Guía del Programador
El simulador de protocolos de nivel de enlace

Aspectos generales

El simulador de protocolos de nivel de enlace está compuesto por dos programas. Uno de los programas es el simulador propiamente dicho, mientras que el otro programa es una interfaz gráfica para el simulador.

La comunicación entre los dos programas se realiza mediante ficheros. La interfaz gráfica genera un archivo de texto, llamado fichero de entrada, a partir de los parámetros que el usuario configura para realizar la simulación y ejecuta el simulador pasándole este archivo como argumento. El simulador al ejecutarse genera otro archivo, el fichero de salida, con los datos de la simulación que es leído por la interfaz, la cual presenta estos datos al usuario. La estructura de estos archivos se encuentra detallada en la Guía de Usuario.

El simulador está escrito un lenguaje C. El código se ha programado utilizando funciones pertenecientes al estándar ANSI para que sea portable. Para el desarrollo y depuración del código se ha utilizado el entorno de desarrollo proporcionado por las herramientas Cygwin. Las herramientas Cygwin son un conjunto de utilidades para Windows que emplean la librería Cygwin, la cual proporciona las llamadas al sistema y el entorno Unix que estas utilidades necesitan. Éstas herramientas, entre otras, incluyen un compilador (gcc), un editor (Bloodshed Dev-C++) y un depurador (Insight).

El código C creado puede ser ejecutado bajo sistemas Windows. Para ello ha de ser compilado con el compilador gcc de Cygwin empleando el makefile correspondiente. El archivo ejecutable resultado de la compilación puede ser ejecutado directamente por Cygwin o fuera de Cygwin utilizando la librería cygwin1.dll.

La interfaz gráfica está programada en lenguaje Visual Basic utilizando el entorno de desarrollo proporcionado por Microsoft Visual Studio. La interfaz ha de ser compilada con el propio entorno de Visual Basic. Con este entorno se puede crear un programa de instalación que además de la propia interfaz instala las librerías y los componentes necesarios para la ejecución de la misma. La interfaz gráfica solo puede ser ejecutada bajo entornos Windows.

El simulador

El simulador consta de 5 hilos independientes. El hilo principal, llamado Main, da lugar a otros cuatro hilos hijos, dos hilos para los nodos, M0 y M1, y otros dos pertenecientes a los canales que unen ambos nodos, C0 y C1. Para comunicar estos 5 hilos independientes se crean doce pipes cuyos descriptores de fichero se describen a continuación:

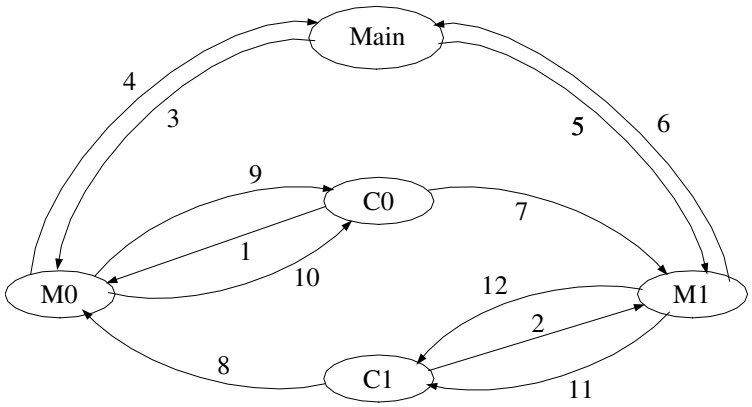

Ilustración 1. Esquema de hilos y pipes del simulador

- Comunicación Main – M0:
  - w3, r3: señal de continuación de main a M0
  - w4, r4: señal de preparado de M0 a Main

- Comunicación Main – M1:
  - w5, r5: señal de continuación de main a M0
  - w6, r6: señal de preparado de M0 a Main

- Comunicación M0 – C0:
  - w1, r1: tramas de M0 a C0
  - w9, r9: señal de continuación de M0 a C0
  - w10, r10: estado del transmisor de C0 a M0

- Comunicación M1 – C1:
  - w2, r2: tramas de M1 a C1
  - w11, r11: señal de continuación de M1 a C1
  - w12, r12: estado del transmisor de C1 a M1

- Comunicación C0 - M1:
  - w7, r7: tramas de C0 a M1

- Comunicación C1 - M0:
  - w8, r8: tramas de C1 a M0

El hilo principal, Main, es el encargado de leer el fichero de entrada donde se recogen los datos necesarios para la simulación y de escribir en el fichero de salida tanto la cabecera como los últimos datos. Otras funciones del hilo principal son sincronizar todos los hilos y monitorizar el estado de la simulación para detectar bucles infinitos o el fin de la misma.

