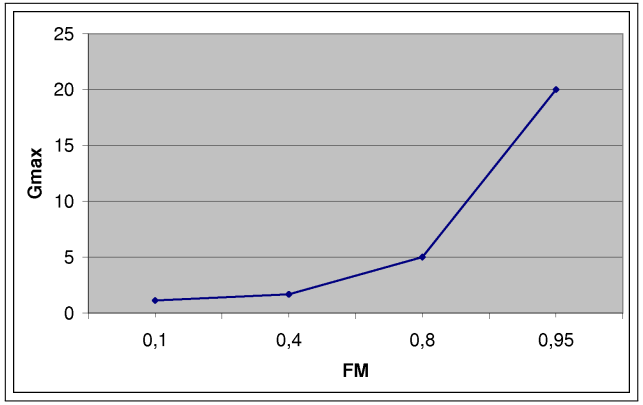
El examen está compuesto de dos bloques de 12 cuestiones
Para aprobar o compensar con el segundo parcial será necesario tener un mínimo de 4 cuestiones correctas en cada bloque
Las respuestas con valores no enteros deben tener 3 decimales
Todas las cuestiones tienen la misma puntuación ($10/24 = 0{,}4167$)

Bloque I

❑ Se tiene un programa que tarda 20 segundos en ejecutarse sobre un procesador antes de aplicar cierta mejora. Dibujar la gráfica de ganancia máxima que se puede conseguir con la mejora en función de la fracción de mejora, representando los puntos correspondientes a tiempos de aplicación de la mejora de 2, 8, 16 y 19 segundos. Escalar los ejes de la gráfica.



❑ En la tabla que sigue se dan las mediciones en segundos para cada uno de los dos programas que constituyen la carga de trabajo de dos computadores ¿Cuál será la ganancia de velocidad con dicha carga de trabajo al sustituir el computador 1 por el 2 si los programas se ejecutan durante el mismo tiempo en el computador 1?

Programa	Comp1	Comp2
A	30	30
B	100	50

1,333

Explicación: Primero se traducen los pesos de cada uno de los programas en el computador 1 (50 %-50 %) al computador 2 (66,67 %-33,33 %). Después se ponderan las ganancias individuales para cada programa con el peso de cada uno de ellos en la carga.

❑ Al incorporar cierta mejora a un computador se produce un incremento de velocidad del 40 %. Si el CPI sin mejora es de 1,4 y el efecto de la mejora sobre el número de instrucciones es un aumento del 10 % ¿cuál será el CPI resultante con la mejora?

0,909

Explicación: La ganancia dada será el resultado de comparar los tiempos sin mejora y con mejora teniendo en cuenta sus 3 componentes: NI, CPI y T.

❑ Se tiene un programa que consta de un bucle que se ejecuta durante 100000 iteraciones y cuyo cuerpo contiene el número de operaciones flotantes descrito en la tabla que sigue. También se adjunta una tabla de normalización de operaciones flotantes. Teniendo en cuenta que el tiempo de ejecución del programa es de 7 segundos:

Tipo	Sum.	Mult.	Div.	Raiz Cuad.
N Op. Flot.	300	200	150	50

Tipo	Sum./Res./Mult.	Div.	Raiz Cuad./Exp./Log.
OF Norm.	1	4	8

A) ¿Cual será el valor del rendimiento en MFLOPS nativos?
B) ¿Cual será el valor del rendimiento en MFLOPS normalizados?

A: 10 MFLOPS B: 21,429 MFLOPS

Explicación: Aplicación directa de la definición de MFLOPS y MFLOPS normalizados

❑ En la tabla del anexo pueden verse los resultados correspondientes a la evaluación de un computador para un conjunto de benchmarks. En función de los mismos contestar a las siguientes preguntas:

— ¿Que ganancia de velocidad respecto a la máquina de referencia supone pasar de ejecutar la aplicación 1445.gobmk a la 462.libquantum si se utilizan las opciones normales de compilación?

12,768

Explicación: Comparando los ratios de velocidad de base correspondientes a dichas aplicaciones se obtiene la ganancia de velocidad solicitada.

— ¿Que ganancia de velocidad respecto a la máquina de referencia supone pasar de ejecutar la aplicación 1445.gobmk a la 462.libquantum si se utilizan las opciones más agresivas de compilación?

