UNIVERSIDAD DE CASTILLA-LA MANCHA

ESCUELA SUPERIOR DE INFORMÁTICA

**Sistema Distribuido de renderizado de Gráficos**

*Jesús Gamero Tello*

18/04/2020

# Indice

# 1. Sistema distribuido de renderizado de gráficos

## 1.1. Enunciado del problema

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 pixeles 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 400filas por 400columnasde 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.

Para compilar el programa para openMPI en linux el comando es:

**mpicc pract2.c -o pract2 -lX11**

## 1.2. Planteamiento de la solución

Bueno para el desarrollo de la solución se dispone de un código en C en que tenemos varios métodos y código escrito. Uno de estos utiliza la ventana gráfica que utiliza el servidor X11. Para el desarrollo de nuestra solución también disponemos del fichero foto.dat, el método que usa el servidor X11 y el método dibujaPunto.

En nuestro código tenemos dos partes diferenciadas, es decir dos tipos de procesos los procesos de tipo maestro y los de tipo trabajador. Dependiendo de si es un tipo de proceso o otro realiza una función u otra.

Si nuestro proceso es de tipo maestro realizara lo siguiente:
- Inicializará la ventana de gráficos
- Lanza los procesos trabajadores que procederán a la lectura del fichero foto.dat

- Recibirá los datos de los procesos trabajadores para pintar dichos datos en la ventana de gráficos que previamente habrá inicializado

Si nuestro proceso es de tipo trabajador realizara lo siguiente:
- Abren el fichero, proceden a su lectura y procesamiento de datos
- Aplican a la imagen el filtro elegido
- Finalmente envían la imagen con el filtro al proceso maestro para que este pinte la imagen

## 1.3. Diseño del programa

La imagen de la que disponemos en el fichero foto.dat tiene una estructura de 400x400, los pixeles de la imagen están representados mediante 3 unsigned char que se corresponden con los valores de colores R, G y B. Ahora procederemos a explicar las funciones que realizaran tanto los procesos trabajadores como el maestro o rank 0.

- **Maestro:** Crea la ventana gráfica donde se renderizará la imagen cuando se haya terminado de procesar el fichero foto.dat, para lo que se usara la función *initX().* A continuación, se lanzarán los procesos de los trabajadores el numero de procesos trabajadores esta definido mediante #define N. Para ejecutar todos los procesos utilizamos la función *MPI_Comm_spawn(),* que nos devuelve un intercomunicador para que los procesos creados se puedan comunicar, básicamente se utilizara para dibujar posteriormente los puntos en la ventana gráfica. La función *reciboPuntos()* recibe el intercomunicador y en esta función se realizan tantas iteraciones como píxeles tiene el fichero para recibir los puntos y que estos puntos sean pintados en mediante la función *dibujaPunto()*
- **Trabajadores:** abrimos el fichero mediante *MPI_File_Open(),* posteriormente mediante *MPI_File_set_view()* vamos a acceder a los datos correspondientes al rank. Leemos el fichero con la función *MPI_File_read()* y posteriormente aplicaremos el filtro modificando los colores R, G y B. Para aplicar los filtros utilizaremos un switch donde el case 0 es la imagen original, el case 1 B&N, el case 2 el sepia y el case 3 para invertir los colores, también disponemos de un default para que cuando se nos olvide aplicar un filtro salga con los colores originales. Cuando se ha aplicado su respectivo filtro se envían los datos al proceso maestro mediante la función *MPI_Bsend().*

## 1.4. Explicación del flujo de datos

Aquí pasaré a explicar las primitivas utilizadas en el desarrollo de la solución del problema:

- **MPI_File_open:** Esta primitiva abre un archivo para que todos los procesos tengan acceso a él. Abre un archivo de manera colectiva, permitiéndonos seleccionar el comunicador que en este caso es el MPI_COMM_WORLD.

```
MPI_File_open(MPI_COMM_WORLD,FOTO,MPI_MODE_RDONLY,MPI_INFO_NULL,&f);
```

- **MPI_Comm_spawn:** Lanza los procesos que ejecutaran el mismo binario, se levantaran tantos procesos como se le indique en nuestro caso tantos procesos como valor tenga N, y ejecutaran el binario “pract2”

```
MPI_Comm_spawn("pract2",MPI_ARGV_NULL,N,MPI_INFO_NULL,0,MPI_COMM_WORLD,&commPadre,array_of_errcodes);
```

- **MPI_File_read:** sirve para leer los datos de un fichero, en nuestro caso se lee pixel a pixel los datos de foto.dat

```
MPI_File_read(f,colores,3,MPI_UNSIGNED_CHAR,&status);
```

