ATC (Arquitecturas Distribuídas)
Preguntas Seleccionadas

Curso 2005-2006

Esta es una colección de 60 preguntas ideadas y redactadas por los alumnos de la asignatura. Seis de ellas se preguntarán en el examen parcial de la asignatura (si se trata de un problema que requiere operación, los datos podrían ser diferentes en el examen).

Las respuestas se han dejado tal como las envió cada alumno, aunque en algunos casos son incorrectas o incompletas. En estos casos, el profesor ha añadido una breve nota entre corchetes al final de la respuesta.

Problema 1. General-Sockets

Tenemos un cliente que lee la hora enviada por un servidor que sigue el protocolo time. El código presenta el siguiente aspecto:

```
//printf("Hacemos la conexion\n");
if (connect(s, (struct sockaddr *)&servidor,sizeof(servidor))<0){
   printf("Error al hacer el connect");
   return(-1);
}

//printf("Leemos el dato\n");
?????????
if (num_bytes<0){
   printf("El servidor no responde\n");
   close(s);
   return(-1);
}
//convertimos la hora al formato adecuado
//restamos el # de segs que hay entre 1900 y 1970
????????????
hora = hora - 2208988800;
if (close(s)){
   printf("Error: problemas al cerrar\n");
   return (-1);
}
```

¿Sabiendo que se esperan cuatro bytes de datos, cómo leemos el dato?

¿Como convertimos la hora en un formato adecuado?

Solución:

```
num_bytes = read(s, &hora, 4);
hora = ntohl(hora);
```

Problema 2. General-Sockets

¿Qué funciones irían en los números del siguiente esquema de un protoclo UDP?
SERVIDOR socket()->bind()->(1)->(2)->close()
CLIENTE socket()->bind()->(3)->(4)->close()

Solución:

```
(1)(4)recvfrom()
(2)(3)sendto()
```

Explicación:

Esquema sacado de las transparencias del tema "Programación con socktes"

Problema 3. General-Sockets

a) retcod = select(maxfd, &selector, null, null, null);
¿Que consecuencias tiene el último parametro de esta llamada?

b) timeval carga;
   carga.tv_sec = 0;
   carga.tv_usec = 0;
   retcod = select(maxfd, &selector, null, null, &carga);

¿Y en esta nueva llamada?

Solución:

a) Se bloqueará indefinidamente, hasta que se reciba algo por alguno de los sockets vigilados.

b) No se bloqueará, sino que retornará inmediatamente indicando cuántos sockets están listos, o cero si no hay. Permite implementar un polling sobre los sockets vigilados, sin bloquearse.

Problema 4. General-Sockets

Explica brevemente la principal diferencia entre concurrencia aparente y concurrencia real, y que función se utiliza para su implementación habitual en cada caso.

Solución:

La concurrencia real se basa en un proceso independiente por petición recibida, habitualmente implementado mediante el uso de fork.
La concurrencia aparente se basa en un único proceso que atiende todas las peticiones, habitualmente implementado con el uso de select.

Problema 5. General-Sockets

Indicar cuáles de las siguientes funciones pueden bloquear al programa que las invoca durante un tiempo indefinido y si es invocada por un cliente o por un servidor:

A) accept()
B) bind()
C) connect()
D) listen()
E) sendto()
F) socket()
G) recvfrom()

Solución:

A) y G)

Explicación:

A) El servidor se bloquea en la llamada accept hasta que reciba una nueva conexión, por lo que el tiempo de bloqueo es indefinido.
G) Un cliente o un servidor se pueden bloquear un tiempo indefinido en una llamada a recvfrom hasta que se reciba un paquete de datos desde el otro extremo de la conexión.

Problema 6. General-Sockets

El código que se muestra a continuación es el cuerpo principal despues de una llamada a select():

```
...
while (1) {

   ...

   select(maxFd, &selector, (fd_set *) 0, (fd_set *) 0, &tiempo);

   if (FD_ISSET(s1, &selector)) {
      sendto(s1, buffer, numBy, 0, (struct sockaddr *)&cli2, sizeof(cli2));
   }

   if (FD_ISSET(s2, &selector)) {
      s3 = accept(s2, (struct sockaddr *)&cli1, (socklen_t *)&tamCli);
   }

   if (FD_ISSET(s3, &selector)) {
     nBytesR = read(s3, buffer, sizeof(buffer));
   }
}
...
```

Teniendo en cuenta el anterior código, responde a las siguientes preguntas:

❑ **6.1** ¿Cual o cuales de las siguientes afirmaciones son ciertas?

A) Hay más de un socket de conexión TCP asociado al select().
B) El socket s2 es un socket UDP.
C) El socket s1 es un socket UDP.
D) El socket s3 es un socket de escucha (listener).

Solución:

C

Explicación:

La afirmación A es falsa, puesto que de todos los sockets con los que trabaja el select, solo acepta conexiones TCP el socket "s3".
La afirmación B también es falsa, ya que no se puede realizar una operación "accept()" sobre un socket UDP.
La afirmación C es cierta, ya que "sendto()" es una operación que se realiza sobre sockets UDP.
Y por último, la afirmación D es falsa debido a que el socket sobre el que se el accept es "s2" y se retorna en "s3", por tanto, el socket que está a la escucha es "s3", y quien soporta la conexión para envío/recepción de datos TCP es "s2".

Problema 7. General-Sockets

Explica brevemente los distintos tipos de valores de retorno de la función Select.

Solución:

Devuelve el número total de descriptores que están listos. En caso de que haya expirado el timeout devolverá el valor cero. Por último retornará el valor -1 en el caso de que haya ocurrido algun error.

Problema 8. General-Sockets

¿Qué funciones es necesario utilizar para, sin usar XDR, evitar problemas de comunicación entre máquinas Big-Endian y Little-Endian?

Solución:

htons,htonl,ntohs,ntohl

Explicación:

htons() host to network short -> convierte un valor corto (2 bytes) formato usado por el host (nuestra máquina) al usado por la red.

htonl() host to network long -> convierte un valor largo (4 bytes) formato usado por el host (nuestra máquina) al usado por la red.

ntohs() network to host short -> convierte un valor corto (2 bytes) formato usado por la red al usado por el host.

ntohl() network to host long -> convierte un valor largo (4 bytes) formato usado por la red al usado por el host.

