Programación con OSF RPC y Microsoft RPC

1- Entorno de desarrollo en Windows

Objetivo

Compilar y ejecutar la aplicación “HolaMundo” de ejemplo.

Ficheros necesarios: (archivo **hola-windows.zip**, en la web de la asignatura)

hola.idl:	Declaración del interfaz ¡Debe modificarse el uuid!
hola.acf:	Fichero de configuración
server.c:	Implementación del servicio (y de main)
client.c:	Implementación de un cliente de ejemplo

Se suministra también un **Makefile** (específico para Windows), para automatizar la creación de ejecutables, que no usaremos de momento.

Proceso de compilación

- **Copiar a un lugar accesible el fichero `vcvars32.bat` (normalmente disponible en**
  `C:\Archivos de Programa\`
  `\Microsoft Visual Studio .NET 2003\Vc7\bin`**)**
- **Abrir un interfaz de comandos y en él:**
  - **Ejecutar `vcvars32`**
  - **Ejecutar `uuidgen` y copiar el uuid generado en el interfaz**
  - **Compilar el interfaz, usando `MIDL /osf /Os`**
  - **Compilar cada fichero fuente por separado (cliente, servidor y stubs), usando `CL /c`**
  - **Enlazar los objetos adecuados y la librería `RPCRT4.LIB` para crear el ejecutable del cliente.**
  - **Enlazar los objetos adecuados y la librería `RPCRT4.LIB` para crear el ejecutable del servidor.**

Visual Studio .NET 2003

Ejecución de prueba y variaciones

- **En diferentes ventanas, lanzar servidor y clientes**
- **Añadir control de errores al cliente.**
- **Otros protocolos de comunicación:**
  - **Editar el fuente del cliente y cambiar el protocolo a *Local* (`"ncalrpc:"`). Rehacer el ejecutable. Probarlo.**
  - **Editar el fuente del servidor y añadirle el registro del protocolo *Local*. Rehacer el ejecutable. Probarlo.**
- ***Handle* implícito (sólo en el cliente)**
  - **Crear un archivo `hola_cliente.acf`, copia de `hola.acf` y modificarlo para que el *handle* sea implícito.**
  - **Modificar el cliente para que use *handle* implícito.**
  - **Generar de nuevo el stub de cliente, y rehacer el ejecutable**

```
MIDL [...] /acf hola_cliente.acf /server none hola.idl
```

2- Entorno de desarrollo en orion

Objetivo

Compilar y ejecutar la aplicación “HolaMundo” de ejemplo.

Ficheros necesarios: (archivo **hola-orion.tar.gz**, en la web de la asignatura)

hola.idl:	Declaración del interfaz. Igual al de MS
hola.acf:	Fichero de configuración. Igual al de MS
server.c:	Implementación del servicio (igual que en MS) y de main (diferente al de MS)
client.c:	Implementación de un cliente de ejemplo (diferente al de MS)

Se suministra también un **Makefile** (específico para orion), para automatizar la creación de ejecutables, que no usaremos de momento.

Proceso de compilación

- Modificar el interfaz (hola.idl) para incluir el mismo uuid que habíamos generado en la máquina Windows.
- Compilar el interfaz,
  `/opt/dce/bin/idl -keep c_source hola.idl`
- Compilar cada fichero fuente por separado (cliente, servidor y stubs), usando `gcc -I. -c`
- Enlazar los objetos y bibliotecas adecuados para crear el ejecutable del cliente:
  `gcc -o client client.o hola_cstub.o -L/opt/dce/lib -ldcerpc -ldcethread -lpthread`
- Enlazar los objetos y bibliotecas adecuados para crear el ejecutable del servidor.

Ejecución de prueba y variaciones

- **En diferentes ventanas, lanzar servidor y clientes.**
- **Modificar los clientes (orion y windows) para que pidan el *string binding* al usuario en lugar de tener uno prefijado.**
- **Ejecutar el cliente de orion de modo que conecte con el servidor de windows (protocolo tcp/ip).**
- **Ejecutar el cliente de windows de modo que conecte con el servidor en orion (protocolo tcp/ip)**
- **Añadir control de errores al cliente linux. Probar a ejecutarlo cuando el servidor no está corriendo.**
  - **Lista de *status* posibles en `/opt/dce/include/dce/rpcstat.idl`**

3- Promedio de lista

Objetivo

Escribir un ejemplo que transmita datos más complejos.

El servicio recibe un array adaptable (*conforming*), que contiene una serie de números reales. Devuelve su promedio o bien una cadena de error si el array no contiene elementos.

Hacer dos versiones de la función

- El array adaptable es un parámetro de la función,
  y su tamaño es otro parámetro
  (la función recibe dos parámetros).
- El array adaptable es un campo de una estructura
  y su tamaño es otro campo
  (la función recibe un único parámetro).

Tipo retornado

Especificación:

- El tipo retornado puede ser, bien un *string*, bien un *float*.
- Un discriminante de tipo entero indica el caso:

  -1 Hubo error (se retorna *string*)
  0 No hubo error (se retorna *float*)

Implementación

- El tipo retornado será una estructura. Un campo será una unión y otro el discriminante entero.

Tareas (a desarrollar bajo Windows)

- **Generar uuid para el nuevo servidor (probar `uuidgen /i`)**
- **Escribir interfaz: `promedio.idl`**
- **Escribir configuración: `promedio.acf` (usar handle implícito)**
- **Escribir servidor, que registre sólo el protocolo tcpip**
- **Escribir cliente que pida al usuario**
  - **Nombre de la máquina servidora**
  - **Número de elementos del array**
  - **Valor de cada elemento**
- **...y haga la invocación remota mostrando el resultado, según su tipo.**
- **Escribir `Makefile` y probar todo.**

Tareas (a desarrollar bajo linux, orion)

- **Transmitir a orion el interfaz escrito para Windows, así como el fichero de configuración.**
- **Adaptar el servidor windows**
- **Adaptar el cliente windows**
- **Escribir `Makefile` y probar todo.**
- **Hacer pruebas cruzadas (invocar cliente windows desde orion y viceversa)**