ET1032 Informática Industrial

11 de junio de 2014

Soluciones al examen.

NOTA: El nivel de detalle con que se responde a las preguntas en este documento no es el que se requiere ni se espera para la calificación de los ejercicios. Al redactar este documento se pretende dar información más amplia y completa con fines documentales y didácticos.

Pregunta 1. (**1 punto**) - Indicad cómo afecta a la latencia de las operaciones de entrada/salida, en particular a su duración y variabilidad, el que éstas se gestionen mediante prueba de estado o mediante interrupciones. Justificad adecuadamente la respuesta.

La latencia de una operación se define como el tiempo que transcurre desde que ésta se inicia hasta que el primer dato comienza a recibirse en el destino. Sin pretender seguir una definición estricta, podemos considerar la latencia de una operación de entrada/salida como el tiempo transcurrido desde que el dispositivo informa –mediante algún bit o conjunto de bits de estado- que tiene datos disponibles –o requiere alguna otra atención- hasta que la Unidad Central de Proceso comienza a leer tales datos o a atender la solicitud del dispositivo.

Como sabemos, la UCP puede advertir tal señalización bien leyendo esos bits explícitamente mediante instrucciones de acceso a la E/S, lo que sería prueba de estado –encuesta o *polling*- bien mediante una interrupción, que es un mecanismo *hardware* desencadenado automáticamente cuando el dispositivo activa dichos bits de estado –suponiendo, claro está, que esa interrupción esté activada.

Con todo esto, podemos ver que la latencia, en la gestión de interrupciones –al menos la latencia desde que se genera el aviso hasta que se ejecuta la rutina de tratamiento de interrupción- es generalmente corta, pues depende del *hardware*, y de duración poco variable –con diferencia de pocos ciclos de reloj entre la menor y la mayor duración. Por otra parte, si la gestión se realiza por prueba de estado, se pueden dar dos situaciones. Si nuestra UCP no hace otra cosa que consultar el estado del dispositivo, la latencia es la menor posible y, de nuevo, su variabilidad prácticamente nula. No es este, sin embargo, el caso más habitual. Normalmente la UCP está dedicada a otras tareas y consulta cada cierto tiempo –cierto número de instrucciones- el estado del dispositivo por si tiene que atenderlo. Con estas circunstancias, la variabilidad de la latencia es máxima, yendo desde prácticamente 0 si el dispositivo consulta el dispositivo justo cuando éste ha señalizado un evento, hasta el peor caso, en que el dispositivo activa los bits justo cuando se ha terminado su consulta por parte de la UCP. Este tiempo depende lógicamente de la frecuencia con que se consulte el estado pero, en promedio, es en general mayor que si se utilizan interrupciones.

Pregunta 2. (1 punto) - Describid las ventajas e inconvenientes de los buses serie punto a punto con señalización diferencial. Justificad por qué este tipo de buses son los más utilizados actualmente.

Hoy en día, los buses con mayor funcionalidad y productividad, que además son los últimos estándares que han ido apareciendo, utilizan como se indica en el enunciado señalización diferencial, transmisión serie de datos y topología punto a punto en su nivel físico. Ejemplos muy conocidos de esto serían USB y PCI Express.

Veamos en primer lugar las ventajas de este tipo de buses:

