Tema 2:
La Información

Sistemas Distribuidos

2022-2023

Middleware. Formatos de representación

El problema de la API de sockets

- Tediosa de programar (control de errores)
- El problema de la posibilidad de `read` y `write` incompletos
- Los datos se transmiten en binario, tal como están en memoria

```
int j = 2013;
// se omite la inicialización del socket y su conexión
send(sock_datos, &j, sizeof(j));
```

```
int dato;
// se omite la inicialización del socket y su conexión
recv(sock_dat, &dato, sizeof(dato));
```

Soluciones: *Middleware*

- Capa de software intermedio
- Se ejecuta tanto en servidor como en cliente
- Accede a la API de sockets "por debajo"
- Pero codifica la información antes de enviarla
- Y proporciona otros servicios de infrastructura
- Que posibilitarán implementar las RPC

> Un ejemplo de este *middleware* es XDR

XDR

Qué es XDR (Xternal Data Representation)

- Estándar de codificación binaria *independiente de la arquitectura* Biblioteca de funciones "filtro" Lenguaje para declarar tipos nuevos Herramientas de generación automática de código

Filtros XDR

- Convierten tipos de datos *nativos* a un formato *independiente*
- Y viceversa
- Todos usan la misma sintaxis:

```
#include <rpc/types.h>
#include <rpc/xdr.h>
bool_t xdr_tipo(XDR *operacion, tipo *dato)
```

¿ `XDR *operacion` ?

- El tipo `XDR` es *opaco*
- No necesitamos acceder a sus campos
- Se inicializa y destruye mediante funciones de la biblioteca

```
#include <rpc/types.h>
#include <rpc/xdr.h>
bool_t xdrstdio_create(XDR *operacion, FILE *fichero,
                       enum xdr_op sentido)
```

```
#include <rpc/types.h>
#include <rpc/xdr.h>
void xdr_destroy(XDR *xdrs)
```

Ejemplo (emisor)

```c
// xdr-escribe-entero.c
#include <stdio.h>    // Para FILE*
#include <errno.h>    // para perror()
#include <stdlib.h>   // para exit()
#include <rpc/types.h>
#include <rpc/rpc.h>
int main() {
  int j = 2013;     // Dato a escribir
  FILE *fichero;    // Fichero donde se escribirá
  XDR operacion;
  fichero=fopen("datos.xdr", "w");  // Abrir para "w"rite
  if (fichero==NULL) {              // Comprobar errores
      perror("Al abrir fichero");
      exit(1);
  }
  // Inicializar variable operacion para filtros subsiguientes
  xdrstdio_create(&operacion, fichero, XDR_ENCODE);
  // Escribir la variable j en el fichero, en representacion externa
  xdr_int(&operacion, &j);   // Llamada al filtro. Codifica y guarda
  // Terminado, labores finales "domésticas"
  xdr_destroy(&operacion);   // Destruir la variable operacion
  fclose(fichero);           // Cerrar fichero
  return 0;
}
```

Ejemplo (receptor)

```c
// xdr-lee-entero.c
#include <stdio.h>    // Para FILE*
#include <errno.h>    // para perror()
#include <stdlib.h>   // para exit()
#include <rpc/types.h>
#include <rpc/rpc.h>
int main() {
  int dato_leido;   // Dato a leer
  FILE *fichero;    // Fichero de donde se leerá
  XDR operacion;
  fichero=fopen("datos.xdr", "r");  // Abrir para "r"ead
  if (fichero==NULL) {              // Comprobar errores
      perror("Al abrir fichero");
      exit(1);
  }
  // Inicializar variable operación para filtros subsiguientes
  xdrstdio_create(&operacion, fichero, XDR_DECODE);
  // Leer sobre la variable dato_leido
  xdr_int(&operacion, &dato_leido);   // Lee y decodifica
  // Terminado, labores finales "domésticas"
  xdr_destroy(&operacion);   // Destruir la variable operacion
  fclose(fichero);           // Cerrar fichero
  // Imprimir el valor leido:
  printf("Dato leido=%d\n", dato_leido);
  return 0;
}
```