Problema 9. General-Sockets

¿ Cúal es la funcionalidad de la funcion bind en la programación con sockets?

A) Asignar la IP y puerto del servidor antes de conectar.
B) Esperar una conexión entrante.
C) Conectarse a un puerto definido en una dirección IP
D) Asignar dirección local al socket

Solución:

D

Problema 10. General-Sockets

¿En qué se diferencian la concurrencia real y aparente?

Solución:

En la concurrencia real cada vez que el servidor recibe una petición de un cliente, un proceso independiente se encarga de atenderla, mientras que en la concurrencia aparente, el servidor atiende las peticiones de múltiples clientes con únicamente un proceso activo.

Problema 11. General-Sockets

En un socket TCP, ¿cuándo sabe el servidor que el cliente ha cerrado la conexión?

a) Al llamar a la función read, si devuelve el ASCII del carácter fin de línea seguido de un retorno de carro.
b) Al llamar a la función read, si devuelve cero
c) Al llamar a la función read, si devuelve un valor menor que cero
d) No es posible conocer esta información

Solución:

La respuesta correcta es la b)

Problema 12. General-Sockets

¿Qué devuelve la llamada a la función fork() y qué significa?

Solución:

Devuelve un 0 al proceso hijo y un entero que es el PID del nuevo proceso creado al padre.

Problema 13. General-Sockets

Si queremos implementar un servidor capaz de recibir 1 un cliente por UDP y 10 por TCP mediante concurrencia aparente ¿Cuantos sockets son necesarios?

Solución:

12:
1 para UDP
1 para capturar las peticiones TCP
10 para atender las peticiones TCP

Problema 14. General-Sockets

Sea el siguiente fragmento de código de un servidor con sockets, cuya única misión es recibir una cadena de texto de un cliente y mostrarla por pantalla:

```
main() {
      int s, n_sock;
      /* Inicializaciones */
      while (1) {
            n_sock = accept( s, (struct sockaddr *)&cliente,
                  &longitud_cliente);
            /* Se omite la gestión de errores */
            memset(buffer, 0, sizeof buffer);
            leidos= read(n_sock, &buffer, sizeof buffer);
            /* Se omite la gestión de errores */
            printf("Cadena recibida: %s\n", buffer);
            close(s);
      }
}
```

❑ **14.1** Ignorando tanto la gestión de errores que debería hacer el servidor y que no se muestra como las declaraciones de estructuras y otras variables, indicar si el código es correcto y, en caso de que no lo sea, cómo corregirlo

Solución:

El código no es correcto.
Cambiar el close(s) por close(n_sock)
[N. del P: Además el código asume que el texto recibido cabe en el buffer]

Explicación:

Con close(s) no estamos cerrando el socket de la conexión actual, sino el de escucha.

Problema 15. General-Sockets

La función shutdown del interfaz de sockets permite:
a) Especificar que no se hagan más lecturas sobre un socket.
b) Especificar que no se hagan más escrituras sobre un socket.
c) Cerrar un socket.
d) Todas las anteriores son correctas.

Solución:

d)

Explicación:

El modo 0 implica la respuesta a), el 1 la b) y el 2 la c).

Problema 16. General-Sockets

¿Cual es la codificacion UTF-8 del caracter la letra japonesa Hiragana "I", si su código Unicode es el U+3044?

Solución:

E3 81 84

Explicación:

- Caso entre U+0800 y U+FFFF, requiere 16 bits:
0011 0000 0100 0100
3 0 4 4

* La codificación requiere 3 bytes.
1110xxxx, 10xxxxxx, 10xxxxxx

* Los 16 bits se han de dividir en grupos de 4, 6 y 6, y resultan:
0011 000001 000100

* Al primer grupo se le pone delante 1110, y a los restantes se les pone 10, lo que resulta en los tres bytes:
11100011, 10000001, 10000100

Problema 17. General-Sockets

Dada la estructura FD_SET fdset y el socket int sock, indique las instrucciones necesarias para resetearla, poner a 1 el bit correspondiente al socket y finalmente comprobar si el bit del socket se encuentra activo.

Solución:

```
FD_ZERO(&fdset);
FD_SET(sock,&fdset);
if(FD_ISSET(sock,&fdset))....
```

Explicación:

FD_ZERO:resetea el valor de la estructura fdset.
FD_SET: actiba el bit correspondiente a sock en la estructura fdset
FD_ISSET: comprueba si sock ha sido activado en la estructura fdset

Problema 18. General-Sockets

Un servidor tarda 1s en responder a una petición, y 100ms en crear un proceso. Si recibe 10 peticiones consecutivas ¿Cúal será la ganancia de utilizar un servidor concurrente respecto de uno iterativo?

Solución:

G=5

Explicación:

Los tiempos asociados a la ejecución en cada tipo de sevidor son los siguietes:

Tconcurrente=10*1=10s
Titerativo=10*0,1+1=2s

con lo que la ganancia resultante es:

G=Tconcurrente/Titerativo=10/2=5

Problema 19. General-Sockets

¿Cuál o cuáles de las siguientes afirmaciones son ciertas?

A) No tiene sentido usar htons() y htonl() si el dato a transmitir siempre ocupa 1 byte.
B) Pueden existir dos sockets con el mismo protocolo, dirección final local y dirección final remota.
C) El socket creado con la llamada a accept() se queda bloqueado a la espera de nuevas conexiones.
D) La interfaz sockets no aporta ningún tipo de seguridad.

Solución:

A y D.

Explicación:

La afirmación B es falsa, ya que con la terna protocolo, dirección final local y dirección final remota queda perfectamente definido un socket, por lo que no puede haber dos iguales. La afirmación C también es falsa debido a que el socket creado por la llamada a la función accept() es el socket de datos, mientras que el que se queda bloqueado a la espera de nuevas conexiones es el de escucha. Por otro lado, las afirmaciones A y D son ciertas.

