMAGEN 400
#define INICIO 0
#define FINAL 1

/*Variables Globales */

XColor colorX;
Colormap mapacolor;
char cadenaColor[]="#000000";
Display *dpy;
Window w;
GC gc;

/*Funciones auxiliares */

void initX() {

      dpy = XOpenDisplay(NIL);
      assert(dpy);

      int blackColor = BlackPixel(dpy, DefaultScreen(dpy));
      int whiteColor = WhitePixel(dpy, DefaultScreen(dpy));

      w = XCreateSimpleWindow(dpy, DefaultRootWindow(dpy), 0, 0,
                              400, 400, 0, blackColor, blackColor);
      XSelectInput(dpy, w, StructureNotifyMask);
      XMapWindow(dpy, w);
      gc = XCreateGC(dpy, w, 0, NIL);
      XSetForeground(dpy, gc, whiteColor);
      for(;;) {
            XEvent e;
            XNextEvent(dpy, &e);
            if (e.type == MapNotify)
                  break;
      }
      mapacolor = DefaultColormap(dpy, 0);
}

void dibujaPunto(int x,int y, int r, int g, int b) {

        sprintf(cadenaColor,"#%.2X%.2X%.2X",r,g,b);
        XParseColor(dpy, mapacolor, cadenaColor, &colorX);
        XAllocColor(dpy, mapacolor, &colorX);
        XSetForeground(dpy, gc, colorX.pixel);
        XDrawPoint(dpy, w, gc,x,y);
        XFlush(dpy);

}
```

```
/* -------------- Declaracion de funciones --------------- */

void tratarImagen(int rank,int *reparto, long bytesLeer, MPI_Comm
commPadre,MPI_File imagen,char *flag);
void aplicarFiltro(int x,int y,unsigned char *pixel,MPI_Comm commPadre, char
*flag);
void esperarPuntos(MPI_Comm commPadre);
int *RepartirTarea(int rank,int size);

/* Programa principal */

int main (int argc, char *argv[]) {

  int rank,size;
  MPI_Comm commPadre,intercomm;
  int tag;
  int errcodes[NUM];
  MPI_Status status;
  int buf[5];
  int i;


  MPI_Init(&argc, &argv);
  MPI_Comm_rank(MPI_COMM_WORLD, &rank);
  MPI_Comm_size(MPI_COMM_WORLD, &size);
  MPI_Comm_get_parent( &commPadre );
  if ( (commPadre==MPI_COMM_NULL) && (rank==0) )  {

      initX();

      /* Codigo del maestro */

MPI_Comm_spawn("pract2",argv,NUM,MPI_INFO_NULL,0,MPI_COMM_WORLD,&intercomm,err
codes);
      printf("[Proceso PRINCIPAL] He creado %d procesos \n",NUM);
      esperarPuntos(intercomm);
      printf("[Proceso PRINCIPAL] Pulse cualquier tecla para finalizar la
ejecucion...\n ");
      getchar();
      /*En algun momento dibujamos puntos en la ventana algo como
      dibujaPunto(x,y,r,g,b);  */
        }
  else {
     /* Reparto */
     int filaReparto = FILA_IMAGEN/NUM;
     int *reparto;
     reparto = RepartirTarea(rank,size);

     long bytesLeer = filaReparto*FILA_IMAGEN*3*sizeof(unsigned char);

      MPI_File imagen;
      MPI_File_open(MPI_COMM_WORLD, FOTO, MPI_MODE_RDONLY, MPI_INFO_NULL,
&imagen);
      MPI_File_set_view(imagen, rank*bytesLeer,
            MPI_UNSIGNED_CHAR, MPI_UNSIGNED_CHAR, "native", MPI_INFO_NULL);

      printf("[Proceso %d] y tengo inicio %d y final
%d\n",rank,reparto[INICIO],reparto[FINAL]);
      tratarImagen(rank,reparto,bytesLeer,commPadre,imagen,argv[2]);

  }
```

```
  MPI_Finalize();

}
/**
 *
 * Funcion encargada de leer los pixeles del fichero foto.dat y llamar a la
funcion aplicarfiltro por cada pixel
 * de la imagen
 *
*/
void tratarImagen(int rank,int *reparto, long bytesLeer, MPI_Comm
commPadre,MPI_File imagen,char *flag)
{

      int i,j;
      unsigned char pixel[3];
      int mensaje[5];
      MPI_Status status;