11,721

Explicación: Comparando los ratios de velocidad de pico correspondientes a dichas aplicaciones se obtiene la ganancia de velocidad solicitada.

— ¿Que ganancia de velocidad respecto a la máquina de referencia supone pasar de utilizar las opciones normales de compilación a utilizar las más agresivas para el benchmark 462.libquantum?

1

Explicación: Comparando los ratios de velocidad base y de pico correspondientes a dicha aplicación se obtiene la ganancia de velocidad solicitada.

— ¿Que ganancia de velocidad respecto a la máquina de referencia supone pasar de utilizar las opciones normales de compilación a utilizar las más agresivas para el conjunto de benchmarks?

1,148

Explicación: Comparando los ratios de velocidad agregados de base y de pico se obtiene la ganancia de velocidad solicitada.

❑ Si se adopta una nueva tecnología de fabricación de transistores que consigue reducir en un 30 % el tamaño característico de los mismos:

A) ¿cuantos transistores podemos integrar en una superficie donde antes integrábamos 10000?
B) ¿cuál será el incremento porcentual de capacidad de cálculo por chip?

A: 20408 B: 191,545 %

Explicación: A) La proporción o ganancia entre el nuevo número y el original será inversamente proporcional al cuadrado del nuevo tamaño característico relativo (0,7). B) La proporción o ganancia entre la nueva capacidad y la original será inversamente proporcional al cubo del nuevo tamaño característico relativo (0,7).

❑ Suponer que un procesador ejecuta una carga de trabajo formada por un 50 % de instrucciones aritmético/lógicas, un 30 % de carga/almacenamiento y un 20 % de control. Otros datos de interés son:

CPI ideal = 1
Tasa de fallos en la cache de datos = 5 %
Penalización por fallo en la cache = 30 ciclos

¿Cuál será el CPI en condiciones reales?

1.45

Explicación: CPI = CPI ideal + ciclos perdidos por instrucción = 1 ciclos + [0.3(mem/inst) x 0.05(penal/mem)] x 30(ciclos/penal) = 1 ciclos + 0.45 ciclos = 1.45 ciclos

❑ Indica lo que falta para completar cada una de las siguientes afirmaciones sobre la memoria cache:

1. Al aumentar el grado de asociatividad la tasa de fallos ...
2. Al aumentar el tamaño de bloque la penalización por fallo de cache ...

1: disminuye 2: aumenta

❑ Se dispone de un diseño de cache unificada y se quiere conocer la tasa de fallos a que daría lugar la alternativa de un conjunto de caches independientes de instrucciones y datos, contando para ello con la siguiente información:

% fallos caché unificada (c.u.) = 16 %
% fallos de cache instrucciones = 106 % fallos c.u.
% fallos de cache datos = 65 % fallos c.u.
% accesos a instrucción en el programa = 40 %

13 %

Explicación: La tasa de fallos del conjunto de caches independientes se obtiene como suma de las tasas de fallos de instrucciones y datos ponderadas con el peso de los accesos a unas y otros.

Bloque II

❑ Se implementa una aplicación cliente-servidor en la que el servidor realiza una operación de cálculo entre dos enteros de 16 bits que le envía el cliente, y retorna a éste el resultado (también un entero de 16 bits).

Para evitar dependencias de la arquitectura, el envío de datos y respuestas se realiza en formato *Big Endian*, siendo reponsable cada parte de la adaptación a este formato de las variables involucradas en la comunicación, usando para ello las macros apropiadas.

El siguiente listado muestra el código de la función que implementa esta comunicación en el lado del cliente. Los parámetros recibidos son el socket (que se asume ya conectado) y los dos enteros a enviar. La función retorna el entero respondido por el servidor.

```
1  int enviar_y_recibir(int s, short int a, short int b)
2  {
3    short int result,
4              aa=          ,
5              bb=          ;
6    write(s, &aa, sizeof(a));
7    write(s, &bb, sizeof(a));
8    read(s, &result, sizeof(result));
9    return(                );
10 }
```

