ATC (Arquitecturas Distribuídas)
Preguntas Seleccionadas

Curso 2005-2006

Estos son ejercicios de ejemplo suministrados por los profesores de la asignatura. Las preguntas propuestas por los alumnos deben tener un enfoque similar, en particular en lo que respecta al tipo de respuesta, que puede ser breve (para preguntas con una respuesta muy concreta) o de longitud media (para las "de desarrollo"),

Problema 1. General-Sockets ej-distribuidas-1.txt

¿Cuál o cuáles de las siguientes afirmaciones son ciertas?

A) La función bind debe usarse en el cliente para asignar la IP y puerto del servidor antes de conectar.
B) La función htons convierte un dato de tipo short int del orden de máquina al orden de red.
C) En algunas máquinas la función htonl no tiene efecto alguno (devuelve el mismo número que recibe)
D) Si un proceso crea un socket y seguidamente varios hijos que ejecutan un accept() sobre ese socket, cuando llegue un cliente todos los hijos le atenderán.

Solución:

B y C

Explicación:

La afirmación A es falsa, puesto que bind() es para asignar una dirección al socket local. La dirección del otro extremo se especifica en el connect(). La afirmación D también es falsa, puesto que sólo uno de los hijos es desbloqueado. B y C son ciertas.

Problema 2. General-Sockets ej-distribuidas-1.txt

Explica brevemente de qué dos formas se puede usar fork() en la programación de un servidor que admita varios clientes

Solución:

Un servidor puede llamar a fork() cada vez que acepta un nuevo cliente. El proceso padre, que nunca termina, retornará al accept() a esperar por nuevos clientes. El hijo dará servicio al cliente y después terminará con exit().

Otra posibilidad es que un servidor llame a fork() varias veces antes de entrar en el bucle de espera por clientes. De este modo todos los hijos ejecutarán el mismo código que el padre, entrando todos a la vez en el accept(). Cuando llegue un cliente, solo uno de ellos saldrá del accept() y continuará para darle servicio. Cuando haya terminado el servicio, cerrará la conexión con ese cliente y retornará al accept() a esperar por otro.

Problema 3. XDR-SunRPC ej-distribuidas-1.txt

A continuación se muestra una parte del fichero def.x que define el interface de una aplicación basada en el ONC RPC de Sun.

```
program OPERADOR {
        version OPVERS {
                void  ACABA (void) =1;
                tipo3 OP1 (int)=2;
                int   OP2 (tipo2)=3;
                tipo2 OP3 (tipo3)=4;
        }=5;
}=0x2095F999;
```

❑ **3.1** Si en la máquina del cliente tenemos declaradas la siguientes variables

```
CLIENT *clnt;
int    *dat1, i;
tipo2  *dat2;
tipo3  *dat3;
```

Escribe el código necesario para hacer una invocación remota al servicio OP1 en la máquina serv.com.

Solución:

```
clnt=clnt_create("serv.com",OPERADOR, OPVERS, "tcp");
dat3=OP1_5(dat1,clnt);
```

Explicación:

Se debe inicializar la estructura clnt llamando a clnt_create(). En esta llamada se le pasa el nombre de la máquina ("serv.com"), la constante que identifica al servidor (OPERADOR), la constante que especifica el número de versión del servidor (OPVERS), y el protocolo de transporte, en el que, ya que no se especifica lo contrario, podemos poner "tcp".

Una vez inicializado clnt se puede llamar al servicio. Hay que recordar que la función a invocar se llama como el servicio declarado en el interfaz, pero con un sufijo indicando el número de versión, que en este caso será _5. El parámetro que recibe el servicio es un int, según el interfaz, pero debe recordarse que en la implementación en C lo que hay que pasarle es un puntero a ese tipo, por tanto un int*. Ya que la variable dat1 es precisamente de ese tipo, basta con pasarle esa variable (en este caso no hay que poner el & delante, ya que dat1 ya es la dirección de un int). El servicio devuelve un tipo3, de acuerdo con el interfaz, pero debemos recordar que en la implementación C se recibe un puntero a este tipo. La variable dat3 es por tanto la indicada para recoger este resultado.