Los hilos de los nodos ejecutan el código perteneciente al protocolo. Este código se ha dotado de un gran número de parámetros para que el comportamiento del protocolo sea muy flexible y pueda adaptarse a las necesidades del usuario. Este hilo realiza las funciones correspondientes al nivel de enlace de un nodo de comunicaciones.

Los hilos de los canales ejecutan el código perteneciente al transmisor y al canal. Este hilo lee las tramas que le envía su nodo correspondiente, las transmite, las propaga y las entrega al nodo destino.

1 El hilo principal: Main

El código perteneciente al hilo principal se encuentra en los ficheros sim.c y parametros.c. Se compone de una función principal, main(), la cual llama a las otras funciones parse_args(), set_up_pipes(), fork_off_workers() y terminate().

La primera función que se ejecuta es main(). Al comienzo de esta función se declaran e inicializan las principales variables del simulador. Estas variables son:

- *tick*: tick actual de la simulación
- *rwfd[2], wwfd[2]*: descriptores de fichero para comunicarse con los nodos
- *word*: almacena los mensajes para y de los nodos
- *hanging[2]*: numero de ticks que un nodo está inactivo
- *args*: estructura donde se guardan los parámetros de la simulación
- *p*: estructura donde se almacenan los pipes para la comunicación
- *fin_ficheros*: cuando esta variable alcanza valor 2 ambos nodos se han quedado sin nada que transmitir y se supone que se ha entrado en un bucle infinito
- *duracion*: al final de la simulación almacena la duración en tiempo real de la simulación

Tras declarar e inicializar estas variables, el siguiente paso es analizar los parámetros que proporciona el usuario. El usuario, en lugar de pasar los parámetros por línea de comandos, los entrega mediante un fichero de texto.

Para analizar este fichero, la función main() llama a la función parse_args(), pasándole como argumentos los mismos argumentos que tiene la función main(). De estos argumentos se obtiene el nombre y la ruta del fichero de entrada.

Para comenzar, parse_args() comprueba que se le halla pasado en los argumentos el nombre de un fichero que existe y si es así procede a leerlo, asignándole un descriptor de fichero.

Los parámetros que lee esta función los va escribiendo en la variable *args*. Esta variable es una estructura que está formada a su vez por otras 5 estructuras. En ella se almacenan todos los parámetros relativos a la simulación.

El fichero de entrada se va leyendo palabra a palabra mediante la función avanza(), que devuelve la siguiente palabra presente en el fichero, sin tener en cuenta los comentarios, que empiezan siempre por el carácter almohadilla (#).

Para conocer cual es el parámetro que se va a leer, se compara la palabra leída con las palabras clave que pueden aparecer en el fichero. Para ello se utiliza la función iguales(), a la que se le pasan dos cadenas de texto (constantes) y devuelve un 1 si las dos cadenas son exactamente iguales.

La función parse_args() recurre a las funciones parse_sim(), parse_nodo() y parse_canal() para leer los parámetros generales de la simulación, los parámetros de los nodos y los parámetros de los canales respectivamente.

Una vez leído completamente el fichero de entrada, se sustituye la salida estándar por el fichero de salida (ya creado). Así, cada vez que se realice un printf() la cadena generada se guardará en el fichero de entrada, y si éste no se ha especificado, se imprimirá en pantalla.

Tras esto, se realizan algunos cálculos necesarios para la simulación, como, por ejemplo, el cálculo del número máximo de secuencia, que ha de cumplir las dos condiciones siguientes:

$$\text{ventana\_mayor\_tamaño} \leqslant \frac{(\text{max\_seq}+1)}{2} \quad \text{y} \quad \text{max\_seq} = 2^n - 1 \quad \text{con} \quad n \in \mathbb{N}$$

Para realizar este cálculo utiliza la función calcula_max_seq() que a su vez utiliza la función exp2(i) que devuelve la i-ésima potencia de 2.

Por último, la función parse_args() presenta un resumen de los parámetros de entrada que hace las veces de cabecera del fichero de salida.

Una vez analizado este fichero y extraídos los parámetros de la simulación, el hilo principal crea los doce pipes necesarios con la función set_up_pipes() y los guarda en la estructura *p*.