Problema 20. General-Sockets

¿Cual/cuales de las siguientes afirmaciones son FALSAS?

A)El sistema operativo UNIX coloca los descriptores de sockets en una tabla de descriptores diferente a la de los ficheros.
B)Para crear un descriptor de socket se le añade al sistema operativo una nueva llamada al sistema: la función "socket".
C)Cada socket activo se identifica por un entero denominado, su descriptor de socket.
D)Una aplicación puede tener un descriptor de fichero con el mismo valor que un descriptor de socket.

Solución:

A y D

Explicación:

Tanto en A como en D, la realidad es justamente lo contrario. La afirmación A es falsa ya que se utiliza la misma tabla para ficheros y sockets. Para la afirmación D lo correcto sería que no pueden tener el mismo valor.
[N. del P: En C) conviene decir que el entero es único en el proceso]

Problema 21. XDR-SunRPC

4 - Dado el siguiente archivo de descripción del interface para una aplicación:

```
typedef int Entero;

struct enteros {
        Entero primero;
        Entero segundo;
};

program CALCULADORA {
        version CALCVERS {
                int SUMA(enteros) = 10;
        } = 1;
}=0x23456789;
```

¿Cómo sería la implementación del servicio SUMA en el servidor?, ¿Qué pasos son

necesarios para realizar la llamada a éste servicio en el cliente? (especificar los parámetros de las funciones que se utilicen).

Solución:

Servicio suma en el servidor:
int * suma_1_svc (enteros * e, struct svc_req *req)
{
  static int resultado;
  resultado = e->primero + e->segundo);
  return(&resultado);
}

Descripción de invocación en el cliente:
Simplemente habría que realizar una llamada a la función clnt_create(maquina,CALCULADORA,CALCVERS,"udp") para obtener una estructura de tipo cliente correctamente inicializada y a continuación llamaremos al servicio suma_1(&e, clnt), siendo clnt la estructura client devuelta por clnt_create() y e una variable de tipo enteros. El valor devuelto por suma_1() debemos recogerlo en una variable de tipo puntero a entero.

Problema 22. SunRPC

Se tiene la siguiente definición de variables XDR : int a; int b; struct estructura string nombre<20>; int array<10>; union mi_union switch (int que) case 1: int *x; case 2: int y[2]; default: void;

Despues de realizar la codificación a XDR de estas variables se obtiene el siguiente resultado en bytes:

00|00|00|04|4A|4F|53|45|00|00|00|05|4A|4F|53|45|53|00|00|00|00|00|00|01|00|00|00|05|00|00|00|01|00|00|00|00

Figura 2

Se pide el valor de la variable a y de la struct estructura (y todos sus campos) una vez realizada la decodificación.

Nota: 'E'=45 ; 'S' = 53 ; 'O' = 4F ; 'J'=4A ;

Solución:

El valor de las variables es el siguiente:

a = 4
estructura.nombre = JOSES
estructura.array = (5)
estructura.mi_union = NULL (Sería el entero, pero es un dato opcional y su valor es null)

Problema 23. XDR-SunRPC

A continuación se muestra el trozo de código de un programa XDR que envía unos datos a través de un socket.

```
int main() {
	Entero i;
	int sTCP;
	-- HUECO 1 --
	XDR  operacion;
	...

	...

	printf("\n Tipo Entero\n----------\n\nIntroduce un Entero: ");
	scanf("%d", &i);

	connect(sTCP, (struct sockaddr *) &servidor, sizeof(servidor));

	-- HUECO 2 --

	if (xdr_Entero(&operacion, &i) != TRUE) {
		fprintf(stderr, "Error al escribir el Entero\n");
	}

	...

	return 0;
}
```

¿Qué instrucción o instrucciones faltan en las líneas marcadas como -- HUECO 1 -- y -- HUECO 2 --?

Solución:

```
-- HUECO 1 --
FILE *fSock;
-- HUECO 2 --
fSock = fdopen(sTCP, "w+");
xdrstdio_create(&operacion, fSock, XDR_ENCODE);
```

Explicación:

Para poder usar los filtros XDR es necesario crear primero la operación XDR. Para crear la operación hay que llamar a la función "xdrstdio_create()". Como primer parámetro necesita la dirección de un tipo de dato XDR. Fijándonos en la declaración de los datos vemos que la varible es "operacion". El segundo parámetro se explicará a continuación. Y el tercer parámetro es el tipo de operación que se desea realizar. Como es un envío de datos, usaremos "XDR_ENCODE".

El segundo parámetro necesita un "FILE *", y nosotros lo que tenemos es el descriptor de fichero asociado al socket ("sTCP"). Por tanto, debe asociarse el descriptor de fichero con un "FILE *". Para ello usaremos la función "fdopen()". Como primer parámetro pasaremos "sTCP" y el segundo parámetro será "w+" para permitir la escritura sobre el stream.

Ahora sólo falta declarar la variable de tipo "FILE *" en la declaración de variables. Así, en -- HUECO 1 -- declaramos "fSock", que será la variable a usar en las llamadas de -- HUECO 2 --.

Problema 24. SunRPC

Indica cuales de las siguientes afirmaciones son ciertas cuando estamos hablando de Multidifusión de RPC:

a)la llamada se realiza a través del localizador, no directamente

b)una RPC en multidifusión sólo devuelve un resultado de uno de los servidores instalados

c)se puede utilizar tanto TCP como UDP

d)todas las llamadas en multidifusión utilizan por defecto el esquema de seguridad AUTH_UNIX

Solución:

Las afirmaciones ciertas son a) y d)

Explicación:

b) es falsa porque en realidad se puede devolver un resultado por cada servidor instalado
c) es falsa porque sólo se puede utilizar UDP

Problema 25. SunRPC

Dada la siguiente estructura en XDR, cuáles son los tamaños máximo y mínimo que puede tener la estructura? ¿Qué valores hacen que se tomen los valores mínimos y máximos?

```
union prueba
switch(int que) {
      case 1: hyper x;
      case 2: double y;
      case 3: struct z{
                  string texto<>;
                  lista * siguiente;
            };
      case 4: void;
}
```

Solución:

Mínimo = 4, máximo = ínfinito
Mínimo -> 3 y 4, máximo -> 3

Explicación:

El valor mínimo se da en los casos 3 y 4, en el 3 si la lista enlazada que se representa es nula (sólo codificaría un 0), que son 4 bytes y en el caso 4 porque un void ocupa 4 bytes. Los casos 1 y 2 ocupan siempre 8 bytes.

