



O A X A C A

# Universidad Tecnológica de la Mixteca

## Sistemas Distribuidos

Profesor: Gabriel Gerónimo Castillo

### Reporte: Proyecto “Adivina Palabras”

Alumnos:

Ventura Mijangos Geovanni

Peralta Rosales Oscar Esaú

Entrega Final

702-A Ing. Computación

Fecha: 12/Febrero/2014

## INTRODUCCIÓN

En el presente documento se desarrolla el proyecto final de la materia de Sistemas Distribuidos, el cual se denomina “Adivina Palabras”. El proyecto tiene como finalidad aplicar de manera práctica los conocimientos adquiridos en la materia a lo largo del curso.

Para el desarrollo e implementación del proyecto, se trabajará bajo el sistema operativo Linux en cualquiera de sus distribuciones. Asimismo es indispensable contar con una red de computadoras para la implementación.

El proyecto fue desarrollado mediante el lenguaje C. En el planteamiento del proyecto, el problema nos menciona que se desarrollará un juego multiusuario, el cual será posible jugarlo entre N número de usuarios dependiendo este número N del archivo de configuración de la red que se creará. El juego consiste en que los usuarios adivinen en un cierto lapso de tiempo determinado, la palabra que se esconde dentro de un conjunto de caracteres (Letras) que forman parte de la palabra a adivinar. Estas palabras formarán parte de un tema en específico. Cada palabra trae consigo un Tip, esto quiere decir que se dará una sola pista para cada una de ellas. Estos temas serán elegidos por el servidor de la red de computadoras mediante la lectura de un archivo donde se encontrarán los temas, palabras y tips. Además de que éste, será capaz de elegir temas que no estén en el archivo ya que podrá crear su propio tema en el inicio de la ejecución del juego. Para esto deberá ingresar el nombre del tema, el número de palabras para el juego y por último las palabras con sus respectivos tips. Durante el juego se implementa un tiempo de 15 segundos de espera, este es el lapso de tiempo que el usuario tiene para escribir su respuesta, de lo contrario la palabra se pasará al siguiente jugador para que tenga la oportunidad de dar su respuesta. En caso que ningún usuario tenga éxito adivinando la palabra en su primer turno, la palabra seguirá pasando de usuario a usuario hasta que alguien logre adivinarla y así dar paso a la siguiente y esto se repite hasta terminar las palabras del tema elegido por el servidor.

Para el desarrollo del proyecto se utilizaron sockets, memoria compartida y generación de procesos padres y procesos hijos o subprocessos.