Tipos básicos XDR

- Entero (`int`, `unsigned int`, `hyper`)
- Real (`float`, `double`)
- Carácter (`char`)
- Booleano (`bool`)

> Todos se codifican con 4 bytes o múltiplo de 4, y *Big Endian*

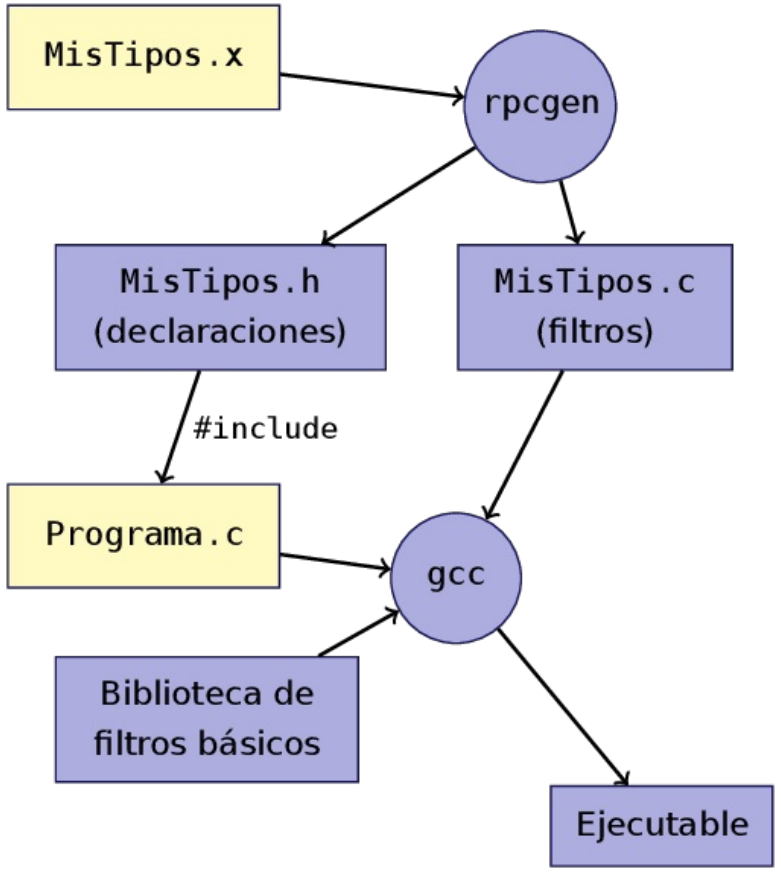
Restantes tipos

- Arrays (longitud fija o variable, *strings*)
- Estructuras
- Uniones (discriminadas)
- Punteros

El programador debe declarar estos tipos en un fichero `.x` con una sintaxis específica

Las funciones "filtro" para estos tipos son generadas a partir del `.x` por la herramienta `rpcgen`

Infraestructura de desarrollo usando rpcgen



Arrays de longitud fija

- Se declaran como en C

```
/* Fichero ejemplo.x */
typedef int TresEnteros[3];
```

- XDR codificará *TODOS* sus elementos en secuencia

Arrays de longitud variable

- Se declaran con `< >` en vez de `[ ]`
- Dentro de los `< >` va su tamaño máximo
- Su tamaño real se decide en tiempo de ejecución
- XDR sólo codificará el número *real* de elementos

```
/* ejemplo.x */
typedef int VariosEnteros<100>;
```

Arrays de longitud variable (cont)

Son convertidos por `rpcgen` a un tipo `struct` para C:

```
// Parte de ejemplo.h
typedef struct {
  u_int VariosEnteros_len;
  int *VariosEnteros_val;
} VariosEnteros;
typedef struct VariosEnteros VariosEnteros;
```

El programador podrá declarar variables de tipo `VariosEnteros`, pero debe saber que son `struct` y cómo se llaman sus campos.