El servidor implementa diferentes operaciones sobre los dos datos recibidos. En el siguiente listado se muestra el caso de la suma, las otras son análogas. Observar que también cierra el socket de datos.

```
1  void servicio_sumar(int s)
2  {
3    short int x,y,res;
4    read(s, &x, sizeof(x));
5    read(s, &y, sizeof(y));
6    res=                                ;
7    write(s, &res, sizeof(res));
8    close(s);
9  }
```

El servidor crea varios sockets de escucha, cada uno sobre un puerto diferente, correlativos a patir del que se le indique desde línea de comandos, y permanece a la espera en cualquiera de ellos haciendo uso de la función `select()`. Según el puerto al que se conecte el cliente, se elige cuál de las operaciones debe realizarse, a través de un array de punteros a funciones. El siguiente listado muestra el bucle principal del servidor, que hace uso de las siguientes funciones de apoyo cuyo código no se muestra:

- `maximo(int *, int)` Recibe un array de enteros y el contador de cuántos elementos tiene, y devuelve el valor máximo entre esos enteros.
- `inicializar_sockets(int *, int, int)` Recibe un array de enteros y un contador cuántos elementos tiene. Inicializa cada entero del array con un desciptor de socket de escucha, cada uno escuchando en un puerto diferente, comenzando por el número de puerto que recibe como tercer argumento.
- `rellenar_fd(fd_set *, int *, int)` Recibe un puntero a una estructura `fd_set` que debe ser inicializada, un array de enteros que representan descriptores de sockets, y un entero que indica la longitud del array. Se ocupa de inicializar la estructura `fd_set` de modo que incluya todos los descriptores que vienen en el array.

```
1  #define NUM_SERVICIOS 4
2  int main(int argc, char *argv[])
3  {
4    // Array de sockets de escucha, y socket de datos
5    int sockets[NUM_SERVICIOS], sdat;
6    // Array de punteros a funciones (servicios)
7    void (*servicios[NUM_SERVICIOS])(int)={
8      servicio_sumar, servicio_restar,
9      servicio_multiplicar, servicio_dividir };
10   int puerto_inicial;
11   fd_set fd;
12
13   // inicializacion de puerto_inicial omitida
14   inicializar_sockets(sockets, NUM_SERVICIOS,
15                       puerto_inicial);
16   while(1) {// Bucle infinito de atención a clientes
17     int i;
18     rellenar_fd(&fd, sockets, NUM_SERVICIOS);
19     select(maximo(sockets, NUM_SERVICIOS) + 1,
20            &fd, NULL, NULL, NULL);
21     for (i=0;i<NUM_SERVICIOS;i++)
22       if (                                ) {
23                                           ;
24         // Invocar el servicio adecuado
25         servicios[i](sdat);
26       }
27   } // while
28 } // main
```

— ¿Qué falta en los tres huecos de la función `enviar_y_recibir` del cliente?

htons(a)
htons(b)
htons(resultt)

Explicación: Como se indica en el enunciado, cliente y servidor intercambian sus datos en formato *Big Endian*, por lo que deben

adaptar el valor de las variables de su formato interno al orden de red. Para ello deben usar la macro `htons` sobre cada variable que van a enviar (variables `a` y `b`), y la macro `ntohs` sobre cada variable cuyo valor haya sido leído de la red (variable `result` en este caso).

— ¿Qué falta en el hueco del listado de la función `servicio_sumar`?

```
htons( ntohs(x) + ntohs(y) )
```

Explicación: La operación a realizar es sencilla, basta sumar los datos recibidos. El problema es que los datos han sido recibido en formato *Big Endian* y para poder sumarlos deben ser convertidos al formato propio de la arquitectura (esto lo hace la macro `ntohs`), y por otro lado, el resultado de la operación se almacena en la variable `res` que, como podemos ver en la línea siguiente, es enviado a la red sin más transformaciones, por lo que este resultado debe estar ya en formato *Big Endian*. Así pues, el resultado de la suma debe pasarse a la macro `htons` antes de asignarse a `res`. A partir de estas consideraciones, la respuesta es `htons(ntohs(x)+ntohs(y))`.

— Completa la línea **??** del bucle principal del servidor, en la que se determina por qué socket pasivo se ha recibido una conexión.

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

Explicación: Los diferentes sockets están almacenados en el array `sockets` que se recorre con un bucle. Para cada iteración del bucle se comprueba si el elemento `sockets[i]` está en el conjunto `fd` con ayuda de la macro `FD_ISSET()`.

