A continuación se recoge una selección de problemas aparecidos en los exámenes del curso 2008-2009. Tras cada pregunta aparece la caja de respuesta, en el mismo tamaño que tenía en el examen, y dentro de ella la solución. Para leerla debes poner la hoja ante un espejo o mirarla por detrás al trasluz. Seguidamente viene la explicación de cómo se llega a esa respuesta, por si tienes dudas.

❑ ¿Cuál o cuáles de los siguientes problemas en un sistema distribuído mejorarán si se evitan los recursos centralizados?

A) La escalabilidad
B) La fiabilidad
C) La flexibilidad
D) La heterogeneidad

A, B

Explicación: La escalabilidad mejora puesto que el mantener un recurso centralizado impone un límite al crecimiento del sistema (cuando ese recurso se sature). La fiabilidad también mejora, puesto que si el recurso centralizado fallara, afectaría a todo el sistema. La flexibilidad no depende de que haya recursos centralizados o no, sino a la posibilidad de que el sistema sea utilizado para otros fines diferentes de los iniciales. Finalmente la heterogeneidad empeoraría al eliminar recursos centralizados, puesto que se tiene más equipos en lugar de uno solo.

❑ Dada la siguiente secuencia de bytes, que codifica un texto en UTF-16, se pide proporcionar la secuencia de bytes que codificaría el mismo texto como UTF-8.

```
FE FF 00 6F 01 22
```

6F, C4, A2

Explicación: Los dos primeros bytes de la secuencia suministrada son el *byte order mark* (BOM) que sirven tan sólo para identificar la *endianity* de la máquina en la que se generó el fichero. En este caso, ya que el carácter BOM (cuyo código Unicode es `FEFF`) aparece como `FE FF`, se deduce que la máquina es Big Endian. Esto nos da la interpretación de los siguientes caracteres Unicode, que serán por tanto `U+006F` y `U+0122`.

El BOM no se codifica al pasarlo a UTF-8 (si se hiciera, tampoco sería del todo incorrecto y el resultado sería `EF BB BF`). El siguiente carácter `U+006F` es ASCII y por tanto en UTF-8 queda simplemente como `6F`. El último carácter `U+0122` requiere al menos 9 bits, por lo que se halla en el caso de dos bytes en UTF8. El primer byte debe comenzar por `110` para indicar que la secuencia se compone de dos datos, y el segundo debe comenzar por `10` para indicar que es parte de una secuencia. Quedan por tanto 5 bytes libres en el primer byte y 6 en el segundo, lo que nos da un total de 11 bytes. El dato `0122` escrito en binario con 11 bytes es el `001 0010 0010`, que al dividirlo en dos grupos de 5 y 6 bytes nos da `00100` y `100010`, respectivamente. Si anteponemos al primero el prefijo `110`, obtenemos `11000100` (`C4h`) y anteponiendo al segundo el prefijo `10`, obtenemos `10100010` (`A2h`). Estos son los dos bytes de la secuencia.

❑ Se tienen dos máquinas A y B cuyos relojes comienzan sincronizados, en el instante 0. La máquina A tiene un reloj exacto, mientras que la máquina B tiene un reloj defectuoso que va a la mitad de velocidad que el de A.

La máquina B envía a A un mensaje preguntándole la hora, en el instante en que el contador de B vale 1. En el instante en que su contador vale 4 recibe la respuesta de A, que contiene la estampa de tiempo 5. La máquina B actualiza entonces su contador aplicando el algoritmo de Christian.

— ¿Qué nuevo valor le asigna?

6,5

Explicación: De acuerdo con el algoritmo de Christian, la respuesta que llega del servidor se asume que se ha originado en el punto medio entre el instante en que el cliente la pidió y el instante en que la recibe. Estos instantes son 1 y 4, y el punto medio es 2,5. De modo que, cuando en el cliente el contador era 2,5, la hora "real" era 5 (la marca de tiempo que viene en el mensaje). Es decir, el cliente va 2,5 unidades retrasado. Si actualiza en el instante 4 su reloj, deberá sumarle 2,5 unidades para ponerlo en hora, y por tanto el nuevo valor será 6,5

— Si no se requiriera la hora exacta, sino sólo la consistencia lógica entre relojes aplicando el algoritmo de Lamport ¿cuál sería el valor a asignar al contador de B a la recepción de la respuesta del servidor?

