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

❑ 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?

❑ 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?

❑ 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?

— ¿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?

— ¿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?

— ¿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?

❑ 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?

❑ 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?

❑ 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 ...

❑ 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 %

A

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?

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

— Completa la línea 22 del bucle principal del servidor, en la que se determina por qué socket pasivo se ha recibido una conexión.

— ¿Qué falta en la línea 23 del servidor para que funcione correctamente?

❑ ¿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.

❑ 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`?

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=[redacted], strlen_utf8=[redacted]
Cadena: |Eñe|, strlen=[redacted], strlen_utf8=[redacted]
Cadena: |1€|, strlen=[redacted], strlen_utf8=[redacted]
Cadena: |世界您好|, strlen=[redacted], 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`.

❑ 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   [redacted]
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 ([redacted]) {
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: Bytes:

— Completa la inicialización de `fich1` y `fich2` en las líneas 19 y 20.

A

— ¿Qué falta en los huecos de las líneas 23 y 24?

— Completa el hueco de la línea 31.

— 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

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.