— ¿Qué falta en la línea **??** del servidor para que funcione correctamente?

```
sdat=accept(sockets[i], NULL, NULL)
```

Explicación: Una vez se ha encontrado que en el socket `sockets[i]` hay un cliente esperando, debe aceptarse dicho cliente para crear así el socket de datos que nos permitirá comunicarnos por él. Este socket de datos se asignará a la variable `sdat`, pues vemos más adelante que es ésta variable la que se le pasa al servicio. `accept()` requiere dos parámetros más, aparte del socket de escucha, a través de los cuales recibirá la dirección (IP y puerto) del cliente. En este caso esa información no es necesaria, puesto que no vemos que el programa la use. De hecho ni siquiera ha declarado una variable de tipo `struct sockaddr_in` en la cual recibir esta información. Por tanto podemos pasar `NULL` en estos dos parámetros.

❑ ¿Cuál o cuáles de las siguientes afirmaciones son ciertas?

A) La transparencia de migración significa que un recurso pueda cambiar su localización sin afectar a las aplicaciones que lo usaban.

B) Para aumentar la fiabilidad tenemos que aumentar la redundancia tratando de evitar los riesgos que se puedan producir por la aparición de la inconsistencia.

C) Los sistemas distribuidos son más fáciles de administrar y gestionar además de permitir una escalabilidad sin límites del sistema.

D) Un sistema centralizado se puede defender más eficientemente frente a los ataques de seguridad que uno distribuido.

```
A, B, D
```

Explicación: La opción C es falsa ya que un sistema distribuido es más complejo de administrar que un sistema centralizado.

❑ Se muestra en el listado siguiente una posible implementación de una función llamada `strlen_utf8` que recibe un puntero a una secuencia de bytes, terminada en 0, y devuelve cuántos caracteres tiene esa cadena si se interpreta como codificada en UTF-8 (en contraposición a lo que haría `strlen` que simplemente retornaría el número de bytes presentes hasta el terminador).

Para simplificar la implementación, se comprueba previamente si la cadena recibida es UTF-8 válido, con la ayuda de otra función de apoyo (`valido_utf8()`) que no se muestra. Si la cadena pasa este test, ya es seguro operar sobre ella asumiendo que todos los caracteres multibyte que contenga están correctamente codificados. En caso contrario retorna -1. Esta es la implementación en cuestión:

```
1  #define MASK0  0x80  // 10000000
2  #define MASK1  0xE0  // 11100000
3  #define MASK2  0xF0  // 11110000
4  #define MASK3  0xF8  // 11111000
5
6  #define VALOR0 [redacted]  //
7  #define VALOR1 [redacted]  //
8  #define VALOR2 [redacted]  //
9  #define VALOR3 [redacted]  //
10
11 int strlen_utf8(char *s)
12 {
13   int i=0, longitud=0, n=0;
14   if (!valida_utf8(s)) return -1;
15   while (s[i]!=0) {
16     longitud++;
17     if ((s[i] & MASK0) == VALOR0) n=1;
18     if ((s[i] & MASK1) == VALOR1) n=2;
19     if ((s[i] & MASK2) == VALOR2) n=3;
20     if ((s[i] & MASK3) == VALOR3) n=4;
21     i+=n;
22   }
23   return longitud;
24 }
```

— ¿Qué falta en los huecos en que se definen las constantes `VALOR0`, ..., `VALOR3`?

```
#define VALOR0 0x00  //  00000000
#define VALOR1 0xC0  //  11000000
#define VALOR2 0xE0  //  11100000
#define VALOR3 0xF0  //  11110000
```