      /* Recorremos cada fila que nos ha tocado */
      for(i=reparto[INICIO];i<reparto[FINAL];i++)
      {
            /* Recorremos cada columna */
            for(j=0;j<COLUMNA_IMAGEN;j++)
            {
                  MPI_File_read(imagen, pixel, 3, MPI_UNSIGNED_CHAR, &status);
                  aplicarFiltro(j,i,pixel,commPadre,flag);
            }
      }
      MPI_File_close(&imagen);

}
/**
 *
 * Funcion encargada de aplicar el filtro al pixel leido y mandarlo al proceso
principal para que lo
 * dibuje en pantalla.
 *
*/
void aplicarFiltro(int x,int y,unsigned char *pixel,MPI_Comm commPadre, char
*flag)
{
      int mensaje[5];
      MPI_Status status;
      mensaje[0]=x;
      mensaje[1]=y;
      int i=0;

      switch (*flag)
      {
            case 'R':
                  mensaje[2] = (int)pixel[0];
                  mensaje[3] = 0;
                  mensaje[4] = 0;
                  break;
            case 'G':
                  mensaje[2] = 0;
                  mensaje[3] = (int)pixel[1];
                  mensaje[4] = 0;
                  break;
            case 'B':
                  mensaje[2] = 0;
                  mensaje[3] = 0;
                  mensaje[4] = (int)pixel[2];
                  break;
            case 'W':
```

```
                  mensaje[2] =
(int)(pixel[0]*0.2986)+(int)(pixel[1]*0.5870)+(int)(pixel[2]*0.1140);
                  mensaje[3] =
(int)(pixel[0]*0.2986)+(int)(pixel[1]*0.5870)+(int)(pixel[2]*0.1140);
                  mensaje[4] =
(int)(pixel[0]*0.2986)+(int)(pixel[1]*0.5870)+(int)(pixel[2]*0.1140);
                  break;
            case 'S':
                  mensaje[2] =
(int)(pixel[0]*0.393)+(int)(pixel[1]*0.769)+(int)(pixel[2]*0.189);
                  mensaje[3] =
(int)(pixel[0]*0.349)+(int)(pixel[1]*0.686)+(int)(pixel[2]*0.168);
                  mensaje[4] =
(int)(pixel[0]*0.272)+(int)(pixel[1]*0.534)+(int)(pixel[2]*0.131);
                  break;
            case 'N':
                  mensaje[2] = 255-(int)pixel[0];
                  mensaje[3] = 255-(int)pixel[1];
                  mensaje[4] = 255-(int)pixel[2];
                  break;
            default:
                  mensaje[2] = (int)pixel[0];
                  mensaje[3] = (int)pixel[1];
                  mensaje[4] = (int)pixel[2];
                  break;
      }
      /* comprobamos que los pixeles no se hayan pasado de valor */
      for(i=2;i<=4;i++)
      {
            if(mensaje[i]>255)
            {
                 mensaje[i] = 255;
            }
            else if (mensaje[i]<0)
            {
                  mensaje[i] = 0;
            }
      }

      MPI_Bsend(&mensaje,5,MPI_INT,0,1,commPadre);

}
/**
 *
 * Funcion que es llamada por el proceso principal, se encarga de recibir
todos los puntos de los
 * demas procesos y llamar a la funcion dibujaPunto para pintarlo
 *
*/
void esperarPuntos(MPI_Comm intercomm)
{
      int buff[5];
      int i;
      long size_imagen = FILA_IMAGEN*FILA_IMAGEN;
      MPI_Status status;
      for(i=0;i<size_imagen;i++)
      {
            MPI_Recv(&buff,5,MPI_INT,MPI_ANY_SOURCE,1,intercomm,&status);
            dibujaPunto(buff[0],buff[1],buff[2],buff[3],buff[4]);
      }
}

/**
 *
 * Cantidad de lineas de la imagen que le toca a cada proceso
 *
```

```
*/
int *RepartirTarea(int rank,int size){
      /* Reparto */
     int filaReparto = FILA_IMAGEN/NUM;
     static int reparto[2];
     reparto[INICIO] = rank * filaReparto;
     reparto[FINAL] = (rank+1) * filaReparto;
     if(rank==size-1)
     {
           reparto[FINAL] = FILA_IMAGEN;
     }


      return reparto;
}
```