Con las consideraciones anteriores, es inmediato completar el código como muestra la solución.

❑ **3.2** Si tipo3 es un array variable de int, escribe el código necesario tras la invocación anterior para mostrar el resultado recibido. Por ejemplo, la salida podría ser:

Recibidos 3 elementos:
4 5 9

Solución:

```
printf("Recibidos %d elementos:\n",
       dat3->tipo3_len);
for (i=0; i<dat3->tipo3_len; i++)
  printf("%d ", dat3->tipo3_val[i]);
printf("\n");
```

Explicación:

Recordemos que un array variable se convierte en C en una estructura, cuyos campos se llamarían en este caso tipo3_len y tipo3_val. La variable dat3 es por tanto un puntero a una estructura con estos campos.

Para imprimir el número de elementos recibidos debemos acceder al campo tipo3_len de la estructura apuntada por dat3. Esto se logra con la sintaxis dat3->tipo3_len o alternativamente (*dat3).tipo3_len.

Para imprimir cada elemento debemos realizar un bucle que se repita tantas veces como indica dat3->tipo3_len, y en cada iteración mostrar uno de los elementos apuntados por dat3->tipo_val. Podemos usar sintaxis de array con este campo y acceder a dat3->tipo_val[i], o alternativamente (*dat3).tipo_val[i].

Problema 4. DCE-RPC ej-distribuidas-1.txt

Define brevemente que es un uuid y cómo se obtiene

Solución:

Es un identificador único universal. Se trata de una constante numérica que identifica un interfaz. Debe garantizarse que no hay dos interfaces diferentes con el mismo uuid. Se obtiene algorítmicamente mediante el programa uuidgen, el cual garantiza que nunca generará dos uuid iguales. El algoritmo toma la MAC de la tarjeta de red del ordenador en que se ejecuta, y el instante temporal, para garantizar que dos ordenadores diferentes no puedan producir el mismo uuid, ni el mismo ordenador pueda producirlo en dos instantes diferentes.

Problema 5. General-Sockets ej-distribuidas-2.txt

Un programa ha creado 5 sockets y los ha almacenado en un array. Cada socket ha sido asignado a un puerto diferente, y se le ha puesto en modo escucha. Se desea que seguidamente el servidor quede a la espera de clientes en cualquiera de los sockets, y según el socket que reciba

la conexión se ejecute una función u otra. Para ello, la función EjecutarServicio() recibe un entero, entre 0 y 4, que indica qué servicio se debe ejecutar.

Este es el código del programa principal. Los interrogantes son instrucciones que faltan, por las que se preguntará más adelante.

```
/* ------------------------------------------------- */
/* Variable global */
int sockets[5];
...
main() {
 /* Variables locales de main */
 fd_set cjto;
 int max;
/* No se muestra la inicialización de los sockets */
while (1) {
 FD_ZERO(&cjto);
 max = 0;
 for (i=0; i<5; i++) {
      ??????????????????????????
      ??????????????????????????
 }
 /* Esperar conexiones */
 select(max+1, &cjto, NULL, NULL, NULL);
 /* Elegir función, según el socket que
   haya recibido la conexión */
 for (i=0; i<5; i++) {
   if (????????? ( ?????????? , ????????? ))
      EjecutarServicio(i);
  } // for
 } // while
} // main
/* ------------------------------------------------- */
```

Y seguidamente se muestra también (completo) el código de la función EjecutarServicio()

```
/* ------------------------------------------------- */
// Funcion EjecutarServicio()
void EjecutarServicio(int n) {
  int sDat;
  sDat=accept(sockets[n], NULL, 0);
  switch(n) {
    case 1: Servicio1(sDat); break;
    case 2: Servicio2(sDat); break;
    ...
    default: fprintf(stderr, "Error\n");
  }
  close(sDat);
}
/* ------------------------------------------------- */
```