El valor máximo es infinito ya que es una lista enlazada que puede tener tantos elementos como quiera, por lo que el tamaño máximo se puede decir que es infinito.

[Nota del P.: El valor mínimo es correcto, pero no la explicación ya que se tiene en cuenta un valor erróneo del tipo void. El valor máximo no está bien definido ya que desconocemos cómo es la estructura lista. Dependiendo de esa estructura, el valor máximo podría ser correcto.]

Problema 26. XDR-SunRPC

Se tiene el siguiente fichero EjemploExamen.x:

```
union EjemploUnionXDR switch(int d)
      {
        case 1: int i;
        case 2: float f;
        default: string cadena<20>;
      };

struct datoSuma
{
  int sumando1;
  int sumando2;
};

program PRUEBA{
   version CALCULA
   {
      int suma(datosSuma) = 1;
       double dividir(float) = 2;
   } = 3;
}=0x121ADFBF
```

❑ **26.1** Completar el siguiente código de 'cliente.c' para realizar una invocación remota al servicio 'suma' en la máquina 'servidorEjemplo.com' y comprobar que no se produce ningún error en la llamada.

```
#include "EjemploExamen.h"
main(int narg, char *args[])
{
   CLIENT *clnt;
     int *resul;
     datosSuma datos;
     ...
}
```

Solución:

```
clnt = clnt_create("servidorEjemplo.com",PRUEBA,CALCULA,"tcp");

datos.sumando1 = 3;
datos.sumando2 = 7;
resul = suma_3(&datos,clnt);

if(resul==NULL)
{
   clnt_perror(clnt,"Fallo en la llamada:");
   exit(-1);
}
```

Explicación:

Se inicializa la estructura 'clnt' llamando a la función 'clnt_create'. A esta se le pasa el nombre de la máquina ("servidorEjemplo.com"), la constante que identifica al servidor (PRUEBA), la constante que especifica el número de versión del servidor (CALCULA) y el protocolo de transporte, como no se indica nada, se puede poner "tcp".

A continuación se rellenan los campos del tipo de dato 'datosSuma', para pasárselo al servicio 'suma'. Este se llama con el nombre que tiene declarado en el interfaz más un sufijo, que indica el número de versión, '_3'. Lo que recibe es un dato de tipo 'datosSuma', le pasamos una referencia, puesto que es lo que debre recibir la implementación en C. El servicio devuelve un entero, en la implementación C se recibirá un puntero a entero, por tanto, se recoge en la variable 'resul'. Se comprueba si el resultado devuelto es NULL, en cuyo caso, se llama a la función 'clnt_perror()' la cual envía un mensaje a la salida estándar indicando por qué falló la llamada al procedimiento, y se finaliza el programa.

Problema 27. SunRPC

Definir un array de longitud fija de 3 elementos cada uno de los cuales es una cadena de longitud variable de 20 elementos. ¿Como se codificaría en XDR uno de estos tipos si no contiene nada?

Solución:

```
typedef string caracteres<20>;
caracteres miArray[3];

00 00 00 00  00 00 00 00  00 00 00 00
```

Explicación:

Tendra tres elementos correspondientes a la longitud del array fijo. Cada uno de los caracteres se codificará con un byte que indique el número de caracteres(long. cadena), en este caso cero.

Problema 28. SunRPC

¿Que tipos de peticiones admite la función clnt_control y cual es su significado?

Solución:

```
- CLSET_TIMEOUT : Establece cual es el tiempo máximo de espera para
una respuesta por parte del servidor.

- CLGET_TIMEOUT : Recupera el tiempo máximo de espera para una
respuesta por parte del servidor.

- CLGET_SERVER_ADDR : Obtiene los datos de la máquina del servidor.

- CLSET_RETRY_TIMEOUT : Fija el tiempo de espera del servidor antes
de reintentar una llamada.

- CLGET_RETRY_TIMEOUT : Devuelve el tiempo de espera para reintentar
una llamada que esta activo.

Los 2 últimos valores solo estan disponibles para estructuras
CLIENTE inicializadas con UDP.
```

Problema 29. XDR-SunRPC

¿Cómo se puede conseguir que un clientes realice múltiples RPC sin esperar por respuesta del servidor?

Solución:

Usando clnt_control y cumpliendo una serie de condiciones:

- El tiempo de espera del cliente tiene que ser 0
- El servidor no debe enviar ninguna respuesta
- Se debe utilizar un protocolo de respuesta fiable como tcp
- Se recomienda terminar la serie de peticiones con una RPC "normal" para garantizar el envío de las anteriores.

Problema 30. XDR-SunRPC

Explica la sintaxis genérica de un filtro XDR

Solución:

Los filtros de tipos simples y los de tipos compuestos tienen la misma sintaxis.

bool_t xdr_tipo(XDR *xdrs,tipo *dato)

El primer parámetro defiine:
- El sentido de la conversión (codificación o decodificación)
- El lugar donde se escribirá o leerá el dato en codificación XDR.

El segundo parámetro es un puntero al dato en su "codificación C"

Problema 31. XDR-SunRPC

En el siguiente programa rellenar el código para poder enviar un entero por el socket.

```
#includes...
main()
{
 int sock, dato; struct sockaddr_in dir;
 FILE *fsock; XDR hxdr;

 sock=socket(PF_INET, SOCK_STREAM, 0);
 dir.sin_family=AF_INET; dir.sin_port=htons(15000);
 dir.sin_addr.s_addr=inet_addr("156.35.151.2");

_____
_____
_____

 dato=1228;
 xdr_int(&hxdr, &dato);
 fclose(fsock); close(sock);
}
```

Solución:

```
fsock=fdopen(sock, "w+");
xdrstdio_create(&hxdr, fsock, XDR_ENCODE);
connect(sock, &dir, sizeof dir);
```

Problema 32. SunRPC

Si tenemos esta estructura en XDR:

```
 struct s
 {
   int v<5>;
   string texto<20>;
 };
```

¿Que código daría lugar en C?