- **Velocidad**. La transmisión de datos en una única línea lógica de comunicaciones –un par diferencial- es óptima según las restricciones del medio de transporte. Dado que solo hay dos extremos en la línea, las condiciones eléctricas del medio no se han de calcular para el peor caso –por ejemplo, las tarjetas que se insertan en una conexión multipunto añaden capacidades al medio, por lo que los parámetros de la transmisión se han de calcular para el máximo número de tarjetas añadidas. Así pues, la velocidad de transmisión será mayor que en otra topología, a igualdad del resto de parámetros. Esto conlleva en la práctica a las mayores velocidades de transmisión de datos por línea lógica.
- **Inmunidad al ruido**. La señalización diferencial es mucho más inmune al ruido inducido por radiación electromagnética, ya que éste afecta en modo común a ambas líneas y es, por tanto, eliminado al evaluar la diferencia entre las magnitudes eléctricas de cada una de ellas.
- **Coste**. El hecho de que toda la información se transmita en modo serie, en general bit a bit, a través de un canal lógico de comunicaciones hace que el número de conexiones eléctricas del bus sea lo más reducido posible y, por lo tanto, tenga menor coste. De este modo además no se requieren señales adicionales de sincronización que encarecen y complican todavía más los buses paralelos. Esto, que evidentemente va en detrimento de la velocidad de transmisión de información, se puede paliar en cierta medida utilizando técnicas –que por supuesto aumentan el coste- como constituir una conexión lógica bidireccional mediante dos conexiones unidireccionales, para tener un bus *full-duplex*. Esta técnica, que utilizan USB 3.0 –los estándares anteriores eran *half-duplex*- y PCI Express, acaba requiriendo cuatro conexiones físicas, sin contar masas, para una conexión lógica. Otra solución, la empleada por PCI Express además de la comentada, para aumentar el ancho de banda es añadir conexiones lógicas para enviar varias palabras –bytes, en este caso- en paralelo por las diferentes líneas lógicas, pero manteniendo el envío serie de los bits dentro de una palabra y reduciendo el sobrecoste temporal de la sincronización.

La principal desventaja de este tipo de buses es la conectividad limitada. Siendo buses punto a punto, cada segmento de bus a nivel físico solo puede comunicar dos dispositivos. De esta manera, si se quiere construir un bus con mayor conectividad es necesario que existan dispositivos activos de comunicaciones para redirigir los mensajes entre los diferentes nodos conectados al bus. El avance y abaratamiento de la tecnología, el hecho de que los circuitos conmuten y traten las señales a altas velocidades, hace que esta opción, la de buses punto a punto con elementos de bus activo, sea factible y, por esto y lo comentado más arriba, la más utilizada en la actualidad.

Pregunta 3. **(1,5 puntos)** - El sistema operativo puede verse como una máquina virtual que realiza operaciones complejas cuando las aplicaciones de usuario así se lo solicitan. Indicad mediante qué mecanismo las aplicaciones pueden pedir estas operaciones al sistema operativo. Las instrucciones que lo implementan, ¿se ejecutan a nivel de usuario o de supervisor? Dad algún ejemplo de una función del lenguaje C que utilice este mecanismo al ejecutarse.

Como se ha visto, una de las visiones del sistema operativo es la de constituir una máquina extendida o virtual que se ejecuta sobre las componentes físicas del ordenador y ofrece a las aplicaciones de usuario un conjunto de operaciones de alto nivel para interactuar con los recursos del sistema mediante una interfaz común y, en la actualidad, estandarizada. De esta forma, para interactuar con muchos de los dispositivos del ordenador no es necesario conocer sus aspectos de bajo nivel, como direcciones que utilizan, tamaño de sus almacenamientos intermedios, etcétera, sino que basta con solicitar al sistema operativo que realice esas operaciones estandarizadas sobre ellos. Así pues es el sistema operativo quien carga con toda la responsabilidad de actuar adecuadamente con los recursos, ofreciendo al programador operaciones de elevada potencia y portabilidad.

El mecanismo mediante el cual las aplicaciones acceden a estos servicios del sistema operativo se conoce como llamadas al sistema. Estas llamadas acaban implementándose mediante instrucciones específicas de la arquitectura que fuerzan el cambio del modo de ejecución desde el nivel de usuario (aplicación) hasta el de supervisor (sistema operativo). Sin embargo estas instrucciones se encuentran incluidas como parte de ciertas funciones de la biblioteca estándar de cualquier lenguaje de programación, de tal modo que una de estas funciones comienza ejecutándose a nivel de usuario, realizando ciertas verificaciones y adaptaciones de datos, y tras ejecutar una de estas instrucciones especiales fuerza el cambio de modo del procesador al nivel de supervisor. Este código satisface la petición del usuario de la forma que corresponda y, una vez terminada su función, devuelve el procesador al modo de ejecución de usuario y el programa a la instrucción siguiente a la que efectuó el cambio. Posiblemente de forma discontinua en el tiempo, de acuerdo con el diseño del sistema operativo y sus mecanismos de planificación.