Arrays de longitud variable (cont)

```c
#include "ejemplo.h"
// ...
  VariosEnteros v;
  int n;

  // Leemos n de alguna forma
  v.VariosEnteros_len = n;
  // Reservamos memoria para ese numero de enteros
  v.VariosEnteros_val = malloc(n * sizeof(int));
  if (v.VariosEnteros_val==NULL) {
     fprintf(stderr, "Error, no hay memoria suficiente\n");
     exit(1);
  }
  // Podemos usar el array con la sintaxis:
  // v.VariosEnteros_val[indice]
```

Arrays de longitud variable (decodificación)

XDR puede decodificar (recibir) estos arrays, pero:

> Antes de llamar al filtro para decodificar hay que poner `NULL` en el puntero interno de la unión

El propio filtro XDR reservará memoria para el array cuando lo reciba.

> Después de terminar de usar el array hay que llamar a `xdr_free()` para liberarlo.

Puesto que XDR reservó esa memoria, XDR debe liberarla

Arrays de longitud variable (decodificación)

```c
#include "ejemplo.h"
// ...
  VariosEnteros v;
  v.VariosEnteros_val = NULL;
  // ...
  xdr_VariosEnteros(&operacion, &v);
  // Podemos acceder ya a sus elementos
  // v.VariosEnteros_len      <-- Cuántos son
  // v.VariosEnteros_val[i]   <-- Cada uno de ellos

  // Liberar
  xdr_free((xdrproc_t) xdr_VariosEnteros, &v);
```

Cadenas (string)

- El C no tiene este tipo, pues lo implementa como `char[]` o `char*`
- Eso sería ineficiente en XDR (pues 1 char = 4 bytes)
- XDR tiene su propia implementación

```
/* ejemplo-string.x */
typedef string Texto<500>;  /* 500 es el maximo */
```

Codificación

. Un entero (4 bytes) que indica la longitud de la cadena . 1 byte por cada carácter
. Relleno de ceros para alcanzar múltiplo de 4 (si es necesario)

Cadenas (cont.)

- `rpcgen` convertirá el tipo `string` a `char *`
- El C lo usará como una cadena "normal" (terminada en nulo)

```
#include "ejemplo-string.h"
// ...
  Texto t;
  t = "Cadena de prueba";
```

¿Cómo queda codificada la cadena anterior?

Estructuras

Su sintaxis y semántica es como en C:

```
/* ejemplo-struct.x */
struct Persona {
  int edad;
  string nombre<>;
  string apellidos<>;
};
```

`rpcgen` lo convierte en otra estructura C:

```
// Parte de ejemplo-struct.h
struct Persona {
    int edad;
    char *nombre;
    char *apellidos;
};
typedef struct Persona Persona;
```

Se codifican los campos en el orden en que están declarados

Uniones

Preliminar: ¿Qué es una unión C?

```
union Datos {
  int    n;
  double x;
  char   a;
}  variable;
```

Se parece formalmente a una estructura, pero:

- Sus campos se almacenan *en la misma* dirección.
- No puede usarse más de uno.
- Es como una variable con varios tipos.

```
variable.n = 15;
variable.a = 'A';
printf("%d\n", variable.n)
```

Unión discriminada XDR

- XDR implementa la idea de una *unión con un discriminante*.
- La sintaxis es específica de XDR y no se parece a la del C.

```
/* ejemplo-union.x */
union Resultado switch(int tipo) {
  case 1: int entero;
  case 2: double real;
  default: string error<>; /* Puede omitirse */
                           /* el tamaño maximo */
};
```

odificación: . El valor del discriminante (4 bytes) . El valor del campo elegido (según su tipo)

Unión discriminada, implementación en C

- Ese tipo no existe en C (existe la `union` "sin más")
- `rpcgen` lo convierte en un `struct` que contiene el discriminante, más una `union` C

```c
/* Parte de ejemplo-union.h */
struct Resultado {
  int tipo;
  union {
    int entero;
    double real;
    char *error;
  } Resultado_u;
};
typedef struct Resultado Resultado;
```

- El programador debe conocer los nombres de los campos de este `struct` para poder usarlo.