Lo siguiente es llamar a la función fork_off_workers(). En esta función se crean los otros 4 hilos hilos: M0, M1, C0 y C1. El proceso de creación de cada hilo es el siguiente: primero se bifurca el hilo actual mediante la orden fork(), en uno de los hilos que surgen se cierran los extremos de los pipes que no se van a utilizar, y los otros se asignan a la variable *fd* que es una estructura que almacena todos los descriptores de fichero que ha de utilizar un hilo, y por último se llama al código que ha de ejecutar el hilo, que para los hilos M0 y M1 es la función protocol() y para los hilos C0 y C1 es la función canal().

Una vez creados todos los hilos hijos, el hilo principal entra en el bucle que ejecutará hasta el final de la simulación. Cada ejecución del bucle es un paso en el tiempo del simulador, es un tick.

Al comienzo del bucle, el hilo principal lee el estado de uno de los nodos. La respuesta puede ser OK, que indica que todo continúa bien, NOTHING, que indica que no se ha realizado ninguna acción, o END_FILE, que indica que ese nodo ha recibido del otro nodo la señal de fin de fichero. Cuando los dos nodos responden NOTHING demasiadas veces, se supone que se ha entrado en un bucle infinito y se termina la ejecución de la simulación. Cuando los dos nodos responden END_FILE, también se termina la simulación, pues ya no hay más datos que transmitir. Por último se envía al nodo el tick actual, que es la señal de continuar. Todo esto se repite para el otro nodo y se repite el bucle hasta que se alcance el tick estipulado como final de la simulación.

Al salir de este bucle se cierran todos los descriptores de fichero utilizados y se calcula en tiempo real la duración de la simulación. Antes de terminar la ejecución del hilo principal se presenta un informe, mucho más extenso que el del simulador de partida, en el que se desglosan todos los datos sobre lo ocurrido durante la simulación. Esto lo realiza la función terminate() que finaliza la simulación enviando a cada nodo la señal de fin, además lee de cada nodo los datos estadísticos necesarios para calcular algunos resultados y les da tiempo a escribir sus datos en el fichero de salida. No es necesario leer datos de los nodos y canales pues estos escriben todos sus datos pero se conserva el código por si es de utilidad para uso futuro.

2 Los hilos de los nodos: M0 y M1

El código perteneciente al protocolo que se va a ejecutar se encuentra en los ficheros p.c. Pero éste código se apoya en las numerosas funciones contenidas en el fichero worker.c. Al iniciar el protocolo, lo primero es declarar e inicializar todas las variables empleadas. La descripción de estas variables aparece en el propio código. Por ejemplo, hay que declarar e inicializar la variable stat que es una estructura en la que se guardan todas las estadísticas. Esta variable se inicializa a cero con la función inicia_estadisticas(). Después se elige la semilla a utilizar por las funciones de generación de números aleatorios. La semilla es diferente para los dos nodos, para que sigan patrones aleatorios diferentes.

Tras esto se entra en el bucle que se va ha seguir hasta el final de la simulación. El primer paso dentro del bucle es conocer cual va ha ser el siguiente evento a procesar. Los eventos posibles son:

- network_layer_ready: el nivel de red tiene un paquete preparado
- frame_arrival: ha llegado una trama sin error
- chksum_err: ha llegado una trama con error
- timeout: ha expirado uno de los temporizadores principales
- ack_timeout: ha expirado el temporizador de asentimiento

Si el evento a procesar es network_layer_ready:

- ✔ se expande la ventana de transmisión,
- ✔ se coge el nuevo paquete y se procesa,
- ✔ lo almacena en el buffer de transmisión, en la posición next_frame_to_send,
- ✔ guarda en que tick se comenzó a procesar el nuevo paquete,
- ✔ crea una nueva trama con el nuevo paquete,
- ✔ y avanza el extremo superior de la ventana de transmisión.