6

Explicación: En este caso, si hay contradicción lógica entre los relojes al recibir el mensaje, basta asignar al contador de la máquina que recibe una unidad más que la marca de tiempo que viene en el mensaje. En este caso, el mensaje se recibe cuando el contador local es 4, y la marca de tiempo es 5. Ya que el mensaje no puede venir del futuro, hay contradicción que se resuelve asignando al contador de B el valor $5 + 1$, es decir, 6.

❑ Se implementa un servidor para el protocolo `echo`, tanto para protocolo TCP como UDP, con la particularidad de que devuelve cada línea al revés. Para ello hace uso de las siguientes funciones, cuyo código no se muestra:

- `inicializar_TCP(int puerto)` Crea un socket de tipo TCP, le asigna el puerto dado y retorna el socket creado. Si en cualquiera de las operaciones se produce un error, aborta la ejecución del programa.
- `inicializar_UDP(int puerto)` Análoga a la anterior, pero para protocolo UDP.
- `lee_linea_TCP(int s, char *buf, int max)` Recibe una línea por el socket de datos `s`, dejándola en el buffer `buf`. La línea incluye el retorno de carro, pero no terminador. `max` es el tamaño máximo admisible (si se reciben más caracteres, se trunca en ese punto). Retorna el número de caracteres leídos, o 0 si el cliente ha cerrado. En caso de error aborta la ejecución y no retorna.
- `lee_linea_UDP(int s, char *buf, int max, struct sockaddr *remite, int *lremite)` Análoga a la anterior pero para protocolo UDP. Devuelve también la dirección de la que proviene el datagrama.
- `al_reves(char *cad1, char *cad2)` Recibe dos punteros a *buffers*, el primero apunta a una cadena terminada en nulo, el segundo a un buffer vacío con al menos tanto espacio como el primero. La función retorna en el segundo buffer una copia invertida del primero.
- `maximo(a,b,c)` Recibe tres enteros y retorna el mayor.

El servidor da servicio tanto bajo TCP como UDP, para lo cual crea un socket de cada tipo y usa `select` para esperar en ambos. Bajo TCP admite un único cliente, debiendo esperar los demás a que finalice con él.

```
1  int sockTCP, sockUDP, sockDAT;
2  int puerto=7878;
3  int max, l;
4  FD_SET fd;
5  char recibo[80], envio[80];
6  struct sockaddr_in cli;
7  unsigned int lcli=sizeof(cli);
8  ...
9  sockTCP=inicializar_TCP(puerto);
10 sockUDP=inicializar_UDP(puerto);
11 sockDAT=0;
12
```

```
13 while (1) { // bucle infinito
14  // Inicializar el conjunto fd
15   FD_ZERO(&fd);
16   FD_SET(sockTCP, &fd);
17   FD_SET(sockUDP, &fd);
18   if (sockDAT) FD_SET(sockDAT, &fd);
19
20   // Inicializar max
21   max=
22   // Esperar conexiones
23   select(max, &fd, NULL, NULL, NULL);
24
25   // Atenderlas
26   if (FD_ISSET(sockTCP, &fd)) {
27
28   }
29  if (FD_ISSET(sockDAT, &fd)) {
30     l=lee_linea_TCP(sockDAT, recibo, 79);
31     if (l==0) {
32
33
34     } else {
35       recibo[l]=0;
36       al_reves(recibo, envio);
37       write(sockDAT, envio, strlen(envio));
38     }
39   }
40  if (FD_ISSET(sockUDP, &fd)) {
41     l=lee_linea_UDP(sockUDP, recibo, 79, &cli, &lcli);
42     recibo[l]=0; // Añadir terminador
43
44
45   }
46 } // del while
47 ....
```