Unión discriminada, ejemplo de uso en C

```c
#include "ejemplo-union.h"
Resultado dividir(int a, int b)
{
  Resultado r;  // En r guardaremos el resultado a retorna

  // El resultado tendrá diferente tipo, según:
  // 1) a/b sea una división exacta (será entonces un entero)
  // 2) a/b no sea una división exacta (será entonces un real)
  // 3) no se pueda dividir por ser b==0 (será entonces un string)

  if (b==0) { // Tercer caso, no se puede dividir
     r.tipo = 3;   // Cualquier numero distinto de 1 y 2
     r.Resultado_u.error = "Error, division por cero";
  } else if (a % b == 0) {  // Primer caso, sale exacto
     r.tipo = 1;
     r.Resultado_u.entero = a/b;
  } else {  // Segundo caso, sale real
     r.tipo = 2;
     r.Resultado_u.real = (double) a / (double) b;
  }
  // Una vez obtenido el resultado, lo retornamos
  return r;
}
```

Unión discriminada, decodificación

Si la `union` contiene algún puntero, hay que ponerlo a `NULL` antes de llamar al filtro que recibe.

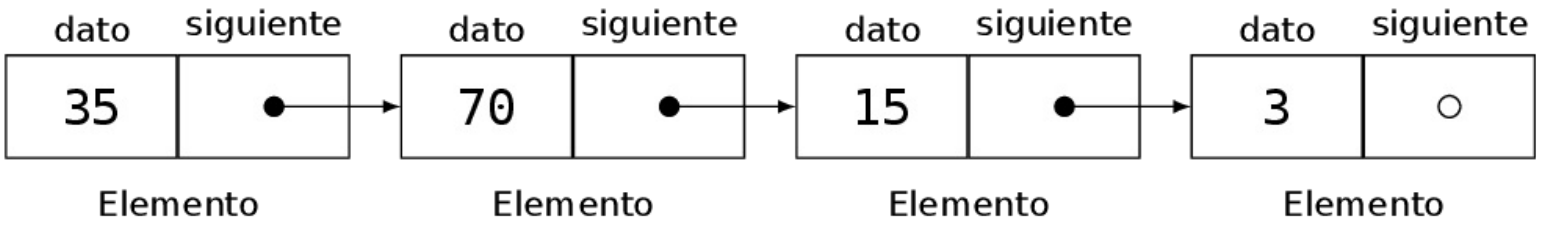
```
// ...
Resultado x;
// Antes de llamar al filtro, inicializar punteros
// (no sabemos si en el fichero habrá el caso default o no)
x.Resultado_u.error = NULL;
// operacion se inicializaría en modo XDR_DECODE
// Decodificar y recibir en x
xdr_Resultado(&operacion, &x);
switch(x.tipo) {
  case 1: // Ha salido exacto
    printf("Resultado = %d\n", x.Resultado_u.entero);
    break;
  case 2: // Ha salido real
    printf("Resultado = %f\n", x.Resultado_u.real);
    break;
  default: // Ha habido error
    printf("Resultado = %s\n", x.Resultado_u.error);
}
// Liberar memoria
xdr_free((xdrproc_t) xdr_Resultado, &x);
```

Punteros

- Los punteros no se pueden transmitir
- Ya que pierden su significado
- Sin embargo se usan mucho en C, por ejemplo para estructuras de datos

Lista enlazada:

```
struct Elemento {
  int dato;
  struct Elemento *siguiente;
};
```



Transmisión de estructuras de datos acíclicas

XDR no puede transmitir los punteros, pero puede transmitir la lista

- Basta transmitir cada dato, y un booleano indicando si hay más
- En destino la lista se reconstruirá a base de `malloc()`
- Es el concepto de **serialización**

Para ello XDR define un nuevo tipo: **Datos opcionales**

- Un dato opcional puede tener valor asignado o no
- Si no lo tiene, se codifica como `FALSE`
- Si lo tiene, se codifica como `TRUE` seguido del valor