Si el evento es frame_arrival:

- ✔ coge la nueva trama y la almacena en r
- ✔ si la trama es de datos y correcta:
    - ✔ si llega fuera de secuencia (su numero de secuencia es distinto de frame_expected):
        - ✔ se genera un nack, solo uno por trama
        - ✔ se manda la nueva trama al nivel fisico
    - ✔ si la trama llega en secuencia comienza el temporizador de asentimiento
    - ✔ si el número de secuencia de la trama está dentro de la ventana de recepción se acepta:
        - ✔ actualiza las estadísticas, primero el retardo y de cuantas tramas se ha contabilizado el retardo
        - ✔ marca la posición del buffer de recepción como ocupada
        - ✔ inserta el paquete en el buffer
        - ✔ actualiza el numero de bytes recibidos
        - ✔ si están dispuestos, pasa los paquetes al nivel de red en orden:
            - ✔ saca el paquete del buffer de recepción
            - ✔ pasa el paquete al nivel de red
            - ✔ permite que vuelvan a enviarse nacks
            - ✔ avanza el extremo inferior de la ventana de recepción
            - ✔ avanza el extremo superior de la ventana de recepción
            - ✔ si aún no ha comenzado el temporizador. de asentimiento, lo inicia
            - ✔ si era el último paquete del fichero lo indica
- ✔ si la trama es un nack y está dentro de la ventana de recepción
    - ✔ genera de nuevo la trama que pide el nack
    - ✔ manda la trama al nivel físico (al canal)
    - ✔ procesa el campo de asentimiento de la trama recibida si el asentimiento está dentro de los esperados
        - ✔ saca una trama del buffer de transmisión
        - ✔ para el temporizador de esa trama
        - ✔ actualiza las estadísticas de retardo medio de ida y vuelta
        - ✔ avanza el extremo inferior de la ventana de transmisión

Si el evento es chksum_err:

- ✔ si se pueden mandar nacks mando uno, pero solo un nack por trama
  - ✔ genera el nack
  - ✔ manda el nack al nivel físico
- ✔ si no se pueden mandar nacks no se hace nada

Si el evento es timeout:

- ✔ genera de nuevo la trama a retransmitir
- ✔ manda la trama al nivel físico

Si el evento es ack_timeout:

- ✔ genera una trama de asentimiento
- ✔ manda la nueva trama al nivel físico

Tras procesar el evento, se comprueba si el numero de tramas en el buffer de transmisión es menor que el tamaño del buffer y aún no se ha llegado al final del fichero a enviar, entonces se habilita el nivel de red. Si alguna de las dos condiciones no se cumple, se deshabilita el nivel de red, que no podrá generar más tramas.

Finalizados estos pasos se vuelve al comienzo del bucle.

La creación de una nueva trama se realiza mediante la función new_frame() que rellena una estructura frame con los datos que se le pasan como argumento. Para mandar una trama al nivel físico, se le pasa a la función send_frame() esta función introduce la trama en una lista doblemente enlazada a la espera de ser enviada al canal cuando el transmisor esté libre, además se actualizan los temporizadores.

Al comienzo del bucle se llama a la función wait_for_event(). La finalidad de esta función es devolver el siguiente evento a procesar por el protocolo y mantener el sincronismo tanto con el hilo principal como con el canal que le corresponde.

Mientras no se produzca ningún evento, se comprueba si se han recibido nuevas tramas del otro nodo (realmente del otro canal), el estado del transmisor (que se encuentra en el hilo del canal) y si hay tramas esperando ser transmitidas. Tras esto, se le envía al hilo principal un mensaje en el que se indica el estado y que puede ser:

- ➔ OK: el protocolo realiza su función normalmente
- ➔ NOTHING: no se ha producido ningún evento ni hay ningún temporizador en marcha durante el último tick.
- ➔ END_FILE: se ha recibido del nodo par la señal de que ha terminado de transmitir todos sus datos.

Tras enviar este mensaje lee el tiempo del hilo principal, en realidad lee un integer de 4 bytes con el tick actual. Este mismo mensaje se lo manda a su canal para mantenerlo sincronizado. También lee del canal un mensaje que le indica si el canal está libre y puede mandarle una nueva trama en caso de tener alguna a la espera de ser transmitida.

Una vez realizadas las sincronizaciones elige el siguiente evento a procesar mediante la función pick_event(). Si no hay ningún evento que procesar se repite toda la secuencia de sincronismo y se vuelve a llamar a pick_event(), todo esto sin devolver el control al protocolo, lo que no se hará hasta que pick_event() devuelve un evento no nulo. El evento devuelto es el mismo parámetro que wait_for_event() devuelve al código del protocolo para que éste continúe.

La función pick_event() comprueba los eventos posibles y entre ellos elige el próximo en ser procesado, según un riguroso orden. Los eventos posibles son, en orden de prioridad:

1. ack_timeout
2. chksum_err
3. frame_arrival
4. network_layer_ready
5. timeout

Según los parámetros de la simulación y la situación instantánea de la ejecución habrá eventos que no puedan producirse. Por ejemplo, para saber si puede producirse el evento timeout se llama a la función check_timers() que recorre todos los temporizadores y compara su valor con el del tick actual. Si para algún temporizador coinciden estos valores, ese temporizador ha vencido y puede producirse un evento timeout.