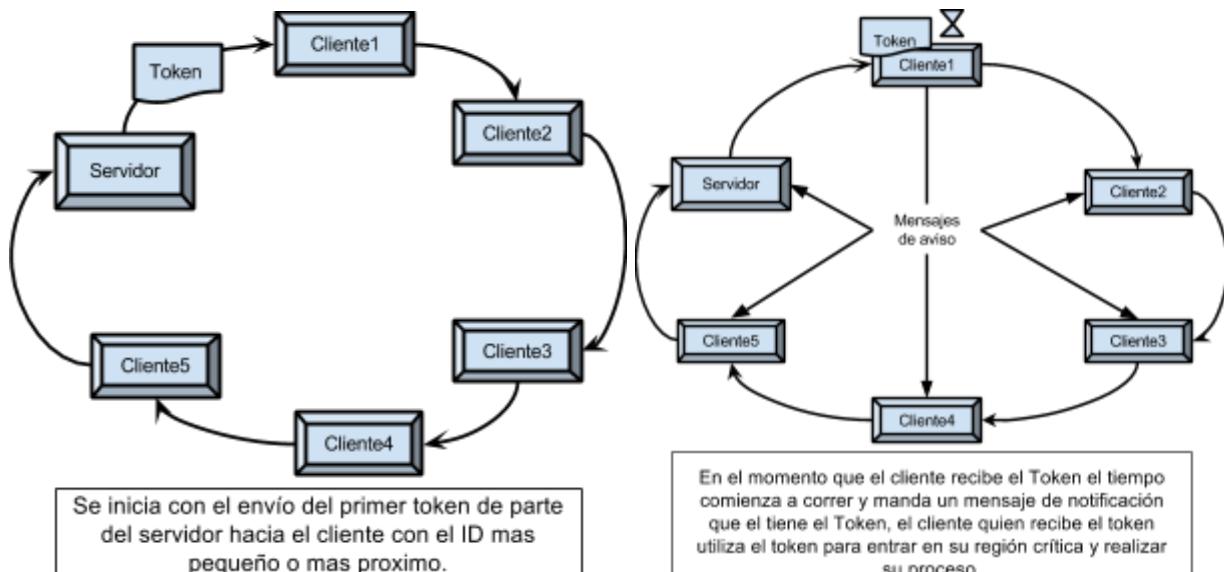
# DESARROLLO

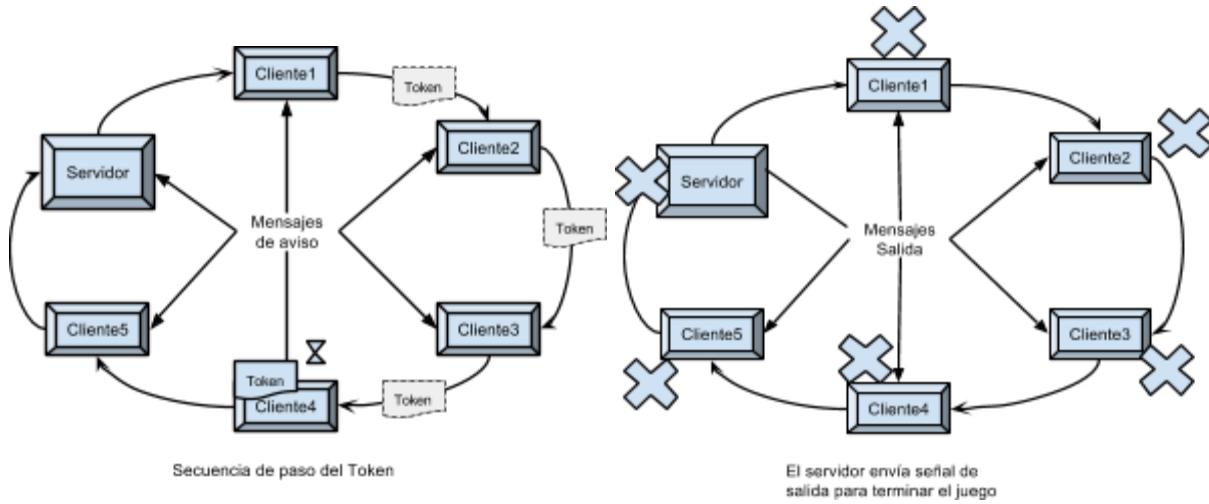
## • Algoritmos Implementados

Para este desarrollo de nuestro proyecto se implementó un algoritmo de Token-Ring, el cual consiste en una topología de anillo y una técnica de paso de Token, este token viaja alrededor del anillo circulando de proceso en proceso. Cada uno de los clientes en el anillo tiene determinado tiempo el token, si en ese tiempo se agota tiene que darle tiempo al siguiente cliente enviando el token. Este token permite que los clientes ejecuten el proceso necesario (entrar en su región crítica) y posteriormente pasen el token, pero si este tiempo se agota el y no se ha ejecutado ningún proceso el token deberá ser enviado al siguiente cliente en el anillo.

El servidor es el monitor de los clientes y es quien tiene el poder de romper el anillo y de esta manera dejar de enviar el Token. En este algoritmo no es permitido entrar en la región crítica dos veces con el mismo token, esto quiere decir que un cliente no puede realizar un proceso terminarlo y teniendo el mismo token volver a ejecutar otro proceso, sino que tiene que esperar a que el servidor vuelva a lanzar un nuevo token.

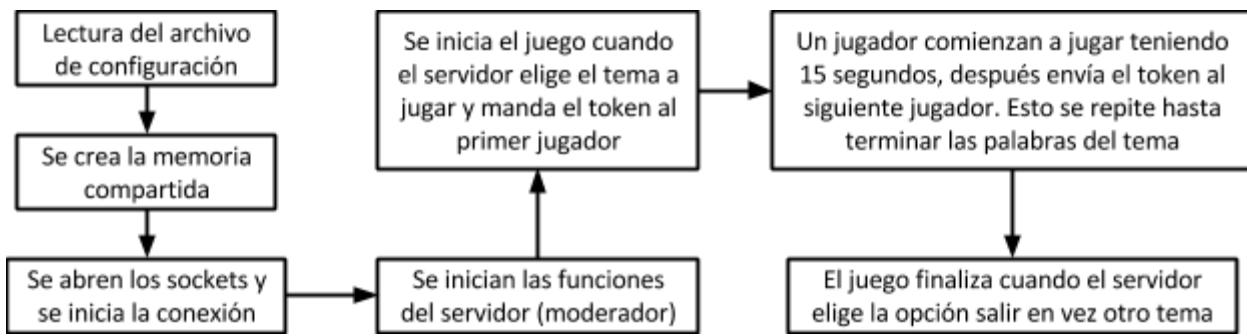
A continuación se muestra una secuencia de imágenes de como trabaja el algoritmo Token-Ring, en cada una de ellas se muestra las fases ya mencionadas por las cuales puede llegar a pasar.