Explicación: La cadena que recibe la función está codificada en UTF-8. Para determinar cuántas letras contiene no basta contar cuántos bytes contiene, ya que UTF-8 admite caracteres que ocupan más de 1 byte. El algoritmo es por tanto examinar cada uno de los bytes que hemos recibido en la cadena y comprobar el patrón de bits por el que comienza. Así, si el primer bit es un cero, se trata de

un carácter ASCII que sólo ocupa 1 byte. Si comienza por `110` es un carácter que ocupa 2 bytes, etc.
La función determina en qué caso está y almacena en la variable `n` cuántos bytes ocupa el caracter que comienza en la posición `i`. Después avanza `i` en esa cantidad, de modo que se "salta" de un golpe todos esos bytes, quedando así preparado para procesar el siguiente carácter.
Para determinar en cuál de los casos está, aplica una máscara binaria al carácter `s[i]`, con la que realiza la operación `AND` bit a bit (operador `&` del lenguaje C). Esta máscara contiene unos en los bits que se pretenden filtrar, y cero en todos los demás cuyo valor no resulta interesante. Así, por ejemplo `MASK0` tiene el patrón `10000000`, que deja pasar tan solo el primer bit, poniendo los otros siete a cero; `MASK1` tiene el patrón `11100000` que deja pasar los tres primeros bits, poniendo los otros 5 a cero, etc. Comparando el resultado de este filtro con unos valores preestablecidos se sabe si el carácter `s[i]` es el primero de una secuencia de 1, 2, 3, o 4 bytes.
Una vez comprendido el mecanismo, determinar los valores concretos con los que comparar el resultado de la máscara es sencillo. Al aplicar la máscara `MASK0`, que deja pasar tan solo el primer bit, si ese primer bit era `0` el resultado de la máscara será `00000000`, y estaremos en este caso ante un código ASCII que ocupa un solo bit. Si en cambio `s[1]` comenzaba por `110`, al aplicar `MASK0` ya sale distinto de cero. Este caso se detecta adecuadamente con `MASK1`, que deja pasar los tres primeros bits. Sólo si estos tres primeros bits eran `110`, el resultado será `11000000`, y estaremos entonces ante un caracter multi-byte de dos bytes, etcétera.
Por tanto, los valores con los que comparar han de ser `00000000` para `VALOR0`, `11000000` para `VALOR1`, `11100000` para `VALOR2` y `11110000` para `VALOR3`. Observar que `VALOR1`, `VALOR2` y `VALOR3` siguen un esquema reconocible, pero `VALOR0` se aparta de él. En concreto, `VALOR0` **no es** `10000000` como pueda pensare a primera vista si no se reflexiona demasiado. Ya que comparamos con el resultado de aplicar una máscara que sólo deja pasar el primer bit, si pusieramos `VALOR0` a `10000000`, estaríamos detectando el caso en que el primer bit fuese `1`, en lugar del caso en que sea `0`, como corresponde a la codificación UTF-8 de los caracteres ASCII.

Se implementa un programa para probar la función anterior, que la llama con diferentes cadenas para mostrar por pantalla el resultado que devuelven `strlen` y `strlen_utf8` sobre cada una. El programa se escribe y se ejecuta desde una terminal capaz de manejar UTF-8. Este es el código del programa de prueba:

```
1  int main()
2  {
3    char *cad[4]={ "Enero", "Eñe", "1€", "世界您好" };
4    int i;
5
6    for (i=0; i<4; i++)
7      printf("Cadena: |%s|, strlen=%d, strlen_utf8=%d\n",
8          cad[i], strlen(cad[i]), strlen_utf8(cad[i]));
9    return 0;
10 }
```

Produciendo la siguiente salida:

```
Cadena: |Enero|, strlen=[5], strlen_utf8=[5]
Cadena: |Eñe|, strlen=[4], strlen_utf8=[3]
Cadena: |1€|, strlen=[4], strlen_utf8=[3]
Cadena: |世界您好|, strlen=[12], strlen_utf8=4
```

— ¿Qué números aparecerían en los lugares que se han ocultado de la salida del programa? Para responder, te será útil saber que el símbolo del euro tiene código `U+20AC` y que los caracteres chinos están en el plano 0, por encima del valor `U+4000`.

