DESARROLLO DE SISTEMAS DISTRIBUIDOS

Proyecto 2

El proyecto consiste en una aplicación cliente-servidor completamente en lenguaje C++.

Se desea construir la primera parte de un simulador espacial 2D para asteroides donde una computadora funciona como visualizador (servidor), y otra computadora se utiliza para realizar cálculos (cliente), la cual no necesariamente dispone de un ambiente gráfico.

Un ejemplo de lo que se desea en la parte gráfica respecto a los asteroides se muestra en el videojuego arcade ***Asteroids*** de Atari:

http://www.youtube.com/watch?v=WYSupJ5r2zo

Dado que es necesaria una transmisión rápida y eficiente de los datos, así como mantener una visualización lo más continua posible, se elige utilizar sockets UDP en lugar de los sockets TCP.

http://www.diffen.com/difference/TCP_vs_UDP

Para enviar la información gráfica se tienen dos opciones: gráficos vectoriales (http://es.wikipedia.org/wiki/Gr%C3%A1fico_vectorial) o gráficos de mapas de bits o raster (http://es.wikipedia.org/wiki/Imagen_de_mapa_de_bits).

Para evitar saturar el ancho de banda se elige utilizar gráficos vectoriales.

Es importante indicar que estas decisiones se toman de acuerdo a las recomendaciones teóricas de desempeño sugeridas por Tanenbaum:

Desempeño

Cuando se ejecuta una aplicación en un sistema distribuido *no debe parecer peor que su ejecución en un único procesador*

Para *optimizar el desempeño* frecuentemente hay que:

- Minimizar el número de mensajes:
    - La dificultad es que la mejor forma de mejorar el desempeño es tener muchas actividades en ejecución paralela en distintos procesadores, pero esto requiere el envío de muchos mensajes.

También se debe prestar atención al *tamaño de grano* de todos los cálculos:

- *Paralelismo de grano fino:*
  - Corresponde a trabajos con un gran número de pequeños cálculos y mucha interacción con otros trabajos, debido a ello requieren mucha comunicación que puede afectar el desempeño.
- *Paralelismo de grano grueso:*
  - Corresponde a trabajos con grandes cálculos, poca interacción y pocos datos, por lo tanto requieren poca comunicación y no afectan el desempeño.

En esta primera parte:

El cliente se encarga de generar n objetos irregulares distintos (asteroides), así como su posición inicial que deberá encontrarse fuera de la pantalla. Se va a considerar que todos los asteroides viajan a la misma velocidad, y que cuando se encuentran los asteroides no existen colisiones que los destruyan como muestra el video.

El número de asteroides n que se van a generar, así como el número máximo de vértices en cada asteroide m, deberán pasarse como parámetro en la línea de comandos del cliente (véase capítulo 2 “Programación de Sistemas Linux”). El límite en el número de asteroides así como el número de vértices en los objetos irregulares, será el tamaño máximo del paquete UDP que por acuerdo dejaremos en 1024 bytes. Es decir, toda la información necesaria para desplegar un frame completo, no deberá ocupar más de 1024 bytes. Nótese que el cliente debe generar toda la información correspondiente a un frame y el servidor únicamente se debe dedicar a imprimir la imagen correspondiente en la pantalla.

¿Cuántos paquetes UDP se deben enviar por segundo al servidor? ¿Qué sucede si se pierde algún paquete?

Dado que se irán añadiendo y/o modificando características al simulador, es importante que todo el código sea totalmente orientado a objetos. Elabore las clases necesarias, con implementación e interfaz en archivos separados.

Para la parte gráfica utilizaremos las librerías X11 de UNIX (véase capítulo 16 “Programación de Sistemas Linux”). Posteriormente puede analizar y utilizar el siguiente código fuente para realizar una simulación sencilla:

```
#include <X11/Xlib.h>
#include <unistd.h>

unsigned long ObtieneColor( Display* dis, char* color_name )
{
   Colormap cmap;
   XColor color_cercano, color_verdadero;

   cmap = DefaultColormap( dis, 0 );
   XAllocNamedColor( dis, cmap, color_name, &color_cercano, &color_verdadero );
   return( color_cercano.pixel );
```

```
}

int main()
{
   Display *disp = NULL;
   Window ventana;
   XColor color;
   int t;

   disp = XOpenDisplay(NULL);
   ventana = XCreateSimpleWindow (disp, XDefaultRootWindow (disp), 100, 100, 500, 500, 1,
1,BlackPixel (disp, DefaultScreen(disp)));
   XMapWindow (disp, ventana);

   for (t = 0; t < 100; t++)
   {
      XSetForeground( disp, XDefaultGC (disp, DefaultScreen(disp)), BlackPixel(disp,
0)^ObtieneColor( disp, "green"));
      XClearWindow(disp,ventana);
      XDrawLine(disp,ventana,XDefaultGC (disp, DefaultScreen(disp)), t*1+80, t*2+40,
t*2+40,t*3+80);
      XDrawLine(disp,ventana,XDefaultGC (disp, DefaultScreen(disp)), t*5+80, t*3+40,
t*3+40,t*5+80);
      usleep(41666); //24 por segundo
   }

   XDestroyWindow( disp , ventana );
   XCloseDisplay( disp );

   return(0);
}
```

Se recomienda utilizar `XDrawLines` en lugar de `XDrawLine` .

Para subirse a MOODLE, los códigos de los programas elaborados (principal, interfaz, implementación, Makefile) deben concatenarse en un archivo de texto plano, dejando una línea de asteriscos “*” entre cada archivo. Al inicio de cada archivo y en la primera línea debe aparecer el nombre del archivo, posteriormente y en las siguientes líneas un breve comentario sobre lo que realiza dicho archivo.

El nombre del archivo debe ser el nombre del alumno separado con guion bajo, materia (DSD), grupo, numero de proyecto y extensión txt. El no cumplir con estos requisitos provocará la disminución de la calificación.

Ejemplo de un nombre de archivo:

`Juan_Perez_Molinar_DSD_4CM2_2.txt`

Advertencia: Evite copiar programas y que le sean copiados, cualquier acto de plagio se castigará para plagiario y plagiado con cero.