### • Relacion de Modulos Principales

Para su implementación el proyecto se organizó en tres archivos principales, el archivo de mayor jerarquía es el servidor.c el cual contiene el cuerpo esencial del juego y las llamadas a las funciones principales para realizar las conexiones para formar la red, las cuales se encuentran en el archivo que se denominó main.c. También se cuenta con el archivo temas.c, en el cual se tiene todas las funciones relacionadas con la lectura del archivo que contiene la información para el juego. A continuación se muestra un diagrama a bloques con las partes más esenciales del funcionamiento del programa:



Ahora explicaremos cada uno de los módulos programados es decir detallaremos los características y funcionalidades de cada uno de los archivos y sus funciones, en la forma en la cual se utiliza en el momento de la ejecución del proyecto.

En el archivo principal Servidor, se inicia llamando a la función `read_IP_file()`; la cual se encuentra implementada en el archivo main.c, esta función realiza la tarea de lectura del archivo de configuración donde se localiza la información acerca de las computadoras que forman parte de la red para poder jugar. En este archivo de configuración se tiene

primeramente el número de jugadores que se conectarán a la red incluyendo a la máquina servidor, posteriormente contiene las direcciones IP's de las computadoras así como un Id de identificación. A continuación en la Figura 1 se muestra un ejemplo del contenido de este archivo de configuración:

```
3
192.168.43.67 - 1
192.168.43.50 - 2
192.168.43.194 - 0
```

Figura 1. Archivo de Configuración

El primer número representa el número de máquinas que estarán en la red de computadoras, enseguida las direcciones IP's seguidas de su identificador, el cual deberá ser de cero en adelante sin saltos, es decir, deberá ser una secuencia finita de identificadores(0,1,2,3,..., N) separados por un guión (-). La primer IP en el archivo deberá de ser del servidor local y las siguientes de los servidores externos. Un dato importante de este archivo para el programa es el identificador ya que el número mayor de los identificadores será el número que se tomará como el servidor (coordinador), quien es el que elige los temas y quien elige en qué momento se termina el juego. Finalmente en esta primera función se hacen las impresiones de lo identificado, quien es el servidor (coordinador) y quienes son los demás servidores identificados como se ve en la Figura 2.

```
+ x ...ocumentos/Final
geovanni@Gio:~/Documentos/Final$ gcc servidor.c
^[[AgeoVanni@Gio:~/Documentos/Final$ ./a.out 4362
Leyendo archivo de configuración...
Número de Servidores: 3
IP Servidor Local: 192.168.43.67 | ID = 1
Servidor 1: 192.168.43.50 | ID = 2 => COORDINADOR
Servidor 2: 192.168.43.194 | ID = 0
```

Figura 2. Impresiones de lo leído en archivo de configuración.

Ahora bien después de la identificación del archivo se valida que al menos existan dos jugadores en la red ya que es el número mínimo con el que se realizó la validación para poder jugar. Enseguida se procede a la función **createRequest()**; en esta función se crea la memoria compartida del programa, primero se obtuvieron dos claves para la memoria compartida mediante la llamada a la función *ftok(X,Y)*, donde **X** es nombre de un fichero existente y accesible, mientras que **Y** representa una variable entera que no debe ser 0, posteriormente se crea el identificador de la memoria compartida con la función *shmget(key\_t key, int size, int shmflg)*, el primer argumento es la llave creada por el sistema mediante la función *ftok* antes mencionada, el segundo argumento nos indica el tamaño de la

memoria compartida y el tercer argumento indica los permisos para acceder a la memoria compartida, para crear la zona de memoria usamos 0777 | IPC\_CREAT el número indica permisos de lectura, escritura y ejecución. Por último se crea una variable del proceso que apunte a la zona de memoria mediante la función `void *shmat(int shmid, char * shmaddr, int shmflg)`, en el primer argumento se tiene el identificador obtenido con la función anterior, el segundo argumento normalmente, el valor será de Cero o Nulo esto indica al sistema operativo que busque en una zona de memoria libre. Este apuntador a la memoria compartida fue inicializado en -1 con la función `memset`.