5, 5
4, 3
4, 2
12, 4

Explicación: La función `strlen()` cuenta el número de bytes de una cadena, por tanto en este caso contará cuántos bytes tiene la codificación UTF-8 de cada cadena de ejemplo. La función `strlen_utf8()`, tal como ha sido programada, cuenta el número de caracteres Unicode que hay en cada una de las cadenas, la cual podemos averiguar simplemente contando el número de letras que tienen estas cadenas tal como se muestran en el listado.
Así, la primera cadena `"Enero"` tiene cinco letras, y ya que la codificación UTF-8 de cada una ocupa 1 byte (pues todas son ASCII), `strlen()` también contará 5 bytes. En este caso ambas funciones dan el mismo resultado.
La cadena `"Eñe"` tiene tres letras (este será el resultado de `strlen_utf8()`), pero una de ellas no es ASCII. Como sabemos, el carácter `'ñ'` forma parte del estándar ISO-8859-1 y por tanto su código Unicode es menor de `U+00FF`. Esto implica que so codificación UTF-8 ocupará 2 bytes. Por tanto tenemos 4 byes en la cadena, y este será el valor devuelto por `strlen()`.
La cadena `1€` tiene dos letras (`strlen_utf8()` devolverá 2), pero el código Unicode del carácter `'€'` es el `U+20AC`, y por tanto, de acuerdo con las reglas de codificación de UTF-8, ocupará tres bytes. Por tanto la cadena ocupa 4 bytes (`strlen()` devolverá 4).
Finalmente el texto en chino tiene cuatro caracteres (es difícil ver esto si no se sabe chino, pues los caracteres son de ancho doble y quizás no se vea claramente donde acaba uno y empieza el siguiente. De todas formas el enunciado ya muestra que `strlen_utf8()` devuelve 4 para este caso, por lo que no hay dudas de que la cadena está compuesta por cuatro códigos Unicode). Tal como se nos indica en el enunciado, estos caracteres tienen códigos por encima de `U+4000`, pero dentro del plano cero. Esto implica que cada uno de ellos requiere 3 bytes para su codificación en UTF-8, por lo que el número total de bytes es de 12, y éste será el valor devuelto por `strlen()`.

❑ A continuación se muestra una definición de tipos de datos XDR que utiliza un programa para codificar información. El program recibe a través de la red, usando el interfaz de sockets, un dato de tipo `MiUn`. Si el contenido recibido es una lista, el programa llama al procedimiento `intercambia()` para copiar los valores en una variable de tipo `Lista` que será enviada a continuación a través de un nuevo socket, codificada en XDR.

```
1  typedef int Numeros<41>;
2
3  struct Lista {
4   int dato;
5   Lista *otro;
6  };
7
8  union MiUn switch (int que) {
9   case 1:
10   Lista milis;
11  case 3:
12   Numeros num;
13  case 16:
14   int enteros[5];
15 };
```

```
1   [...]
2   /* Declaramos una variable de cada tipo */
3   Lista lst;
4   MiUn laun;
5   FILE *fich1, *fich2;
6   XDR  op1, op2;
7   int s1,s2;
8   struct sockaddr_in dir;
9
10  /* Inicialización de los sockets */
11  s1=socket(PF_INET, SOCK_STREAM,0);
12  s2=socket(PF_INET, SOCK_STREAM,0);
13  /* Se omite la inicialización de dir */
14  connect(s1, (struct sockaddr *)&dir, sizeof(dir));
15  /* Se omite la inicialización de dir */
16  connect(s2, (struct sockaddr *)&dir, sizeof(dir));
17
18 /* Prepara los sockets */
19  fich1=[redacted]
20  fich2=
21
```

```
22   /* Preparación de la operación XDR */
23   [                                    ]
24   [                                    ]
25
26   /* Llamada al filtro para leer los datos */
27   if (xdr_MiUn(&op1, &laun) !=TRUE) {
28     fprintf(stderr, "Error al recibir el dato\n");
29   }
30 /* Intercambio de listas */
31   if ([              ]) {
32     intercambia(&laun, &lst);
33 /* Llamada al filtro para enviar el dato */
34   if (xdr_Lista(&op2, &lst) !=TRUE) {
35     fprintf(stderr, "Error al enviar la lista\n");
36   }
37   printf("\nDatos enviados\n");
38 }
```

— ¿Cuáles será el tamaño mínimo, en bytes, que puede ocupar una variable de tipo `MiUn` codificada en XDR? Escribe el valor de dichos bytes.