Solución:

```
struct s {
    struct {
        u_int v_len;
        int *v_val;
    } v;
    char *texto;
};

typedef struct s s;
```

Explicación:

Las estructuras de XDR se convierten en una estructura en C, en la que cada campo se convierte por separado.

Un array de tamaño variable pasa a convertirse en una estructura que tiene un campo nombre_len donde se guarda la longitud del array y otro campo nombre_val que es el array en sí.

Un string de XDR se convierte en un char* en C

Problema 33. XDR-SunRPC

¿Qué sentencia se debe incluir en el cliente para crear las credenciales UNIX?

Solución:

```
clnt->cl_auth = authunix_create_default();
```

Problema 34. XDR-SunRPC

Dado el siguiente fragmento de codigo:

```
main ()
{
  CLIENT *clnt;
  struct timeval espera;
        .....
  clnt=clnt_create(maquina, DUPLICA,DUPLIVERS,"tcp");
        .......
  espera.tv_sec=40;
  espera.tv_usec=0;
  if (clnt_control(clnt,CLSET_RETRY_TIMEOUT,&espera)==FALSE) {
    printf("Error");
    exit(-1);
  }
  resul=duplica_1 (&dato, clnt);
         ......
```

Indicar la linea y explicar el motivo de una instrucción incorrecta

Solución:

```
Línea 10: CLSET_RETRY_TIMEOUT es solo para estructuras CLIENT
inicializadas con UDP
```

Problema 35. XDR-SunRPC

Define una variable de tipo doce, e introduce valores en ella.
typedef float doce<12>;

Solución:

```
doce v;
v.doce_val=malloc(7*sizeof(float));
for(int i=0; i<7; i++)
  v.doce_val[i]=5.5+i;
v.doce_len=7;
```

Problema 36. XDR-SunRPC

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;
```

❑ **36.1** Dado el siguiente código de la interfaz y un cliente creado con rpcgen, rellena los espacios en blanco.

INTERFAZ.H

```
#include <rpc/rpc.h>
#define SERVIDOR_CALCULO ((unsigned long)(0x201ABCBF))
#define SC_TERCERA ((unsigned long)(3))
#define duplica ((unsigned long)(1))
extern int * duplica_3();
#define ENTRE3 ((unsigned long)(2))
extern float * entre3_3();
extern int servidor_calculo_3_freeresult();
```

CLIENTE.C

```
#include "interface.h"
main (int narg, char *args[])
{
  CLIENT *clnt;
  int *resul, dato;
  char *maquina="maquina.servidor.es";
  clnt= _____ 1

  if (clnt== NULL) {
    _____ 2

    exit(-1);
  }
  resul=duplica_3 (&dato, clnt);
  if (resul==NULL) {
_____ 3

exit(-1);
}
```

```
printf("\nEl resultado es %d\n", *resul);
```

4

Solución:

[Nota del P: El número de versión en la descripción del interfaz no coincide con el usado en el programa.]

```
1 clnt_create(maquina,SERVIDOR_CALCULO,SC_TERCERA,udp");
2 clnt_pcreateerror(maquina);
3 clnt_perror(clnt,"Fallo en la llamada:");
4 clnt_destroy(clnt);
```

Problema 37. SunRPC

Sabiendo que se utiliza las RPC de Sun Microsystems en nivel SIMPLIFICADO, completa el siguiente código incluido en el fichero servicios.c que calcula el IVA para un valor real dado:

```
---(1)---calcIVA(---(2)---)
{
      ---(3)---
      resultado= ---(4)--- * 1.16;
      return (---(5)---);
}
```

Solución:

```
---(1)--- float *
---(2)--- float *valor
---(3)--- static float resultado;
---(4)--- *valor
---(5)--- &resultado
```

Explicación:

Hay que tener en cuenta que el resultado y el parámetro son SIEMPRE un puntero al tipo de dato que maneja el filtro XDR, en este caso float puesto que la multiplicación por 1.16 genera un número real. Por otro lado, la variable de retorno siempre tiene que ser declarada de tipo static. Además en el nivel simplificado, sólo hay un único parámetro de entrada al servicio.

Problema 38. XDR-SunRPC

¿Se pueden usar sockets en XDR?

Solución:

El uso de sockets en XDR no se diferencia en mucho del uso de ficheros, ya que XDR utiliza descriptores de ficheros y los sockets no son mas que descriptores. Los sockets utilizados deben ser de tipo stream, es decir, es una comunicación orientada a conexión. La declaración y la inicialización del socket de comunicación es exacto que en C. El proceso sería el siguiente:

-Declaración del socket.
-Inicialización del socket.
-Asociación de un descriptor de fichero al socket mediante la función fdopen.
-Llamada a la función connect del socket.
-Intercambio de información mediante la llamada a los filtros XDR
y forzar el envío mediante llamadas a la función fflush.
-Finalización de la conexión mediante llamadas a las funciones fclose y close.

Problema 39. XDR-SunRPC

¿Qué fallos se pueden producir en una RPC, cuándo y si son detectables?

Solución:

Durante la petición del cliente puede ocurrir que el cliente no localice la RPC. Es detectable por el cliente.

Una vez realizada la petición por el cliente esta no llega al servidor. No es detectable ni por el cliente ni por el servidor.

Durante la ejecución del servicio el servidor falla. No es detectable ni por el cliente ni por el sevidor.

Una vez realizado el servicio el servidor envía la respuesta pero esta no llega al cliente. No es detectable ni por el cliente ni por el servidor.

Problema 40. XDR-SunRPC

Completa la tabla referente al problema "tolerancia a fallos" que se pretende resolver con RPC

CLIENTE	SERVIDOR	
Reintenta RPC	Filtra duplicados de RPC	Acción realizada
	No aplicable	
	NO	
	SI	

Solución:

	SERVIDOR	
CLIENTE	Reintenta RPC	Filtra duplicados de RPC	Acción realizada
SI	SI	No aplicable	Retransmisión respuesta
SI	NO	Reejecución servicio
NO	No aplicable	No aplicable

Problema 41. XDR-SunRPC

Tenemos el siguiente Servicio el cual realiza la suma de dos numeros complejos:

```
numComplejo * SumaComplex(numComplejo op1, numComplejo op2,struct svc_req *req)
{
      numComplejo resultado;
      resultado.a = op1.a + op2.a;
      resultado.b = op1.b + op2.b;
      return (&resultado);
}
```

Tenemos declarada previamente una estructura llamada numComplejo:

```
struct numComplejo{
      float a;
      float b;
}
```

Analiza el codigo del servicio e indica si esta correcto o incorrecto, en caso de encontrar errores solucionalos.

Solución:

Los errores encontrados son los siguientes:

1.- Unicamente se puede pasar un parametro de entrada al servicio, en este caso estamos pasando dos. Para solucionar este problema encapsularemos dentro de una misma struct los dos numeros complejos.

2.- La variable de retorno en este caso resultado, tiene que ser declarada siempre de tipo static.

El resultado sería el siguiente:

```
numComplejo * SumaComplex(ParnumComplejo * op,struct svc_req * req)
{
      static numComplejo resultado;
      struct numComplejo op1,op2;
      op1 = op->num1;
      op2 = op->num2;
      resultado.a = op1.a + op2.a;
      resultado.b = op1.b + op2.b;
      return (&resultado);
}

Las estructuras que se usaran seran las siguientes:

struct numComplejo{
      float a;
      float b;
}
struct ParnumComplejo{
      numComplejo num1;
      numComplejo num2;
}
```

Problema 42. DCE-RPC

Tenemos el siguiente código:

```
 char *cadena;
 error_status_t status;
 rpc_string_binding_compose(NULL, "ncacn_ip_tcp", "sirio.edv.uniovi.es",
"1781", NULL, &cadena, &status);
```

```
if (status) {
  fprintf(stderr, "Error en la creaci´on de la cadena");
  exit(-1);
}
printf("Cadena resultante: %s\n", cadena);
rpc_string_free(&cadena);
```

¿Cual sería el resultado mostrado por pantalla?

Solución:

"ncacn_ip_tcp:sirio.edv.uniovi.es[1781]"

Explicación:

Ejemplo sacado de los apuntes DCE-RPC(3).pdf

Problema 43. DCE-RPC

a)Describe brevemente los tres tipos de punteros que existen en el IDL de Microsoft.

b)¿Porque es necesario el uso de la funcion RpcSsAllocate() en un servidor que devuelve un puntero?

Solución:

a)1-Referencia: No puede ser NULL, siempre debe apuntar a la misma dirección y no puede existir otro puntero que apunte al mismo dato.

2-Puntero completo: Elimina las restricciones anteriores.

3-Puntero único: Elimina las dos primeras restricciones

b) Es necesario porque si no el servidor no podrá liberar la memoria, aunque se podría solucionar. Por ello se utiliza esta función que reserva memoria y posteriormente al finalizar el servicio es el stub del servidor el que se encarga de liberarla.

Problema 44. DCE-RPC

En la programación con DCE RPC, ¿cuál es el formato de la cadena de binding? Si el cliente desconoce alguno de esos datos, ¿cómo lo obtiene?

Solución:

El cliente debe conocer el protocolo de transporte utilizado, la máquina donde está ubicado el servicio y su endpoint o número de puerto. Una vez conocidos estos datos, el formato de la cadena de binding es:

"transporte:máquina[endpoint]"

Si el cliente desconoce el endpoint, puede obtenerlo interrogando al endpoint mapper.

Si desconoce la máquina donde se ejecuta el servicio puede obtenerla interrogando al Servicio de Directorio de Celda (CDS).

Problema 45. DCE-RPC

A continuación se muestra el trozo de código en el que se pide un servicio que recibe como parametro un array variable "vector":

```
float vector[25];
float min;
...
min=CalculaMinimo(manejador, vector, 10, 20);
...
```

Escribe el fichero de definición del interface (.idl) de la aplicación escrita para el DCE RPC.

Solución:

```
[
uuid(ba20f0fa-0c6d-13d1-07cc-1040bf7eca29),
version(1.0)
]
interface Ejemplo{
    float CalculaMinimo(
        [in, first_is(inicio), last_is(fin)] float vector[25];
        [in] long inicio;
        [in] long fin;
    );
}
```

Problema 46. DCE-RPC

Indica brevemente los pasos a seguir para inicializar un servidor con DCE.

Solución:

1. Registrar el interfaz.
2. Solicitar endpoint para el protocolo que sea.
3. Obtener vector de endpoint.
4. Registrar el endpoint.
5. Poner en espera de clientes.

Problema 47. DCE-RPC

A continuación se muestra un ejemplo de programación DCE RPC con enlazado implícito:

Fichero pregunta.idl

```
[
uuid(ba20f0fa-0c6d-13d1-07cc-1040bf7eca29),
version(1.0)
]
interface pregunta
{
long MostrarPregunta([in, string] char * txt);
}
```

Fichero pregunta.acf (para el cliente)

```
[
        implicit_handle(handle_t manejador)
]
interface pregunta{}
```

Fichero cliente.c

```
#include "pregunta.h"
int main(){
        error_status_t error;
        long int r1;
        rpc_binding_from_string_binding
("ncacn_ip_tcp:localhost",&error);

        r1=MostrarPregunta("Pregunta de examen");
        ...