Al finalizar la creación de la memoria compartida correctamente el programa proceda a abrir el socket del servidor, en el caso de no poder abrirlo nos manda un mensaje de error, en la Figura 3 se muestra este segmento de código.

```
/* Socket Servidor */
if( (Socket_Servidor = Abrir_Socket_INET( NULL, 0 )) == -1 ) {
    printf("Error al abrir socket Servidor\n");
    exit(EXIT_FAILURE);
}
```

Figura 3. Abriendo Socket Servidor.

Al terminar la apertura del socket servidor se llama a la función `recibe_conexiones()`; esta es quien levanta sockets a la escucha para cada máquina que se conecte a este servidor, en esta función se comienza creando un subprocesso con la función `fork()`, si se logra bifurcar el proceso, comienza a correr un ciclo el cual se detendrá hasta que el número de clientes haya logrado las conexiones con los demás servidores, en este ciclo por cada uno de los servidores externos se creará otro nuevo proceso dentro del cual se abrirán los sockets cliente esto mediante la función `Aceptar_Conexiones` la cual se muestra en la Figura 4. Si la conexión es aceptada este segundo subprocesso manda a llamar a la función `escuchar` esta función escucha peticiones de un socket asociado.

```
int Aceptar_Conexiones(int fd_Servidor, int nc) {
    int fd;
    unsigned int sin_size = sizeof(struct sockaddr_in);
    if( ( fd = accept( fd_Servidor, (struct sockaddr *)&(Host[nc]), &sin_size ) ) == -1 )
        return -1;
    printf("Conexión establecida desde: %s\n", inet_ntoa(Host[nc].sin_addr));
    return fd;
}
```

Figura 4. Función para abrir sockets cliente.

Ahora bien ya recibidas las conexiones en la función anterior, se llama a la función `inicia_conexiones()`; esta es quien realiza la conexión con los demás servidores, en la Figura 5 se observa el código que realiza esta función.

```

void inicia_conexiones() {
    int nc;
    /* Tiempo de espera para levantar las demás máquinas*/
    sleep(7);
    printf("Estableciendo conexión con los demás servidores...\n");
    /* Abre los sockets para conectarse como cliente con las demás máquinas*/

    for( nc=0; nc < NUM_HOST ; nc++ )
        if( (socket_Host_ip[nc] = Abrir_Socket_INET( host_ip[nc], nc )) == -1 ) {
            printf("Error al abrir socket con %s\n", host_ip[nc]);
            exit(EXIT_FAILURE);
        }
    sleep(4);
}

```

Figura 5. Abriendo Socket a la escucha para cada máquina.

Si las conexiones se establecieron correctamente, el programa prosigue a la identificación del Servidor de la red es decir que creará las partidas del juego para los demás jugadores. Si el ID es el del coordinador manda a llamar a la función `lee_Temas()`; la cual se encuentra en el archivo temas.c, en esta se abre un archivo temas.txt el cual contiene la información de los temas con los cuales se podrá iniciar el juego. El formato de este archivo es el que se muestra en la Figura 6, el formato es tal cual en este ejemplo ya que de otra manera no será reconocido de forma correcta.

```

1 Número de TEMAS : 2
2
3 TEMA : Planetas
4 Num_Pal : 9
5
6 1 : MERCURIO
7     Tip : Es el planeta más pequeño del sistema solar
8 2 : VENUS
9     Tip : Los romanos le pusieron este nombre a su Diosa del Amor
10 3 : TIERRA
11     Tip : Le llaman el planeta azul
12 4 : MARTE
13     Tip : Le llaman el planeta rojo
14 5 : JUPITER
15     Tip : Es el planeta más grande del sistema solar
16 6 : SATURNO
17     Tip : Es muy conocido por sus bellos anillos
18 7 : URANO
19     Tip : Es el tercer planeta más grande después de Júpiter y Saturno
20 8 : NEPTUNO
21     Tip : Un personaje con Tridente en la caricatura de Bob Esponja tiene su
22 mismo nombre
23 9 : PLUTON
24     Tip : Es el planeta más alejado del sol
25
26 TEMA : Animales Domésticos y de Granja
27 Num_pal : 10
28 1 : Perro
29     Tip: Se caracteriza por su lealtad
30 2 : Gato
31     Tip: A veces comen ratones
32 3 : Burro
33     Tip: Es muy fuerte y resistente para llevar cargas pesadas
34 4 : Gallina
35     Tip: Amenudo comes algo que sale de su colita D:
36 5 : Hamster
37     Tip: Son muy pequeños y les gusta correr en su rueda
38 6 : Vaca
39     Tip: Produce una bebida muy nutritiva
40 7 : Borrego
41     Tip: Muchas chamarras son calientitas gracias a él
42 8 : Caballo
43     Tip: Cuando es libre vive en manadas y corre mucho
44 9 : Perico
45     Tip: Aprenden a repetir palabras
46 10 : Alpaca
47     Tip: Se dice que el camello es su parente cercano
48