— Completa la línea 21

```
max=maximo(sockTCP, sockUDP, sockDAT) + 1;
```

Explicación: Se está inicializando la variable `max`, la cual, observando cómo se usa más adelante en la llamada a `select`, ha de contener el máximo de los tres sockets existentes (`sockTCP`, `sockUDP` y `sockDAT`) más uno. Dicho máximo lo hallamos con ayuda de la función auxiliar `maximo()`

— ¿Qué falta en el hueco de la línea 27?

```
sockDAT=accept(sockTCP, NULL, NULL);
```

Explicación: El código por el que se pregunta se ejecuta en el caso de que haya actividad en el socket de escucha (la línea anterior comprueba si `FD_ISSET(sockTCP, &fd)`), por lo que lo que debe hacerse en este caso es aceptar al cliente que está intentando conectar. La respuesta es por tanto una llamada a `accept()` sobre `sockTCP`. Los dos parámetros para obtener la dirección del cliente pueden ponerse a `NULL` puesto que no interesan en este caso. El resultado devuelto por la función será el socket de datos por el cual comunicarse con el cliente, que debe ser asignado por tanto a la variable `sockDAT`.

— ¿Qué falta en la línea 32 y siguiente?

```
close(sockDAT);
sockDAT=0;
```

Explicación: Este código se ejecuta si `l` vale cero, esto es, si el cliente ha cerrado la conexión. En este caso tenemos que cerrar también el socket del lado del servidor y, muy importante, poner la variable `sockDAT` a cero, para evitar que en la línea 18 `sockDAT` sea añadido de nuevo a la lista de sockets a vigilar.

— ¿Qué falta en la línea 43 y siguiente?

```
al_reves(recibo, envio);
sendto(sockUDP, envio, l, 0, &cli, lcli);
```

Explicación: Este código se ejecuta si se reciben datos por el socket UDP. En este caso se recibe el texto enviado por el cliente, lo cual se hace en el código que muestra el programa, y lo que queda por hacer es "dar la vuelta" a ese texto llamando a la función `al_reves()`, para enviar el resultado de nuevo al cliente, haciendo uso de la función `sendto()`, ya que se trata de comunicación por el protocolo UDP.

— Escribe cómo sería el código de la función `inicializar_UDP` usada en este programa. (Por motivos de espacio, puedes omitir toda la comprobación de errores).

```
int inicializar_UDP(int puerto)
{
  int sock;
  struct sockaddr_in dir;

  sock=socket(PF_INET, SOCK_DGRAM, 0);
  dir.sin_family=AF_INET;
  dir.sin_port=htons(puerto);
  dir.sin_addr.s_addr=htonl(INADDR_ANY);
  bind(sock, (struct sockaddr *) &dir,
       sizeof(dir));
  return sock;
}
```

Explicación: Esta función debe crear un socket de tipo `SOCK_DGRAM` (UDP) y asignarle el número de puerto que recibe como parámetro y la IP `INADDR_ANY`. No debe ponerle en modo pasivo (`listen`) ya que se trata de un socket orientado a datagramas, no a conexión. Finalmente retornará el socket así creado.

— Explica si crees que sería posible escribir un programa que ofreciera la funcionalidad del anterior (es decir, atender clientes simultáneamente por los protocolos TCP y UDP sobre el mismo puerto) sin necesidad de usar `select()`. Razona tu respuesta.

En principio podría decirse que no es posible hacerlo con un solo proceso, dado que para aceptar un cliente en el socket TCP se requiere llamar a `accept()` y esta función bloquearía el proceso hasta que llegara un cliente por este protocolo, impidiendo al servidor responder a otros posibles clientes UDP. Gracias a `select()` se puede atender dos o más sockets "a la vez".
Si no nos limitamos a un único proceso, sí sería posible. Se pueden crear los dos sockets (TCP y UDP) y mediante `fork()` crear un segundo proceso. El padre podría ocuparse de los clientes TCP y el hijo de los UDP.