Para comprobar si han llegado nuevas tramas del otro nodo se emplea la función queue_frames() y si es así las mete en cola que es un buffer circular. En el programa original, para leer el pipe que comunica el hilo actual con el canal por el que le llegan las tramas, se utilizaba la función fstat() que devolvía el estado de un pipe, en particular su tamaño, para comprobar si se había escrito algo. Pero esta función aunque está incluida en las librerías de Cygwin, no funciona, por lo que se ha recurrido a un camino alternativo para comprobar si se ha escrito algo en el pipe. Lo que se hace es, antes de leer el pipe se introduce en él una trama de tipo end, que es la bandera; después se lee el pipe hasta leer la bandera, que es el final del pipe. Esto se hace así porque si se lee de un pipe vacío el programa se queda colgado esperando.

También hay que resaltar que pueden procesarse varios eventos antes de que se produzca una nueva secuencia de sincronismo, es decir, en el mismo tiempo o tick de la simulación. Este aspecto se representa en el siguiente esquema:

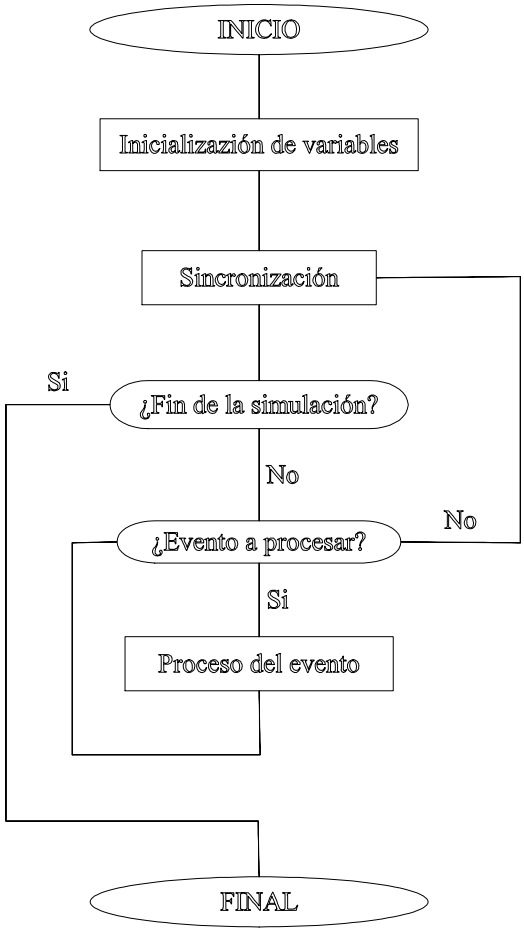

Ilustración 2. Esquema lógico del funcionamiento del protocolo

Una vez que se determina que ha llegado una nueva trama hay que comprobar si la trama tiene errores o se ha recibido correctamente. Esto es cometido de la función frame_type(). Para ello se obtiene un número aleatorio con la función rand() y éste número se compara con el porcentaje de error que el usuario ha definido para ese canal. Según el número sea menor o mayor a ese porcentaje, se trama habrá sufrido errores o no. En esta función además se actualizan las estadísticas correspondientes y se imprime la traza conveniente.

La función from_network_layer() es la encargada de recoger los paquetes nuevos procedentes de la capa de red. En realidad lo que hace esta función es leer una cantidad de terminada de caracteres del fichero designado como fichero a enviar por el nodo. En el paquete también se incluye un número de secuencia del paquete para poder comprobar en el destino que los paquetes se reciben correlativamente.

La función to_network_layer() entrega la información contenida en una trama recibida al nivel de red y hace una comprobación para ver si el paquete está en secuencia, sino la simulación se aborta con un mensaje de "error de protocolo".

El inicio de un temporizador lo realiza la función start_timer(). Esta función añade al tick actual el numero de tick necesarios según los parámetros del usuario para calcular el tick en el que ha de vencer el temporizador. Este valor se almacena en la estructura correspondiente.

La acción contraria la lleva a cabo la función stop_timer(), que para un temporizador poniendo en su variable un 0 indicando que no he de expirar.

Cada vez que se modifican los temporizadores hay que llamar a la función recalc_timers(). Esta función recorre todos los temporizadores y nos dice cuál es el que está más cerca de expirar.