```

Figura 6. Archivo de Información Temas.

Enseguida de culminar la lectura del archivo de los temas se procede a la ejecución de la función `inicia_juego()`, la cual esta ubicada en el archivo servidor c, en esta función se puede visualizar un menú en el cual el servidor puede elegir un tema de los existentes en el archivo o también puede crear un nuevo tema, además cuenta con una opción de salir en caso de que ya no se quiera continuar con el juego. También en esta función se elige una palabra de forma aleatoria de las que contiene el tema elegido, y es mandada mediante la función `send_message` al jugador número uno en la red.

Mediante el apuntador a la memoria compartida el servidor puede ir verificando si la palabra ha sido adivinada por alguno de los usuarios o si han sido adivinadas todas las palabras para así poder elegir otro tema o salir del juego. En seguida se presenta un ejemplo de como esta función utiliza la memoria compartida en un ciclo While (Figura 7).

```

while( vectorRequest[ TOTAL_ADIVINADAS ] < TEMAS[TEMA_ACTUAL].numPalabras ) { }

```

Figura 7.Uso de la memoria compartida.

En este ciclo se muestra la condición donde se utiliza la memoria compartida, el apuntador que en un principio se mencionó fue llamado *vectorRequest* aquí se verifica que el total de palabras adivinadas sea menor al total de número de palabras que el tema actual contiene. Aquí se utiliza un vector de tipo TEMA este es creado en el archivo de temas.c, esta estructura se muestra a continuación al igual que la declaración de la variable (Figura 8).

```
struct tema
{
    char name[100];
    int numPalabras;
    char palabra[100][100];
    char tip[100][200];
};

typedef struct tema TEMA;

TEMA TEMAS[ NUM_MAX_TEMAS ];
```

Figura 8. Estructura de Tema.

Ahora bien si el ID de la computadora no es el del coordinador entonces manda a llamar a la función **adivina\_la\_palabra()**; lo que nos dice que este es un usuario(jugador). La primera acción en esta función es verificar en la memoria compartida si se ha llegado al fin de las palabras en el tema elegido este envía un mensaje diciendo que se ha llegado al final de la partida. Posteriormente si es el turno del usuario X y este está listo, envía un mensaje con la función *send\_message* a todos los usuarios que es el turno de el de adivinar la palabra. A continuación se muestra el contenido de la función *send\_message* (Figura 9).

```
void send_message( int nc, char *message ) {
    send(socket_Host_ip[ nc ], message, strlen(message), 0);
}
```

Figura 9. Envíos de Mensajes entre servidores.

Aquí se muestra como se envía a cada uno de los sockets de cada usuario el mensaje del usuario que esta en uso del tiempo para adivinar la palabra. Después de mandar el mensaje inmediatamente el usuario obtiene el Token que fue enviado por el usuario que le antecede, mediante la función *get\_pal\_tip(s, tip)*; esta obtiene la palabra y el tip enviado.

En el momento que el usuario visualiza la impresión de la palabra en desorden y el tip, que son impresos justo después de obtenerlos, comienza a correr el tiempo que este tiene para responder. También en esta función es crear un subprocesso mediante la función *fork()* , en este subprocesso el usuario ingresa la respuesta que él ha considerado es la

correcta, enseguida que su respuesta es enviada el programa compara la respuesta del usuario con la palabra original. En el momento que la respuesta coincide con la palabra del tema se muestra un mensaje de acierto al usuario y este envía un mensaje mediante la función antes vista send\_message se envía otro mensaje a los demás usuarios que esa palabra ha sido respondida correctamente. Al finalizar el envío de mensajes se accede a la memoria compartida para indicar que dicha palabra ha sido contestada. Si el usuario no acierta en la respuesta, si se encuentra dentro del lapso de tiempo el programa le dice que intente de nuevo.