❑ Se implementa un servidor para la verificación de boletos de la Lotería Primitiva. El servidor utiliza XDR para recibir el boleto a comprobar y para enviar al cliente información sobre el sorteo y premios obtenidos. En concreto, en el siguiente listado se definen los tipos XDR utilizados en estas comunicaciones. El cliente envía al servidor un dato de tipo `Apuesta` en el que están contenidos los seis números jugados y el asignado para el reintegro. El servidor envía al cliente dos respuestas. En la primera (de tipo `Ganadora`) va simplemente cuáles han sido los números premiados en el último sorteo (lo cual es independiente del boleto que envió el cliente). En la segunda (de tipo `Premio`) va información acerca del premio que ha obtenido el boleto enviado por el cliente. En concreto, se le indica en qué categoría ha resultado ganador, un array variable que contiene los números que ha acertado, y el importe en euros del premio obtenido. La categoría es un entero que significa: `0`=no hay premio, `1`=seis aciertos, `2`=cinco aciertos más complementario, `3` (`4` y `5`)=cinco (cuatro y tres) aciertos (respectivamente), y `6`=obtenido el reintegro.

```
1  const   N_MAX=6; // Cantidad de números en una apuesta
2  typedef int   Numeros[N_MAX];
3  typedef int   Reintegro;
4  typedef int   Complementario;
5  typedef int   Acertados<N_MAX>;
6  typedef float Euros;
7  struct Apuesta {
8    Numeros   n;
9    Reintegro r;
10 };
11 struct Ganadora {
12   Numeros        n;
13   Complementario c;
14   Reintegro      r;
15 };
16 struct Premio {
17   int       categoria;
18   Acertados a;
19   Euros     e;
20 };
21 // Se define aqui también el puerto de escucha
22 const PUERTO_SERVIDOR=7777;
23 ──────────────────────────────────────
24                  Fichero loteria.x
```

B Se programa un cliente que solicite al usuario los seis números apostados y el reintegro y, tras consulta al servidor (cuya IP recibe por línea de comandos), le informe de qué números ha acertado y qué premio ha obtenido en euros, en caso de que haya obtenido alguno. El siguiente listado muestra parte de este cliente (se ha simplificado el código eliminando todas las comprobaciones de error). A

```
1  #include <arpa/inet.h>
2  #include "loteria.h"
3  int main(int argc, char *argv[])
4  {
5    int sock;
6    struct sockaddr_in dir;
7    int i;
8    Apuesta  apuesta;
9    Ganadora ganadora;
10   Premio   premio;
11   XDR op;
12   FILE* fs;
13
14   sock=socket(PF_INET, SOCK_STREAM, 0);
15   [redacted]
16   [redacted]
17   [redacted]
18   connect(sock, (struct sockaddr *) &dir, sizeof(dir));
19   printf("Introduzca sus %d números:\n", N_MAX);
20   for (i=0; i<N_MAX; i++)
21     scanf("%d", [redacted]);
22   printf("Introduzca el reintegro:\n");
23   scanf("%d", &apuesta.r);
24   printf("Enviando su apuesta al servidor\n");
25   [redacted]
26   [redacted]
27   [redacted]
```

```
28   fflush(fs);
29   printf("Recibiendo respuesta del servidor\n");
30   fs=fdopen(sock, "r");
31   xdrstdio_create(&op, fs, XDR_DECODE);
32   xdr_Ganadora(&op, &ganadora);
33   // Ignoramos información sobre la combin. ganadora
34   printf("Recibiendo datos del premio.\n");
35   premio.a.Acertados_val=NULL;
36   xdr_Premio(&op, &premio);
37   printf("Numeros acertados: ");
38   if ([redacted])
39      printf("Ninguno\n");
40   else [redacted]
41     printf("%d ", [redacted]);
42   // Imprimir premio
43   if (premio.categoria ==0)
44     printf ("No hay premio\n");
45   else
46     printf("Premio: %da categoría (%f euros)\n",
47            premio.categoria, premio.e);
48   return 1;
49 }
50
51 ──────────────────────────────────────
                  Fichero cliente.c
```