Las mismas acciones se realizan con el temporizador de asentimiento con las funciones start_ack_timer() y stop_ack_timer().

Para habilitar o deshabilitar la capa de red se recurre a las funciones enable_network_layer() y disable_network_layer(). Realmente estas funciones solo cambian el valor de la variable *network_layer_satus* y según esta variable, podrá o no producirse el evento network_layer_ready.

Cada vez que ocurre un hecho significativo durante la simulación, según la traza elegida por el usuario se creará una nueva línea en el fichero de salida. La función print_traza() es la encargada de crear e imprimir esta traza. La cadena que se imprime depende del tipo de traza, del número de nodo, de los datos de la trama y de las estadísticas almacenadas hasta el momento. En el programa original la traza se limitaba a ser un simple mensaje que mostraba datos del transcurso del programa. Ahora además de mostrar datos se prende dar una información algo “gráfica" del tipo de traza que se muestra y de que hilo la produce. Para obtener en una cadena los datos de la trama se utiliza la función fr().

Antes de finalizar la ejecución, se presenta un resumen bastante amplio de lo ocurrido durante la simulación. Alguno de estos datos son pasados al hilo principal y otros datos son leídos del canal mediante el correspondiente descriptor de fichero. Esta tarea la realiza la función print_statistics(). Además ha de realizar algunos cálculos antes de imprimir las estadísticas y por último finaliza la ejecución del hilo.

3 Los hilos de los canales: C0 y C1

El código perteneciente a los canales se encuentra en el archivo canal.c. El canal se encarga de la transmisión y propagación de las tramas. Recibe la señal de sincronismo (un integer de 4 bytes con el tick actual) del nodo que le corresponde y le responde a éste con el estado del transmisor. También recibe de su nodo las tramas que envía al otro nodo una vez han sido transmitidas y propagadas.

El comienzo de su ejecución es la función canal(). Lo primero es inicializar las variables. Después entra en el bucle que ejecutará hasta que se termine la simulación.

Al principio del bucle realiza la sincronización con su nodo correspondiente, recogiendo el tick actual y enviado el estado del transmisor con la función sincroniza(). Si el tick leído es 0, es la señal de fin de simulación, entonces se manda el uso del transmisor al nodo para los cálculos y se finaliza.

El estado del transmisor se obtiene observando la lista doblemente enlazada que contiene todas las tramas en están en el canal. Cada nodo de la lista contiene una trama, un tick en el cual la trama tiene que cambiar de estado, y los punteros a los nodos anterior y siguiente. El estado de la trama puede ser transmitiéndose o propagándose. Si no hay ninguna trama transmitiéndose, el transmisor está libre. El puntero trans apunta a la trama que se está transmitiendo o a NULL su el transmisor está libre.

Tras la sincronización se comprueba si hay tramas que haya enviado el nodo para ser transmitidas, si es así las mete en la lista doblemente enlazada. Esto lo realiza la función nuevos().

El siguiente paso es examinar el estado del transmisor. Si éste está libre y hay tramas esperando ser transmitidas empieza su transmisión la primera trama a la espera. Si termina la transmisión de alguna trama, ésta pasa a propagarse y se introduce otra trama en el transmisor en caso de que haya alguna a la espera. Esto se hace en la función transmisor().

Por último con la función comprueba_ultimo() se comprueba si hay alguna trama propagándose. Si es así y además termina su propagación, esta trama se envía al otro nodo mediante el correspondiente descriptor de fichero. Y se repite el bucle.

Si en el proceso de sincronización, el tick recibido es 0, indica el final de la simulación. En este caso, simplemente se finaliza la ejecución.

La interfaz gráfica InGraSE

La interfaz gráfica InGraSE se desarrolló de manera independiente al simulador. Su finalidad es facilitar la creación de los ficheros de entrada del simulador y la lectura de los ficheros de salida. Posteriormente se le sumó otra función que es la de añadir la posibilidad de realizar baterías de simulaciones y presentar sus resultados mediante gráficas.

La interfaz está programada en lenguaje Visual Basic por su potencia gráfica, su sencillez y facilidad de aprendizaje. Se basa en una interfaz de múltiples documentos (MDI) que consta de un formulario principal y de tantos formularios secundarios como ficheros de simulaciones estén abiertos, además de otros formularios adicionales empleados para realizar funciones añadidas.