Mínimo: 8 Bytes: 00 00 00 00 Ɛ0 00 00 00

Explicación: `MiUn` es una union discriminada y, como todas las uniones discriminadas su codificación en XDR consiste en la codificación del discriminante (4 bytes) más la codificación del tipo discriminado en la unión. Para que la unión ocupe el menor número de bytes posible, tendremos que seleccionar el elemento de la unión que se codifique con menos bytes.
El primer campo de la unión es una estructura de tipo `Lista` que se compone de dos elementos, un entero y un puntero a otra estructura `Lista`. La codificación mínima de este campo de la unión sería de 8 bytes: 4 para la codificación del entero y otros 4 para la codificación del puntero apuntando a `NULL`. Sumados estos 8 bytes a los 4 del discriminante nos da un valor mínimo de 12 bytes para esta selección.
El segundo campo de la unión es de tipo `Numeros`, que es una array de longitud variable de enteros. La codificación XDR de un array de longitud variable consta de la codificación del número de elementos del array seguida de la codificación de cada uno de los elementos. Si el array está vacio el tamaño mínimo de la codificación de cualquier array de longitud variable es de 4 bytes, ya que en esos bytes codificaríamos que el array tiene cero elementos. Por tanto, en este caso, el tamaño de la unión discriminada nos hace un total de 8 bytes.
El tercer campo de la unión discriminada es un array de longitud fija de enteros. En este caso, su codificación es siempre fija y consta de la codificación de los 5 elementos del array. Son 5 enteros a 4 bytes cada uno, más los 4 bytes del discriminante nos da un total de 24 bytes para la longitud total de la unión discriminada.
Analizando las posibilidades, el menor número de bytes para la unión discriminada viene dado cuando el discriminante selecciona el array de longitud variable y éste no tiene elementos. Por lo tanto se debe codificar un valor del discriminante de 3.

— Completa la inicialización de `fich1` y `fich2` en las líneas **??** y **??**.

```
                                    fich1=fdopen(sl,"r");
                                    fich2=fdopen(s2,"w");
```

Explicación: Para poder utilizar los filtros XDR con la API de sockets tenemos que asociar a cada uno de los sockets de nuestro programa un estructura de datos de tipo `FILE *`. Como los sockets funcionan exactamente igual que los descriptores de ficheros, utilizamos la función del sistema operativo `fdopen` que permite asociar a una estructura `FILE *` a un descriptor de fichero.

— ¿Qué falta en los huecos de las líneas **??** y **??**?

```
                                    xdrstdio_create(&op1, fich1, XDR_DECODE);
                                    xdrstdio_create(&op2, fich2, XDR_ENCODE);
```

Explicación: Una vez tenemos asociado con un socket una estructura de tipo `FILE *` podemos usar las funciones del API de XDR para utilizar los filtros XDR en nuestra aplicación. Los filtros XDR necesitan que se inicialice una estructura de tipo `XDR *` en la cual se indica dónde se dejarán o de dónde se tomaran los flujos XDR y si la conversión será hacía XDR *(encode) o desde XDR (decode). Como queremos que los flujos salgan y entren a través del interface de red usando la API de sockets, es necesario utilizar la función de inicialización que asocia la estructura* `XDR` *con una archivo del sistema:* `xdrstdio_create`.

— Completa el hueco de la línea **??**.

```
                                    if (laun.que == 1)
```

Explicación: Según el enunciado del problema, si el parámetro recibido es una unión discriminada en la que su contenido es una estructura de tipo `Lista`, se llama a la función `intercambia`. Para saber el tipo de dato que contiene la unión discriminada únicamente hay que mirar el valor del discriminante y, al ser el equivalente en C de una unión discriminada una estructura en la que uno de sus campos es el discriminante, sólo tenemos que comprobar el valor de ese campo de la estructura.

— Al recibir un dato del tipo `MiUn` el programa ha recibido la siguiente secuencia de bytes:
`00 00 00 03 00 00 00 03 00 00 00 01`
`00 00 00 02 00 00 00 03`
que asignó de manera adecuada a una variable de tipo `MiUn` llamada `drec`. En la tabla adjunta escribe, en la primera columna, cuáles serían los argumentos de una sentencia `printf()` que mostraran el contenido de los campos adecuados de la variable `drec` que se ha recibido; y en la segunda columna, el valor que se mostraría en la pantalla para cada uno de estos `printf`. *Nota: Sólo los argumentos de los* `printf()`*, ningún código adicional.*