— ¿Qué falta en el hueco de la línea 21?

```
                                    &apuesta.n[i]
```

A

— ¿Qué falta en los huecos que comienzan en la línea 15?

```
                    dir.sin_family=AF_INET;
                dir.sin_port=PUERTO_SERVIDOR;
        dir.sin_addr.s_addr=inet_addr(argv[1]);
```

— ¿Qué falta en los huecos a partir de la línea 25?

```
                        fs=fdopen(sock, "w");
    xdrstdio_create(&op, fs, XDR_ENCODE);
                xdr_Apuesta(&op, &apuesta);
```

— Reescribe el código de la línea 38 y siguientes completando también lo que está tapado.

```
                if (premio.a.Acertados_len==0)
                    printf("Ninguno\n");
                                else
  for (i=0; i<premio.a.Acertados_len; i++)
printf("%d ", premio.a.Acertados_val[i]);
```

— ¿Cuántos bytes envía el servidor en total en cada uno de los siguientes casos para el usuario?

Acertó los 6 números: 68	No acertó ninguno: 44

— Durante el envío de la estructura `Premio`, los cuatro primeros bytes transmitidos son `00 00 00 03` ¿Cuáles son los cuatro siguientes?

```
                                    05 00 00 00
```

Explicación: Los cuatro primeros bytes enviados son la codificación XDR del campo `Categoria`, que es un entero. El valor de este entero por tanto es 3. Esta categoría de premio, según el enunciado, significa que el boleto tenía cinco aciertos. Puesto que el siguiente campo de la estructura es un array de longitud variable conteniendo los números acertados, y lo primero que se transmite para los arrays variables es el número de elementos que contienen, se transmitirá en este caso un 5 (pues el array deberá tener 5 elementos, que son los 5 números acertados).

A Seguidamente se da parte del código del servidor, en el cual se ha omitido toda la comprobación de errores y el código de las funciones auxiliares en las que el servidor consulta a la base de datos para averiguar los números premiados y el importe de cada premio. Estas funciones auxiliares son:

- `obtener_ganadora(Ganadora *)` recibe un puntero a través del cual devuelve una estructura convenientemente inicializada con la combinación ganadora, complementario y reintegro.
- `obtener_resultado(Apuesta, Premio * )` recibe una estructura `Apuesta` con los números a comprobar y un puntero a una estructura `Premio` en la que retorna la respuesta a enviar al cliente.

```
1  int main(int argc, char *argv[])
2  {
3    int sock, sdat;
4    struct sockaddr_in dir;
5    int i;
6    Apuesta  apuesta;
7    Ganadora ganadora;
8    Premio   premio;
9    XDR op;
10   FILE* fs;
11
```

```
12    sock=socket(PF_INET, SOCK_STREAM, 0);
13    // Omitida la inicialización de dir
14    [      ]([      ], (struct sockaddr *) &dir, sizeof(dir));
15    listen(sock, SOMAXCONN);
16    while (1) {  // Bucle infinito de atención a clientes
17      sdat=accept(sock, NULL, NULL);
18      printf("Aceptado un cliente\n");
19      fs=[                              ]
20      xdrstdio_create(&op, fs, XDR_DECODE);
21      xdr_Apuesta(&op, &apuesta);
22      obtener_ganadora(&ganadora);
23      fs=[                              ]
24      xdrstdio_create(&op, fs, XDR_ENCODE);
25      printf("Enviando numeros ganadores\n");
26      xdr_Ganadora(&op,&ganadora);
27      obtener_resultado(apuesta, &premio);
28      printf("Enviando premio\n");
29      xdr_Premio(&op, &premio);
30      printf("Desconectando\n");
31      fclose(fs);
32      close(sdat);
33    } // fin del bucle infinito
34    close(sock);
35  }
```

— Completa la línea 14

```
bind(sock, (struct sockaddr *) &dir,
                    sizeof(dir));
```

— ¿Qué falta en las líneas 19 y 23 del listado anterior?