Hay que resaltar que estas acciones mencionadas son hechas por el proceso hijo mientras que el proceso padre lleva a cabo el conteo de tiempo. Después se verifica si la palabra ha sido adivinada si ha sido así el estado se regresa al estado inicial en la memoria compartida, en caso contrario muestra un mensaje de que se ha acabado el tiempo y envía el Token al siguiente usuario y matamos el proceso hijo que se creó. En el caso contrario que no sea turno del usuario X a este le aparecerá un mensaje de que debe esperar a que sea su turno.

Por último si el servidor ha decidido finalizar la sesión de juego se llama a la función **cerrar\_conexiones()**; la cual cierra los puertos abiertos por los sockets a continuación se muestra una imagen del código(Figura 10). Después de cerrar los sockets se libera la memoria compartida creada mediante la función siguiente *shmctl* (*int shmid, int cmd, struct shmid\_ds \*buf*), en el primer parámetro es para el identificador de la memoria compartida, para el segundo parámetro de la función se inserta la opción **IPC\_RMID** que marca esa zona de memoria para ser liberada cuando ya no haya ningún proceso vinculado a ella y por último el tercer parámetro es un valor nulo o 0.

```
void cerrar_conexiones() {
    int nc;
    for( nc=0; nc < NUM_HOST ; nc++)
        close(socket_Host_ip[nc]);
    close(Socket_Servidor);
}
```

Figura 10. Cierre de Puertos.

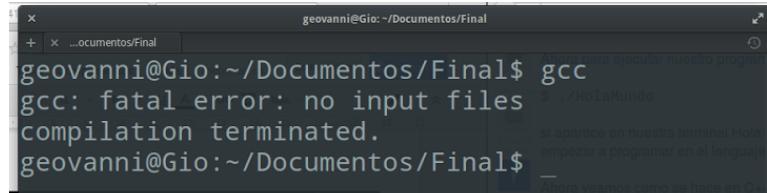
- **Manual de instalación de la aplicación**

Para la instalación de la aplicación “Adivina Palabra” es necesario contar con ciertos requerimientos con los cuales el programa se ejecutará sin ningún problema. A continuación se listan los requerimientos necesarios:

1. Primero que nada será necesario contar con más de un Jugador y un servidor, es decir mínimo deben contar con 3 computadoras que se conectarán a la red.
2. Contar con una red a la cual se conectarán los usuarios para poder interactuar con el juego
3. Contar con una máquina con un sistema operativo Linux, no importando la distribución. En caso de no tener instalado el sistema operativo Linux, deberá ser instalado si no tiene conocimientos acerca de como instalarlo revise el Apéndice A.
4. Ahora bien ya tenemos el Sistema operativo, a continuación tendremos que entrar a nuestra terminal y verificar si tenemos instalado el compilador del lenguaje de programación C, para verificar pondremos en nuestra terminal en siguiente comando:

**\$ gcc**

En caso de que el sistema operativo ya cuente con el compilador del lenguaje nos aparecerá un mensaje parecido al de la Figura 11.



```
geovanni@Gio:~/Documentos/Final$ gcc
gcc: fatal error: no input files
compilation terminated.
geovanni@Gio:~/Documentos/Final$
```

Figura 11. Error generado porque el Compilador ya está instalado.

De lo contrario aparecerá el siguiente mensaje:

El programa «gcc» no está instalado. Puede instalarlo escribiendo:  
**sudo apt-get install gcc**

Solo se tiene que teclear el comando sugerido y podrá comenzar la instalación del compilador, el comando es:

**\$ sudo apt-get install gcc**

Se teclea la contraseña del usuario que nos solicita y respondemos “si” cuando nos pregunte si queremos instalarlo y presionamos la tecla de afirmación ya sea ‘S’ o ‘Y’ y pulsamos enter, esperamos a la descarga e instalación del paquete.

5. Ahora bien copiamos la carpeta que se proporcionará donde se encuentran los archivos Servidor.c, main.c, main.h, temas.c, conf y el archivo de temas.txt solo se le proporcionará al usuario que servirá como servidor. Estos archivos se colocarán en la carpeta que el usuario desee.