> Los punteros C se convierten en datos opcionales XDR

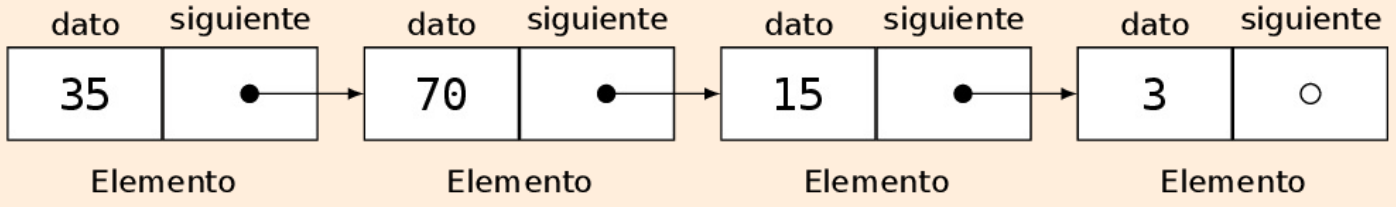
Datos opcionales

La sintaxis es igual a la de un puntero C, pero su significado es otro.

```
/* ejemplo-lista-enlazada.x */
struct ListaEnlazada {
  int           dato;
  ListaEnlazada *siguiente;
};
```

Ver ejemplos de uso en los apuntes y en prácticas

¿Cuál sería el resultado de codificar con XDR esta lista?



XDR y sockets

- XDR fue creado para usarse con ONC RPC (próximas lecciones)
- Pero puede usarse "por separado" para:
  - Convertir datos de formato nativo a un formato independiente
  - Volcar esos datos a disco
  - O enviarlos por la red
  - Y viceversa

La estructura `XDR operacion` se inicializa con un `FILE*` ¿cómo se inicializaría con un socket?

> Si un socket se puede leer y escribir como un fichero ¿no es al fin y al cabo un caso de `FILE*`

Manejadores: ¿tipo int o tipo FILE*?

Existen ambos:

- `int` es el usado por la API del operativo.
  - `open()`, `read()`, `write()`, `close()`, `dup()`
  - Bajo nivel
  - Usada también por los sockets
- `FILE*` está implementada en la biblioteca estándar C
  - `fopen()`, `fread()`, `fwrite()`, `fclose()`, `fprintf()`, `fscanf()`, etc.
  - Alto nivel (formateo de la entrada/salida)
  - Usada por XDR
  - No usada por los sockets
- ¡Pero se puede convertir uno en otro!
  - `FILE* fdopen(int, "mode")`

Uso de un socket con XDR (enviar)

En los apuntes hay ejemplos completos. Aqui se muestra solo la inicialización de la variable `XDR operacion`.

```
// ... includes (el ultimo ha de ser el generado por rpcgen)

  int sock_datos, duplicado_socket;
  FILE *f_enviar;
  XDR operacion_enviar;

  // El socket se inicializaría como siempre y después:

  // --- Enviar datos por el socket
  duplicado_socket = dup(sock_datos);
   f_enviar = fdopen(duplicado_socket, "w");
  xdrstdio_create(&operacion_enviar, f_enviar, XDR_ENCODE);
  // Usar filtros XDR necesarios para enviar datos
  // ... no se muestra ...

  // Liberar recursos
  fclose(f_enviar);
  xdr_destroy(&operacion_enviar);
  // Liberar memoria si habiamos hecho mallocs ...
```

Uso de un socket con XDR (recibir)

```
// ... includes (el ultimo ha de ser el generado por rpcgen)

  int sock_datos, duplicado_socket;
  FILE  *f_recibir;
  XDR operacion_recibir;

  // El socket se inicializaría como siempre y después:

  // ---  Recibir datos por el socket
  duplicado_socket = dup(sock_datos);
   f_recibir = fdopen(duplicado_socket, "r");
  xdrstdio_create(&operacion_recibir, f_recibir, XDR_DECODE);
  // Usar filtros XDR necesarios para recibir datos
  // (recordar poner a NULL los punteros en los datos a recibir)
  // ... no se muestra ...

  // Liberar recursos
  fclose(f_recibir);
  xdr_destroy(&operacion_recibir);
  // Usar xdr_free() si los datos recibidos contenian punteros
```