Sentencia Printf	Valor
`"%d",drec.que`	3
`"%d",drec.MiUn_u.num.Numeros_len`	3
`"%d",drec.MiUn_u.num.Numeros_val[0]`	1
`"%d",drec.MiUn_u.num.Numeros_val[1]`	2
`"%d",drec.MiUn_u.num.Numeros_val[2]`	3

Explicación: En este ejercicio tenemos que asignar a los elementos de la union discriminada los bytes que se han recibido. Para ello vamos aplicando las reglas de codificación XDR de una unión discriminada:

- El primer elemento que se codifica es el valor del discriminante, que siempre será una valor de tipo entero. Los enteros ocupan 4 bytes, luego los primeros 4 bytes del código es el discriminante. En este caso, codifican el valor 3.
- En función del valor del discriminante, se codifica el valor de la unión. El valor 3 del discriminante selecciona el array de longitud variable de enteros como valor de la unión, por lo tanto ahora tenemos que aplicar las reglas de codificación de arrays de longitud variable para saber que es lo que hemos recibido.
  - Los arrays de longitud variable codifican primero, como un entero, el número de elementos que contiene el array. Los cuatro primeros bytes del array codifican un 3, luego ese será el número de elementos que tenemos.

- A continuación, tenemos los elementos del array. En este caso son enteros por lo que cada 4 bytes tenemos la codificación de uno de ellos. Vemos que los enteros tienen los valores 1, 2 y 3.

Nos queda por asignar estos valores a los campos de la variable `drec`. El discriminante se asigna al campo del mismo nombre que contiene la estructura equivalente a la unión discriminada (`drec.que`). A su vez, un array de longitud variable se representa en C como una estructura con dos campos: un entero de nombre el del array más el sufijo `_len` (`Numeros_len` en nuestro caso) que contiene el número de elementos del array; y un puntero al tipo base del array de nombre el mismo que el array más el sufijo `_val`(`Numeros_val` en nuestro caso). Esta estructura está dentro del segundo campo de la estructura en que se convierte la unión discçriminada, que es una unión C con el mismo nombre que la unión XDR añadiéndole el sufijo `_u` (`MiUn_u` en nuestro caso).

Anexo

SPEC CINT2006 Result

NEC Corporation

Express5800/120Lj
(Intel Xeon X5470)

SPECint2006 = 30.2

SPECint_base2006 = 26.3

CPU2006 license: 9006
Test sponsor: NEC Corporation
Tested by: NEC Corporation

Test date: Nov-2008
Hardware Availability: Oct-2008
Software Availability: Nov-2008

Results Table

Benchmark	Base						Peak					
	Seconds	Ratio	Seconds	Ratio	Seconds	Ratio	Seconds	Ratio	Seconds	Ratio	Seconds	Ratio
400.perlbench	438	22.3	**437**	**22.4**	436	22.4	**347**	**28.2**	347	28.1	346	28.2
401.bzip2	**526**	**18.4**	525	18.4	527	18.3	485	19.9	483	20.0	**484**	**19.9**
403.gcc	517	15.6	**517**	**15.6**	514	15.7	**372**	**21.7**	372	21.7	371	21.7
429.mcf	318	28.7	**317**	**28.8**	317	28.8	**316**	**28.9**	318	28.7	315	28.9
445.gobmk	**469**	**22.4**	469	22.4	469	22.4	**430**	**24.4**	431	24.3	430	24.4
456.hmmer	**496**	**18.8**	496	18.8	496	18.8	323	28.9	322	29.0	**323**	**28.9**
458.sjeng	551	22.0	**551**	**21.9**	552	21.9	**525**	**23.1**	524	23.1	525	23.0
462.libquantum	72.7	285	**72.5**	**286**	72.5	286	72.7	285	**72.5**	**286**	72.5	286
464.h264ref	707	31.3	715	31.0	**711**	**31.1**	568	39.0	567	39.0	**567**	**39.0**
471.omnetpp	**394**	**15.8**	393	15.9	395	15.8	359	17.4	358	17.5	**358**	**17.5**
473.astar	**430**	**16.3**	430	16.3	430	16.3	372	18.9	375	18.7	**375**	**18.7**
483.xalancbmk	248	27.8	247	28.0	**247**	**27.9**	248	27.8	247	28.0	**247**	**27.9**

Results appear in the order in which they were run. Bold underlined text indicates a median measurement.