En base a esta información, responder las dos preguntas siguientes:

❑ **5.1** Completar las instrucciones que faltan dentro del primer bucle for

Solución:

```
FD_SET(sockets[i], &cjto);
if (max<sockets[i]) max=sockets[i];
```

Explicación:

Antes de invocar select(), es necesario inicializar una variable que indique el valor numérico del socket con el descriptor más alto (más uno), y un mapa de bits indicando qué sockets queremos "observar". El bucle for inicial, por tanto, tiene como cometido la inicialización de ambas variables. El mapa de bits debe ser una variable de tipo fd_set, y por tanto será la variable cjto. Esta variable se inicializa mediante las macros FD_ZERO (para ponerla completamente a cero) y FD_SET para poner a 1 los bits correspondientes a los sockets que se quieren observar, que son justamente los almacenados en el array sockets[]. Por otro lado, si observamos la llamada a select() en la línea 18, vemos que el primer parámetro que se le pasa es max+1, por lo que la variable max debe ser inicializada simplemente con el mayor de los elementos de sockets[]. A partir de estas ideas no es difícil completar el código pedido.

❑ **5.2** Completar la condición del if

Solución:

```
if (FD_ISSET(sockets[i], &cjto))
```

Explicación:

Una vez se ha salido del select(), la variable cjto ha cambiado de valor, y ahora contiene un 1 sólo en el socket en el que se ha detectado actividad (esto es, un intento de conexión). La macro FD_ISSET permite averiguar si un socket dado está a 1 o no en el mapa de bits. Comprobando con esta macro todos los sockets almacenados en el array sockets[] determinaremos cuál ha tenido la actividad, y llamaremos a la rutina de servicio pasándole el índice del array de ese socket. A partir de estas consideraciones, es directo determinar la línea que falta.

Problema 6. XDR-SunRPC — ej-distribuidas-2.txt

Considera la siguiente declaración XDR:

```
struct Grupo {
   int numeros<10>;
   string *texto<13>;
};
```

¿Cuál sería el máximo tamaño de un dato de este tipo, una vez codificado en XDR? ¿Y su tamaño mínimo?

Solución:

MAXIMO = 64, MINIMO = 8

Explicación:

El tamaño máximo ocurre cuando ambos datos, que son de longitud variable, alcanzan su longitud máxima. En este caso el campo numeros contendrá 10 enteros, y se codificará con 44 bytes (4 para el contador y 40 para los datos). El campo texto, que es opcional (obsérvese el *) debe estar presente, por lo que ocupará 4 bytes para el indicador de que sí hay dato, más lo que ocupe el dato en sí, que en este caso, al ser una cadena de 13 letras serán 20 bytes (4 para el contador, 13 para las letras, y 3 de relleno para alcanzar el siguiente múltiplo de 4). De modo que la estructura completa ocupará 44 + 24 bytes.

El tamaño mínimo ocurre cuando el array variable numeros tiene cero elementos y el campo opcional texto no está presente. En este caso se requieren 4 bytes para el pri- mer campo (para almacenar el contador indicando que hay 0 elementos) y 4 bytes para el segundo (para almacenar el indicador 00 00 00 00 de que no hay dato opcional).

Problema 7. XDR-SunRPC — ej-distribuidas-2.txt

¿Qué información debe enviar el stub del cliente a la función repartidora para identificar el servicio que desea ejecutar? ¿Dónde se define esta información?