6. Una vez conectadas las máquinas a la red se configura el archivo “conf” con las IP’s de las máquinas asociadas, si no se conoce la IP de una máquina el comando “ifconfig” nos las muestra. El archivo debe seguir la siguiente estructura.

```
3
192.168.43.67 - 1
192.168.43.50 - 2
192.168.43.194 - 0
```

Figura 12. Archivo de Configuración

- a. El primer dígito representa el número de computadoras conectadas a la red.
  - b. La primer IP debe corresponder a la computadora local donde se esta configurando.
  - c. Las siguientes IP's corresponden a las demás computadoras de la red.
  - d. Cada IP tiene asociado un ID único como identificador separado por un guión, los ID's deben ser valores secuenciales de la forma (0,1,2,...n), no se permiten saltos.
  - e. La IP de la computadora con el ID más alto representa al Coordinador del juego, así qué todas las IP's yID's deben coincidir en todas las computadoras.
7. El archivo temas.txt contiene los temas predefinidos para el juego, si se desean agregar más temas a este archivo se debe seguir la siguiente estructura.

```

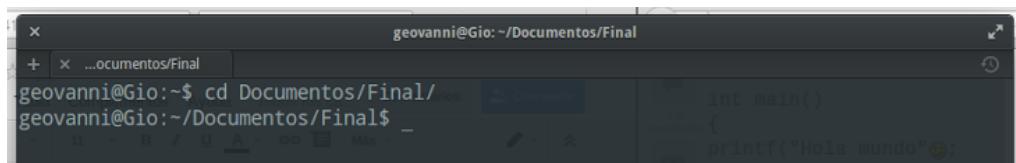
1 Número de TEMAS : 2
2
3 TEMA : Planetas
4 Num_Pal : 9
5
6   1 : MERCURIO
7     Tip : Es el planeta más pequeño del sistema solar
8   2 : VENUS
9     Tip : Los romanos le pusieron este nombre a su Diosa del Amor
10  3 : TIERRA
11    Tip : Le llaman el planeta azul
12  4 : MARTE
13    Tip : Le llaman el planeta rojo
14  5 : JUPITER
15    Tip : Es el planeta más grande del sistema solar
16  6 : SATURNO
17    Tip : Es muy conocido por sus bellos anillos
18  7 : URANO
19    Tip : Es el tercer planeta más grande después de Júpiter y Saturno
20  8 : NEPTUNO
21    Tip : Un personaje con Tridente en la caricatura de Bob Esponja tiene su
22    mismo nombre
23  9 : PLUTON
24    Tip : Es el planeta más alejado del sol
25
26 TEMA : Animales Domésticos y de Granja
27 Num_pal : 10
28   1 : Perro
29     Tip: Se caracteriza por su lealtad
30   2 : Gato
31     Tip: A veces comen ratones
32   3 : Burro
33     Tip: Es muy fuerte y resistente para llevar cargas pesadas
34   4 : Gallina
35     Tip: Amenudo comes algo que sale de su colita D:
36   5 : Hamster
37     Tip: Son muy pequeños y les gusta correr en su rueda
38   6 : Vaca
39     Tip: Produce una bebida muy nutritiva
40   7 : Borrego
41     Tip: Muchas chamarras son calientitas gracias a él
42   8 : Caballo
43     Tip: Cuando es libre vive en manadas y corre mucho
44   9 : Perico
45     Tip: Aprenden a repetir palabras
46  10 : Alpaca
47     Tip: Se dice que el camello es su parente cercano
48

```

Figura 13. Archivo de Configuración de temas

- La primer línea debe contener una etiqueta seguida de dos puntos y el dígito representando el número de Temas.
- La siguiente etiqueta representa el nombre del tema cuyo valor se separa por dos puntos.
- Seguido del número N de palabras del tema representado por su etiqueta y valor separado pos dos puntos
- Dos por N renglones siguen, dos por cada palabra, el primero representa la palabra y el segundo el tip, cada uno con su etiqueta y valor separado por dos puntos
- Si el número de temas es mayor a uno cada tema debe cumplir con la especificación desde el punto b).
- El nombre de las etiquetas no es de importancia, se tomará el valor en el orden dicho, resumiendo.
  - Número n de temas
  - Nombre del n-ésimo temas
  - Número m de palabras del n-ésimo tema.
  - Seguidos de Dos por m renglones

1. m-ésima palabra
  2. m-ésimo tip (palabra y tip en renglones separados)
8. Para correr el programa el usuario entrara en la terminal y se situará en donde se encuentra la carpeta con los archivos mencionados. Por ejemplo en nuestro caso los archivos se encuentran en la carpeta llamada Final que se encuentra en el directorio Documentos, por tanto ingresaremos a este con el comando “cd“ seguido del directorio documentos separado por un espacio y después el nombre de nuestra carpeta Final separado por una diagonal y damos enter. Un ejemplo se muestra en la Figura 14.