}
```

Reescribe los ficheros que consideres necesarios para que el método de enlazado sea explícito.

Solución:

Fichero pregunta.acf (para el cliente)

```
[
        explicit_handle
]
interface pregunta{}

Fichero cliente.c

#include "pregunta.h"
int main(){
        error_status_t error;
        handle_t manejador;
        long int r1;
        rpc_binding_from_string_binding("ncacn_ip_tcp:localhost",
&manejador,&error);

        r1=MostrarPregunta(manejador,"Pregunta de examen");
        ...

}
```

[Nota del P: En el enunciado, la inicialización del manejador es incorrecta]

Problema 48. DCE-RPC

¿Qué 2 maneras existen de gestionar los errores al llamar a un procedimiento remoto?

Solución:

La manera clásica: se utiliza un parámetro adicional en la llamada a la función que indica si todo ha ido bien o ha habido un error. Hay que recordar que si se hace de esta manera, en el archivo de configuración (ACF) se debe especificar que funciones devolverán un código de status.

La manera moderna: utilizando excepciones. El problema que tiene esto es que conocer que error se produjo depende del lenguaje y que algunos compiladores (gcc entre ellos) no tienen soporte para excepciones. Sin embargo esto se soluciona mediante macros.

[Nota del P: No siempre es obligatorio que el código de error aparezca en el acf. Se puede incluir un código de error como parte del interface de la RPC en el idl]

Problema 49. DCE-RPC

¿Cómo podemos hacer que una función remota devuelva un puntero a los datos que el propio servicio ha creado?

Solución:

1-Que la función retorne un puntero
2-Que uno de los parámetros a la rutina sea puntero a puntero.

Explicación:

1-El servicio llamará a RpcSsAllocate() para crear memoria.
El stub de servidor se encargará de liberarla.

2- El puntero será de tipo ptr,
el parámetro será declarado de entrada/salida (in/out)
El cliente debe inicializar el puntero a NULL y pasar una referencia al puntero.
El servidor debe reserver memoria con RpcSsAllocate().

Problema 50. DCE-RPC

Considerando la siguiente imagen:

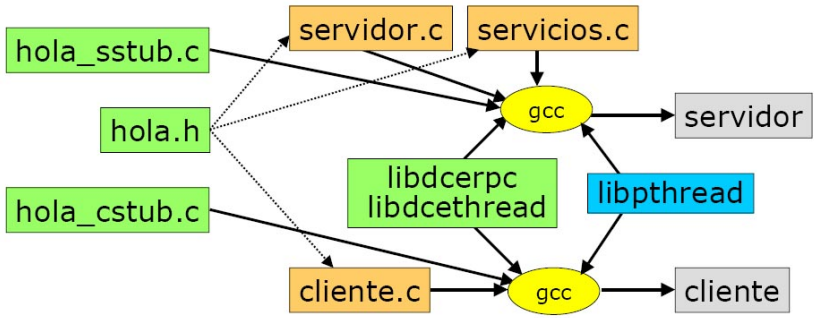

Figura 1

Indicar cual sería la secuencia de instrucciones de compilación necesarias, para obtener correctamente los ejecutables del cliente y del servidor en linux.

Solución:

```
gcc -c hola_sstub.c
gcc -c servidor.c
gcc -c servicios.c
gcc -o servidor hola_sstub.o servidor.o servicios.o -ldcerpc
        -ldcethread -lpthread
gcc -c hola_cstub.c
gcc -c cliente.c
gcc -o cliente hola_cstub.o cliente.o -ldcerpc -ldcethread
        -lpthread
```

Problema 51. DCE-RPC

¿Qué es DCE y qué nos proporciona?

Solución:

DCE es un entorno para computación distribuida y proporciona lo siguiente:
- Transmisión transparente de tipos definibles (sobre CDR)
- Llamadas a procedimientos remotos(RPC) basadas en threads y excepciones.
- Servicios construidos sobre las RPC (servicios adicionales sobre los básicos).

Problema 52. Sincronización-Seguridad

Sin conocer ta (instante de tiempo en el que ocurre el suceso a) ni tb: ¿Cuándo podemos afirmar que el suceso "a" ocurrió antes que el suceso "b" : a --> b ?

Solución:

Cuando a y b ocurren en la misma máquina y Na < Nb (siendo N el contador de interrupciones de la máquina para ese suceso)

Cuando "a" representa el envío de un mensaje y "b" su recepción.

Problema 53. Sincronización-Seguridad

¿Porqué los sistemas distribuidos son mas inseguros que los centralizados?

Solución:

El motivo principal para que los sistemas distribuidos sean más inseguros es una mayor exposición de la información que los sistemas centralizados. Los sistemas distribuidos tienen más puntos, especialmente la existencia de comunicaciones utilizando una red hacen este tipo de sistemas mucho más vulnerables.
El único punto a favor de los sistemas distribuidos en cuanto a seguridad se refiere es que la información está más dispersa y puede haber muchas puertas cerradas protegiendo la información, teniendo que vencerlas todas para acceder de forma ilícita a ella.

Problema 54. Sincronización-Seguridad

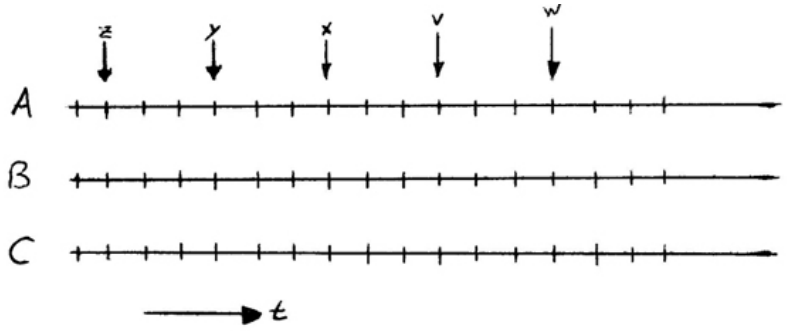

Figura 1

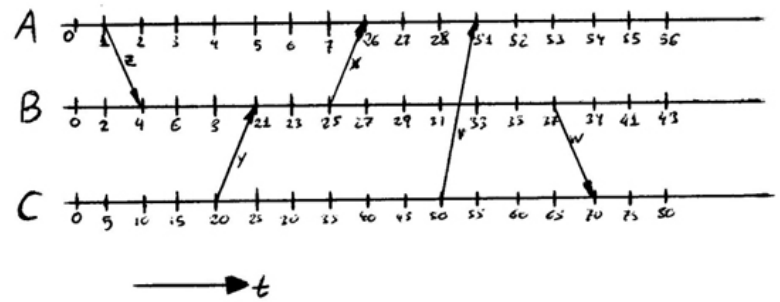

Figura 2

Rellenar el cronograma [1] con los numeros del contador de interrupciones de cada una de las máquinas A,B y C segun el algoritmo de Lamport.

Para ello tener en cuenta que:
- -Cada una de las marcas se corresponde a 10 mseg de tiempo [real]
- -Cada 10 mseg la máquina A genera 1 interrupción, la máquina B genera 2 interrupciones y la máquina C genera 5 interruciones.
- -En los momentos "z, y, x, v, w" se genera un mensaje
- -Un mensaje llega al destinatario en la siguiente marca del cronograma

Los mensajes mandados son los siguientes:
- z : Mensaje de A a B
- y : Mensaje de C a B
- x : Mensaje de B a A
- v : Mensaje de C a A
- w : Mensaje de B a C

Solución:

La solución se muestra en la figura [2]

Problema 55. Sincronización-Seguridad

¿Que es el UTC?

Solución:

(Universal Coordinated Time) Es el contador de segundos (incluido bisiestos) suministrado por el BIH que es referencia mundial de hora exacta

[N. del P: respuesta demasiado escueta ¿qué es un segundo bisiesto?]

Problema 56. Sincronización-Seguridad

¿Cuantas veces por minuto habrá que resincronizar un reloj de ratio máximo de derivada 0.03 si no queremos que la discrepancia sea mayor que 0.5s?

Solución:

7

Explicación:

Habrá que resincronizar el reloj cada 0.5/2*0.06 = 8.33 segundos así que en un minuto habrá que resincronizarlo 7 veces.

[N. del P.: La fórmula es correcta sólo si se sincroniza frente a otro reloj con el mismo ratio máximo de deriva]

Problema 57. Sincronización-Seguridad

¿Cuál de las siguientes afirmaciones es falsa?:

A) En la criptografía de clave pública no es necesario compartir la clave secreta en los dos extremos de la comunicación.
B) El ticket que utiliza Kerberos permite autenticar a un cliente por un tiempo indefinido.
C) Los sistemas distribuidos son más inseguros que los centralizados.
D) En RSA, N se calcula como P*Q y Z como (P-1)*(Q-1)

Solución:

B)

Explicación:

La afirmación B es falsa ya que los tickets Kerberos tienen definido un periodo de validez entre dos tiempos t1 y t2 determinados,y una vez pasado dicho periodo, expiran y por lo tanto no sirven para autenticar al cliente.

Problema 58. Sincronización-Seguridad

Para sincronizar el reloj de nuestro sistema, pedimos la hora exacta a un servidor de nuestra confianza cada día a las 00:00:00 horas.Un día determinado, y a las 00:00:06 recibimos como respuesta que la hora exacta es 00:00:10. ¿Cual es la nueva hora a la que debemos ajustar nuestro reloj?

Solución:

00:00:13

Explicación:

Estimamos el retardo como la mitad de la diferencia entre la hora local de la petición y la de la recepción de la respuesta.

[N. del P.: Algoritmo de Christian]

Problema 59. Sincronización-Seguridad

Explique brevemente en que consite el Algoritmo de Christian

Solución:

Este algoritmo permite aproximar el retardo de la red al recibir la hora UTC desde un servidor, ya que la hora que se recibe es la que tenía en el momento de responder pero no se tiene en cuenta el retardo de la red. Este algoritmo aproxima el retardo por medio de la fórmula (T1-T2)/2 donde T1 es la hora local cuando hizo la petición al servidor y T2 la hora local cuando recibió la respuesta.

[N. del P: La respuesta está mal. Confunde T1 y T2]

Problema 60. Sincronización-Seguridad

Indica cuales de las siguietes afirmaciones son ciertas:

1. Las claves publica y privada generadas por el algoritmo RSA a partir de los dos números primos P=3 y Q=11 son:

   a) Clave publica: Kp=3, N=14
      Clave privada: Ks=7, N=33

   b) Clave publica: Kp=3, N=33
      Clave privada: Ks=7, N=33

   c) Clave publica: Kp=7, N=14
      Clave privada: Ks=3, N=14

   d) Clave publica: Kp=7, N=33
      Clave privada: Ks=3, N=33

2. Sean a, b dos eventos. Na, Nb los valores de los contadores de las máquinas en las que se producen los eventos a y b cuando estos tienen lugar, y ta, tb el instante real en el que se producen los eventos a y b:

   Na>Nb --> ta>tb se garantiza si:

   a) a y b son eventos en máquinas diferentes.
   b) a y b son eventos en la misma máquina.
   c) a representa el envio de un mensaje y b su recepción.
   d) b representa el envio de un mensaje y a su recepción.

Solución:

1. b)

P=3 Q=11

N=P*Q=3*11=33 Z=(P-1)(q-1)=2*10=20

NCD(Kp,Z)=1 -> 20/1=0, 20/3=6,67 -> Kp=3

Ks=(Y*Z+1)/Kp con Y=1,2,3,.. de modo que Ks sea un entero.

Ks=(y*20+1)/3=21/3=7

Luego:

Clave publica: Kp=3 N=33
Clave privada: Ks=7 N=33

[N. del P.: La opción d) también sería correcta]

2. b),d)

Problema 61. Sincronización-Seguridad

¿Cual es la debilidad del sistema de cifrado con clave pública?

Solución:

Una persona puede situarse en medio de la comunicación y sustituir la clave pública de los extremos por la suya propia. De esta manera podría desencriptar los mensajes, pues irían dirigidos a él. Posteriormente usaría la clave pública de los extremos para encriptar el mensaje y enviarlo, de forma que el mensaje llegaría correctamente a su destino.

Índice