1 El formulario principal. frmMain

El formulario principal (frmMain) consta tan solo de un menú y de una barra de herramientas. Desde este formulario sólo se pueden realizar las funciones de abrir un fichero de simulación o de crear uno nuevo, personalizar el formulario principal mostrando u ocultando el menú y la barra de herramientas y mostrar la ventana Acerca de.

El código del formulario principal se encarga de habilitar y deshabilitar los botones de la barra de herramientas según sea conveniente. Al iniciar la ejecución del programa se crean tres matrices redimensionables:

- Document(): es una matriz de formularios secundarios. Almacena todos los formularios secundarios abiertos para poder llevar un control sobre ellos.
- Fstate(): es una matriz de estructuras FormState. Esta estructura contiene información sobre el estado del correspondiente formulario secundario, como si ha sido modificado o si se ha borrado.
- Sims(): es una matriz de objetos Simulacion. En estos objetos se almacenan todos los datos relativos a la simulación asociada al correspondiente formulario secundario, incluyendo los parámetros de la simulación y sus resultados si ya ha sido simulada.

Al descargar el formulario principal se almacenan en el registro los valores relativos a la posición y tamaño del formulario. Estos valores se leen del registro al cargar el formulario.

Desde este formulario solo se puede crear una nueva simulación o abrir una ya existente (leyendo un fichero de entrada). Si se selecciona abrir una simulación existente, se utiliza el dlgCommonDialog que es un objeto que presenta al usuario un cuadro de diálogo en el que el usuario puede navegar por el árbol de directorios para

buscar el fichero que quiere abrir. Este objeto también lo utiliza el formulario secundario para ejecutar la orden Guardar como.

2 El formulario secundario. frmDocument

Los formularios secundarios aparecen cuando se abre una simulación o se crea una nueva. En un formulario secundario se pueden especificar los parámetros de una simulación de una manera más cómoda que con el fichero de entrada. Además, desde un formulario secundario se pueden guardar estos parámetros a un fichero de entrada, s puede lanzar una simulación llamando al simulador con los parámetros que aparecen en el formulario y se pueden realizar baterías de simulaciones.

Para llevar un control de los formularios secundarios abiertos se utilizan las matrices dinámicas ya comentadas. Al abrir un fichero de simulación o crear uno nuevo, se añade un nuevo elemento a cada matriz y cuando se cierra un formulario secundario se busca el elemento de la matriz que pertenece a ese formulario y se elimina. Así, cuando se vaya a realizar una gráfica con los formularios abiertos solo hay que recorrer la matriz y obtener sus datos.

Cada formulario secundario está asociado a un objeto Simulación. Un objeto simulación se compone a su vez de un objeto Result y de un objeto Param además de las variables ID, que es un índice, y Realizada, que es una variable booleana que toma el valor true si ya se ha realizado la simulación y los resultados están disponibles.

A su vez, un objeto Param se compone de un objeto General, dos objetos Nodo y dos objetos Canal además de una variable llamada Archivo donde aparece el nombre del fichero de entrada perteneciente a esta simulación. El objeto Param también cuenta con algunas propiedades:

- Leer(): mediante esta propiedad se puede leer un fichero de entrada y almacenar sus parámetros en el objeto Simulación. Se apoya en las funciones LeerSimulacion(), LeerPalabra(), LeerCanal() y LeerNodo().
- Escribir(): mediante esta propiedad se puede generar un fichero de entrada con los valores de los parámetros almacenados en el objeto Simulacion.

Copia(): esta propiedad realiza una copia de todos los parámetros de simulación de un objeto Simulacion a otro. Se utiliza principalmente a la hora de realizar baterías de simulaciones.

Los objetos General, Nodo y Canal solo contienen variables en las que almacenar los parámetros generales, del nodo y del canal de la simulación respectivamente.

Un objeto Result está compuesto por dos objetos ResNod y otros dos objetos ResCan además de una variable Archivo donde aparece el nombre del fichero de salida asociado a la simulación y otras variables que almacenan los resultados globales de la simulación.

El objeto Result también dispone de un método Leer() mediante el cual lee el fichero de salida generado por el simulador y almacena los datos leídos en el propio objeto.

Los objetos ResCan y ResNod solo contiene los variables que almacenan los resultados de la simulación relativos a los canales o a los nodos.

Cada control TextBox en el que el usuario puede introducir datos tiene un código asociado para validar el valor introducido y que no salga fuera del rango de datos permitido por el simulador.

El formulario secundario cuenta con la función PresentaParam() que lee los parámetros de la simulación de un objeto simulación y los presenta en el formulario. También dispone de la función GeneraParam() que rellena las variables del objeto Param con los datos introducidos por el usuario en el formulario.