Solución:

Debe enviar el número de programa, que es una constante de 8 cifras (en hexadecimal) que identifica al proceso servidor, el número de versión y el número del servicio deseado. Estas tres constantes se definene en el fichero de interfaz (extensión .x). La herramienta rpcgen genera un .h que #define estas constantes con un nombre (también declarado en el interfaz)

Problema 8. DCE-RPC — ej-distribuidas-2.txt

Se ha programado un servicio que será invocado mediante el mecanismo rpc de dce. Este servicio recibe una serie de números como parámetro (en un array adaptable), y devuelve el máximo de ellos, o un error si el array está vacío. El interfaz es el siguiente:

```
[
  uuid(d2298b7d-cf8c-463e-95be-7c0cdc153fdf),
  version(1.0)
]
interface MAX_LISTA
{
  typedef union Return_t switch (short int error) resultado {
   case -1: [string, ptr] char *msg;
   case 0: int maximo;
  } Return_t;
  Return_t max([in] unsigned long int n, [max_is(n), in] int dat[*]);
}
```

Escribe la declaración IDL de un tipo de datos para contener el array adaptable, que permita hacer una función max que reciba un solo parámetro en lugar de dos.

Solución:

```
typedef struct Array_t {
    unsigned long int len;
    [max_is(len)] float datos[*];
} Array_t;
```

Explicación:

El array adaptable necesita una variable que indique cuántos elementos contiene y otra que apunte a los datos en sí. En el interfaz propuesto cada una de estas variables es un parámetro diferente para la función max, y se nos pide combinarlas en un único parámetro.

La solución ha de ser una estructura, uno de cuyos campos será el entero que indica el número de elementos del array y el otro el puntero a los datos, o bien un array adaptable. Este segundo campo necesita que, mediante un atributo, se especifique el nombre del campo que contiene la longitud del array, y que se especifique si esta longitud representa el número de elementos total del array, o bien el índice del último elemento. Este segundo caso será el que nos ocupa, a la vista de la declaración suministrada para el servicio max().

Problema 9. General-Sockets **ej-distribuidas-3.txt**

¿Cuáles son las funciones de la librería de sockets necesarias para construir un SERVIDOR que use el protocolo TCP? Poner únicamente el nombre. No hacen falta los parámetros.

Solución:

socket(), bind(), listen(), accept(), read(), write(), close()

Problema 10. General-Sockets **ej-distribuidas-3.txt**

Usando la librería de sockets para una comunicación con el protocolo UDP, ¿Cuáles son las funciones que nos permiten enviar y recibir datos? Poner únicamente el nombre. No hacen falta los parámetros.

Solución:

sendto() recvfrom()

Problema 11. SunRPC **ej-distribuidas-3.txt**

El fichero de descripción del interface de una aplicación es el que se muestra a continuación:

```
struct tipo1 {
        int dat1;
        tipo1 *otro;
};
struct tipo2 {
        float dat2<120>;
        string dat3<82>;
}
union tipo3 switch (int que) {
        case 1:
                float val1;
        case 10:
                tipo1 val2;
        case 20:
                int val3<10>;
        default:
                string val4<85>;
};
program OPERADOR {
        version OPVERS {
                void  ACABA (void) =1;
                tipo3 OP1 (int)=2;
                int   OP2 (tipo2)=3;
                tipo1 OP3 (tipo3)=4;
                float DIVID (int)=5;
        }=1;
}=0x2095F999;
```

Teniendo esto en cuenta constestar a las siguientes preguntas:

❑ **11.1** ¿Cuál es el equivalente en C del tipo de dato XDR tipo2?

Solución:

```
struct tipo2 {
        struct {
                u_int dat2_len;
                float *dat2_val;
        } dat2;
        char *dat3;
};
typedef struct tipo2 tipo2;
```

Explicación:

El tipo de datos tipo2 es una estructura, por lo que su equivalente en C será también una estructura. El equivalente de una estructura XDR en C se obtiene haciendo la conversión de cada uno de sus elementos por separados. En nuestro caso, el array de longitud variable de números reales se convierte en una estructura del mismo nombre que el array y con dos campos:

- un entero sin signo, que contendrá el número de elementos que hay en el array y cuyo nombre será el del array con el sufijo _len (de "length", longitud);

- y un puntero al tipo base del array, en este caso float, y cuyo nombre será el del array con el sufijo _val (de "value", valor).