- **MPI_File_close:** sirve para cerrar el fichero

```
MPI_File_close(&f);
```

- **MPI_File_set_view:** esta primitiva obtiene una vista parcial de los datos de un archivo, aquí se utiliza para limitar la vista de los procesos trabajadores para que cada uno lea concurrentemente la parte que le toque

```
MPI_File_set_view(f,rank*tamaniobloque,MPI_UNSIGNED_CHAR,MPI_UNSIGNED_CHAR,"native",MPI_INFO_NULL);
```

- **MPI_Bsend:** se utiliza para enviar datos de manera síncrona

```
MPI_Bsend(&buf,5,MPI_INT,0,1,commPadre);
```

## 1.5. Fuentes del programa

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

#define N 10
#define FOTO "include/foto.dat"
#define SIZE 400
#define MODE 2


/*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);

}

```

```c
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);

}

void puntos(MPI_Comm commPadre){
      MPI_Status status;

      int buff[5];
      for (int i=0;i<(SIZE*SIZE);i++){
           MPI_Recv(&buff,5,MPI_INT,MPI_ANY_SOURCE,MPI_ANY_TAG,commPadre,&status);
           dibujaPunto(buff[0],buff[1],buff[2],buff[3],buff[4]);
      }

}
```

```c
/* Programa principal */

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

  int rank,size;
  MPI_Comm commPadre;
  int tag;
  MPI_Status status;
  int buf[5];
  int array_of_errcodes[5];


  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",MPI_ARGV_NULL,N,MPI_INFO_NULL,0,MPI_COMM_WORLD,&commPadre,array_of_errcodes);
      /*En algun momento dibujamos puntos en la ventana algo como
      dibujaPunto(x,y,r,g,b);  */
      puntos(commPadre);

      sleep(5);

      }
```

```
109
110
111  else {
112    /* Codigo de todos los trabajadores */
113    /* El archivo sobre el que debemos trabajar es foto.dat */
114      MPI_File f;
115
116      int filas,tamaniobloque,inicio,fin;
117
118      filas=SIZE/N;
119      tamaniobloque=filas*SIZE*3*sizeof(unsigned char);
120      inicio=rank*filas;
121      fin=((rank+1)*filas);
122      if (rank==N-1){
123           fin=SIZE;
124      }
125      MPI_File_open(MPI_COMM_WORLD,FOTO,MPI_MODE_RDONLY,MPI_INFO_NULL,&f);
126      MPI_File_set_view(f,rank*tamaniobloque,MPI_UNSIGNED_CHAR,MPI_UNSIGNED_CHAR,"native",MPI_INFO_NULL);
127
128      unsigned char colores[3];
129      int i,j;
130
131      for(i=inicio;i<fin;i++){
132           for(j=0;j<SIZE;j++){
133
134                MPI_File_read(f,colores,3,MPI_UNSIGNED_CHAR,&status);
135                buf[0]=j;
136                buf[1]=i;
137
138                switch (MODE)
139                {
140                case 0:
141                     buf[2]=(int) colores[0];
142                     buf[3]=(int) colores[1];
143                     buf[4]=(int) colores[2];
144                     break;
145
146                case 1://B&N
147                     buf[2]=((int) colores[0]+(int) colores[1]+(int) colores[2])/3;
148                     buf[3]=((int) colores[0]+(int) colores[1]+(int) colores[2])/3;
149                     buf[4]=((int) colores[0]+(int) colores[1]+(int) colores[2])/3;
150                     break;
151
152                case 2://sepia
153                     buf[2]=((int) colores[0]*0.393+(int) colores[1]*0.769+(int) colores[2]*0.189);
154                     buf[3]=((int) colores[0]*0.349+(int) colores[1]*0.686+(int) colores[2]*0.168);
155                     buf[4]=((int) colores[0]*0.272+(int) colores[1]*0.534+(int) colores[2]*0.131);
156                     break;
157
158                case 3://invertimos color
159                     buf[2]=255-(int) colores[0];
160                     buf[3]=255-(int) colores[1];
161                     buf[4]=255-(int) colores[2];
```

```
162                         break;
163
164                   default:
165                         buf[2]=(int) colores[0];
166                         buf[3]=(int) colores[1];
167                         buf[4]=(int) colores[2];
168                         break;
169                   }
170
171                   MPI_Bsend(&buf,5,MPI_INT,0,1,commPadre);
172
173             }
174       }
175
176
177
178       MPI_File_close(&f);
179   }
180
181   MPI_Finalize();
182   return EXIT_SUCCESS;
183
184 }
```