Línea 19:
```
fdopen(sdat, r");
```
Línea 23:
```
fdopen(sdat, "w");
```

— Indica entre qué dos líneas del listado anterior habría que incluir código para que este servidor pueda atender hasta 5 clientes simultáneos, con la técnica del `fork()` previo. Escribe también cuál sería el código a añadir en ese punto.

Entre las líneas 15 y 16 se añadiría el siguiente código:

```
signal(SIGCHLD, SIGIGN);
for (i=0;i<4;i++)
  if (fork()==0) break;
```

— Un usuario quiere comprobar su boleto, en el que ha jugado los números 20, 21, 22, 23, 24 y 25 (Reintegro=3) mediante el cliente antes descrito. Si en ese día la combinación ganadora ha sido 4, 10, 22, 30, 34, 37 (Reintegro=2), completa el siguiente cuadro con los 8 primeros bytes de la estructura `Apuesta` enviada por el cliente y los 8 primeros bytes de la estructura `Premio` enviada por el servidor (escribe cada byte en una casilla y en hexadecimal).

Apuesta	00	00	00	14	00	00	00	15
Premio	00	00	00	00	00	00	00	01

Explicación: El primer campo de la `Apuesta` es un array de longitud fija con los números. Por tanto los 4 primeros bytes codifican el primer número, que es el 20 (`14h`) y los 4 siguientes el segundo, que es el 21 (`15h`).
El primer campo del `Premio` es la categoría del premio. En este caso, comparando la lista de números jugados con los premiados, vemos que el usuario sólo ha acertado uno, y no ha obtenido tampoco el reintegro, por lo que la categoría del premio será cero. Ya que se codifica con un entero, los cuatro primeros bytes son `00 00 00 00` . El siguiente campo de la estructura `Premio` es el array de longitud variable que contiene los números acertados. En este caso contendrá sólo el 22. La codificación de un array de longitud variable comienza por la longitud del array, que en este caso es 1, por lo que los cuatro bytes siguientes serán `00 00 00 01`.

Para preparar su respuesta, el servidor hace uso de la función auxiliar `obtener_acertados`. La función recibe una copia de los números jugados en el boleto (parámetro de tipo `Numeros`) y devuelve cuáles de ellos han sido acertados (omitiendo los que no) en un array de longitud variable (tipo `Acertados`). Esta función a su vez hace uso de otras dos funciones auxiliares: `contar_acertados()` que retorna cuántos números acertados contiene la apuesta, y `comprobar_numero()` que recibe como parámetro uno de los números de la apuesta y retorna 1 si ese número está entre los premiados, y 0 si no.

```
1   Acertados obtener_acertados(Numeros apuesta)
2   {
3     Acertados cuales;
4     int i=0, j=0;
5
6     cuales.Acertados_len = contar_acertados(apuesta);
7     cuales.Acertados_val =
8           [                                    ]
9     for (i=0; i<N_MAX; i++)
10      if (comprobar_numero(apuesta[i])) {
11        [                                    ]
12        j++;
13      }
14    return cuales;
15  }
```

— Escribe lo que falta en la línea 11

```
cuales.Acertados_val[j]=apuesta[i];
```

— ¿Qué falta en la línea 8?

```
malloc(cuales.Acertados_len*sizeof(int));
```

— Suponiendo que la combinación ganadora se encuentra almacenada en una variable global llamada `GANADORA`, que es un array de longitud fija (`N_MAX`), escrbe cómo sería el código de la función `comprobar_numero` que recibe como parámetro el número (`int`) a comprobar y retgorna `1` si el número está premiado y `0` si no.

```
int comprobar_numero(int n)
{ int i;
  for(i=0;i<N_MAX;i++)
    if (GANADORA[i]==n)
      return 1;
  return 0;
}
```

— Escribe el código de la función `contar_acertados`, que es llamada en la línea 6. Puedes hacer uso de la función `comprobar_numero()`.

```
int contar_acertados(Numeros n)
( int i, aciertos=0;
  for (i=0;i<N_MAX;i++)
    aciertos+=comprobar_numero(n[i]);
  return aciertos;
}
```