El equivalente en C del tipo XDR string es simplemente un puntero a char. Aunque la declaración XDR tiene el mismo aspecto que una array de longitud variable, el tipo equivalente en C no es el mismo.

❑ **11.2** Después de realizar una llamada al procedimiento OP1, el cliente ha recibido una LISTA ENLAZADA de números enteros con los valores 17 y 32 en ella. ¿Cuáles han sido todos los bytes de respuesta que ha enviado el servidor a través de la red?. Contestar con los valores en hexadecimal y procurando separar adecuadamente los distintos tipos de información.

Solución:

```
00 00 00 0A  00 00 00 11  00 00 00 01  00 00 00 20  00 00 00 00
```

Explicación:

El procedimiento OP1 devuelve como resultado una variable del tipo "tipo3", la cual es una unión discriminada, por lo tanto, lo que recibe el cliente son los bytes con los que se codifica una unión discriminada de tipo "tipo3".

Lo primero que recibe el cliente es el discriminate. Como en el enunciado se no indica que se ha recibido una lista enlazada, el tipo que está en uso en la unión es el que corresponde al tipo "tipo1" que es el único con el que se puede construir una lista enlazada de números enteros. Por lo tanto, los primeros bytes corresponden al discriminante y, en concreto, a la codificación del caso de valor decimal 10 (0A en hexadecimal).

A continuación del deiscriminante, viene la codificación del valor de la unión. Es una lista enlazada basada en la estructura tipo1, por lo que se van codificando los valores de la estructura en orden de declaración: primero el entero y luego el puntero. En nuestro caso el entero es el valor 17 (11h) y el puntero, como tenemos otro elemento en la lista, tiene que valer 1. Ese otro elemento de la lista vuelve a ser otra estructura de tipo1 por lo que es necesario codificar el valor del entero 32 (20h) y a continuación el valor del puntero, que en este caso, al no existir más elementos en la lista, vale cero.

Lógicamente, todos estos valores mencionados se han codificado en la respuesta con todos sus bytes (cada entero ocupa 32 bits).

Problema 12. DCE-RPC _____ ej-distribuidas-3.txt

A continuación se muestra el fichero de definición del interface (.idl) de una aplicación escrita para el DCE RPC. También se muestra el fichero de atributos de configuración (.acf) usado con la parte cliente.

Fichero .IDL

```
[uuid (df5e1744-f927-11d0-98eb-9e620b23aa77),  version (1.0)]  interface SERV
{
 typedef struct complej {
       small x,y;
       }complej;

  void op1(
[in] small dat1,
[in] complej dat2,
[out] long *resul);
}
```

Fichero .ACF del cliente

```
[explicit_handle] interface SERV
{
  op1 ( );
}
```

¿Cómo sería la llamada al servicio op1 desde el programa del cliente? Declarar las variables necesarias. No hace falta inicializarlas.

Solución:

```
handle_t manejador;
idl_small d1;
complej d2;
idl_long res;
op1(manejador, d1, d2, &res);
```

Explicación:

Al tener en el fichero acf del cliente el atributo explicit_handle para el interface, será necesario incluir un manejador de enlazado (correctamente inicializado) en cada una de las llamadas a los procedimientos remotos de ese interface que se hagan desde el cliente. En este caso, como nos solicitan la declaración pero no la inicialización de las variables, tendremos que declarar un manejador de enlazado que en lenguaje C es el tipo handle_t e incluirlo en la llamada a op1. El resto de parámetros del procemiento son los que aparecen en el idl y se declaran variables de los tipos predefinidos equivalentes de DCE.

Índice

Problema 1: General-Sockets 1
Problema 2: General-Sockets 1
Problema 3: XDR-SunRPC 1
Problema 4: DCE-RPC 2
Problema 5: General-Sockets 2
Problema 6: XDR-SunRPC 4
Problema 7: XDR-SunRPC 4
Problema 8: DCE-RPC 4
Problema 9: General-Sockets 5
Problema 10: General-Sockets 5
Problema 11: SunRPC 5
Problema 12: DCE-RPC 6