Desde un formulario secundario se puede lanzar el simulador para realizar una simulación. Al hacer esto, InGraSE llama al simulador y le pasa los parámetros de la simulación que aparecen en el formulario. Al terminar la simulación InGraSE presenta una ventana con los resultados de la simulación realizada. Estos datos se obtienen del fichero de salida que genera el simulador. Todo este proceso lo realiza la función Simular().

3 El formulario de batería de simulaciones. frmBat

Desde un formulario secundario se puede lanzar una batería de simulaciones. Ésta consiste en realizar varias simulaciones sobre los mismos parámetros de simulación variando tan solo uno de ellos. Con ello podemos observar cómo se comporta el protocolo ante la variación del parámetro elegido. Para lanzar una batería de simulaciones hemos de seleccionar la opción de menú Herramientas > Batería de simulaciones. Al hacer esto se abre un nuevo formulario donde podemos seleccionar el parámetro sobre el que se va a realizar la batería, su valor inicial, el final y el paso entre los distintos valores.

Este formulario mediante la función ValidaPaso() comprueba que el valor del paso introducido sea coherente con los valores inicial y final. La función CompruebaNumeros() se encarga de que en los TextBox en los que hay que introducir valores numéricos no se puedan introducir letras.

La principal función de este formulario la realiza la función ComienzaBatería(). Esta función se encarga de ir generando los nuevos formularos secundarios como copia del formulario secundario base utilizado, de actualizar sus valores según el paso y de realizar las simulaciones pertinentes.

4 El formulario de resultados. frmRes

Cuando se ha realizado la simulación correspondiente al formulario secundario activo, se puede acceder al formulario de resultados. En este formulario se presentan los resultados obtenidos de la simulación, agrupados en resultados generales, de los nodos y de los canales. Para facilitar la comprensión de estos datos se han introducido algunos controles gráficos como son los ProgressBar y los MSChart.

Este formulario solo emplea una función, Mostrar(). Esta función lee los resultados del objeto Simulacion y según ellos le da valores a los distintos controles del formulario.

5 El formulario de resultados de la batería. frmResBat

Una vez realizada la batería de simulaciones se abre un nuevo formulario donde aparece una gráfica con los resultados de la simulación. En esta ventana se puede elegir que datos deben aparecer en la gráfica mediante una lista desplegable de los parámetros posibles. En esta ventana también se muestra la media aritmética de los valores que toma el parámetro seleccionado.

Si en la gráfica creada aparece algún valor que nos interese eliminar tan solo tenemos que buscar a que formulario secundario pertenece y cerrar éste. Después seleccionando la opción de menú Herramientas > Crear Gráfica se vuelve a generar la gráfica pero esta vez sin el valor que hemos eliminado.

Esto quiere decir que sean cuales sean los formularios secundarios que tengamos abiertos, al seleccionar Herramientas > Crear Gráfica se creará una gráfica tomando los valores de los formularios secundarios abiertos cuyas simulaciones se hayan realizado.

Otro aspecto a resaltar es que si tenemos abierto un formulario secundario y hemos realizado su simulación, al seleccionar Resultados en el menú accedemos a la ventana que nos muestra los resultados de la simulación.

El código que da valores a los controles de este formulario a partir de los datos de resultados de los formularios secundarios abiertos es la función CreaGrafica() perteneciente al formulario secundario activo. En esta función se van recorriendo los distintos formularios activos y de su objeto Simualcion se leen los resultados escogiendo el valor que después vamos a presentar en la gráfica.

La gráfica presentada es un control MSChart. A este control se le indican los datos que tiene que presentar mediante el método MSChart.Data pero antes hay que decirle ese dato en que posición exacta queremos mostrarlo, para ello empleamos los métodos MSChart.Column y MSChart.Row.

6 El formulario de traza. FrmTraza

A este formulario se accede desde un formulario secundario pulsando sobre el botón que aparece al lado del campo traza. Este formulario facilita la tarea de calcular valor deseado de este campo conociendo para que eventos queremos que se produzca una traza. El formulario presenta uno CheckBox para cada evento posible. Si marcamos este CheckBox indicamos que queremos que este evento produzca una traza.

Una vez seleccionados los eventos que queremos, al pulsar sobre el botón Aceptar se calcula automáticamente el valor que debe tener el campo traza según nuestros requerimientos. Este cálculo lo realiza la función OKButton_Click().