A screenshot of a terminal window titled "geovanni@Gio: ~/Documentos/Final". The window has two tabs: "...documentos/Final" and "geovanni@Gio:~/Documentos/Final". The main pane shows the command "geovanni@Gio:~\$ cd Documentos/Final/" followed by a cursor. The status bar at the bottom indicates the current directory is "geovanni@Gio:~/Documentos/Final\$". To the right of the terminal, there is some code in a code editor:

```
int main()
{
    printf("Hola mundo");
}
```

Figura 14. Accediendo a la carpeta Final.

9. Ahora procedemos a compilar el archivo principal del proyecto este es el Servidor.c, escribiendo el siguiente comando y damos enter.

**\$ gcc servidor.c**

10. Por último en forma sincronizada los usuario deberán ejecutar el programa con el comando de ejecución por default seguido del puerto al cual se van a conectar. Por ejemplo:

**\$ ./a.out 4350**

11. Ahora si ya puede comenzar el juego comenzando con la elección del tema por parte del moderador(máquina que fungo como servidor de la red).

- **Conclusión**

En este proyecto se demostraron los conocimientos adquiridos durante el curso de Sistemas Distribuidos, y aquellos de previos cursos como son Sistemas Operativos y Redes de Computadoras. Las computadoras trabajan en una red distribuida pues todas están conectadas entre sí, la implementación del la red de anillo y el paso de token fue internamente, debido a que cualquier computadora podía fungir como un cliente del juego o como coordinador, todo ello dependía de su ID asociado. Se decidió trabajar de esta manera por el hecho de facilitar la comunicación entre ellas, por ejemplo, cuando un cliente (jugador tenía el token) los demás clientes y coordinador debían conocer quien tenía el token actualmente, algo un poco más difícil si un cliente solo estuviera conectado con su sucesor, resumiendo este problema a un mensaje estilo broadcast.

## Apéndice A

En este apartado te mostramos como hacer la instalación del sistema operativo Elementary. Los pasos a seguir son los siguientes:

1. Descargar la ISO de elementary OS en la siguiente dirección electrónica [elementaryos.org](http://elementaryos.org), o alguna otra distribución.
2. Creamos un disco de arranque USB, desde tu sistema operativo Windows descarga la siguiente herramienta [Yumi Multiboot USB Creator](http://www.yumiusb.com/) , con esta herramienta a seleccionas el ISO que descargaste y la herramienta hará el resto.
3. Ahora reinicia tu computadora y entra a opciones de la BIOS , busca las opciones de Boot y elige el dispositivo de arranque en este caso USB.
4. Selecciona Install Elementary OS to a Hard Disk esto es, instalar elementaryOS en el disco duro, y espera que cargue.
5. Elige el idioma y selecciona instalar elementary OS.
6. Si se quiere instalar solo elementary OS simplemente eligen la opción borrar todo el disco , el instalador hace todo lo demás.
7. Si tienen instalado Windows 7, el instalador detectará la partición y te preguntará si deseas instalar elementary OS junto a Windows, una vez seleccionada la opción, te pasara a una pantalla donde elegirás el espacio que deseas que ocupe cada sistema operativo en tu disco duro.
8. Despues de decidir los tamaños de las particiones, presiona instalar.
9. A continuación te preguntará sobre Actualizaciones y Software de terceros, tu decides si seleccionarlos o no.
10. Ahora tendrás que seleccionar la distribución de teclado, tu ubicación y por ultimo tu información personal.
11. Listo ahora solo tendrás que esperar unos minutos después te pedirá retirar el USB y con un enter la computadora se reiniciara y tendrás instalado tu sistema operativo Elementary OS.

- **Referencias**

- Programación básica de sockets en UNIX.  
<http://es.tldp.org/Tutoriales/PROG-SOCKETS/prog-sockets.html>  
[Última visita 12 febrero del 2015]
- Sistemas Distribuidos / Principios y Paradigmas, TANENBAUM, A / VAN STEEN, M, 9702612802, Editorial PEARSON