Casi todas las funciones que interactúan con la entrada/salida incorporan llamadas al sistema, pues en la mayor parte de sistemas operativos estas operaciones siempre son responsabilidad suya. Así, de la biblioteca estándar de C podríamos citar las funciones de salida **`printf`**, **`puts`**; las de entrada **`fgets`**, **`scanf`** y por supuesto las de gestión de ficheros **`open`**, **`close`**, **`read`**, **`write`**, etcétera.

Pregunta 4. **(1,5 puntos)** - La ejecución del fragmento de código que aparece a continuación

```
#include <stdio.h>

typedef struct _trans {
   char ref[20];
   double hfe, ic, ib, vce, vsat;
   int tipo, frec;
} transistor;

void main()
{
   int i;
   double *pdouble;
   transistor *pt, vt[100];

   printf("El vector vt esta en la direccion %08X\n", vt);
   printf("El campo vce del elemento vt[0] en %08X\n", &vt[0].vce);
   printf("La estructura transistor ocupa %d bytes\n", sizeof(transistor));
   pt = vt;
   for (i = 0; i < 4; i++) pt++;
   pdouble = &pt -> vce;
   printf("\npt vale %08X y pdouble %08X\n", pt, pdouble);
}
```

muestra por pantalla las siguientes líneas de texto, donde los valores **pt** y **pdouble** se han ocultado.

```
El vector vt esta en la direccion 0028E2F0
El campo vce del elemento vt[0] en 0028E320
La estructura transistor ocupa 72 bytes

pt vale ???????? y pdouble ????????
```

Indicad los valores que deberían aparecer para **pt** y **pdouble**, justificando la respuesta adecuadamente.

En principio, basta con no ocultar los valores devueltos por el programa para tener la solución, obtenida de la forma más objetiva y real posible:

```
pt vale 0028E410 y pdouble 0028E440
```

Veamos ahora el porqué de estos valores. El bucle **for** del código incrementa cuatro veces el puntero **pt**. Como sabemos, el compilador conoce el tamaño del tipo de datos apuntados por este puntero, **transistor**, por lo que cada incremento en realidad suma el tamaño de la estructura **transistor**. Como conocemos la posición del primer elemento, que es la inicial de **pt**, es decir **vt**, y el tamaño de la estructura, 72, tenemos que tras el bucle

pt = vt + 4 * 72

es decir

pt = 0x0028E2F0 + 288 = 0x0028E2F0 + 0x120 = 0x0028E410

Para calcular el valor de **pdouble** nos hemos de dar cuenta que apunta al campo **vce** del elemento apuntado por **pt**. Como sabemos la dirección de tal campo en **vt[0]** y la dirección de **vt[0]** –que es **vt**- conocemos el desplazamiento desde el inicio del campo hasta dicha componente, de forma que

pdouble = pt + (&vt[0].vce – vt) = 0x0028E410 + (0x0028E320 – 0x0028E2F0)

pdouble = 0x0028E410 + 0x30 = 0x0028E440

Pregunta 5. **(2,5 puntos)** - Realizad en lenguaje de programación C la función

```
int ponCeros(int dato, unsigned char prim, unsigned char num)
```

Dicha función devuelve el valor **dato** modificado de forma que pone a cero **num** bits a partir de la posición **prim**. Se considera la numeración habitual de bits, siendo 0 el menos significativo -más a la derecha- y, para un entero de 32 bits, 31 el más significativo -más a la izquierda. La función debe funcionar adecuadamente sea cual sea el tamaño de los enteros de la arquitectura. Si el primer bit excede el tamaño, **dato** se devolverá sin modificar; si los bits a cambiar superan el tamaño del entero, los de exceso serán ignorados, poniendo a 0 hasta el último.

A continuación se ponen algunos ejemplos de ejecución de la función pedida, con enteros de 32 bits.

```
ponCeros(0xFFFFFFFF, 16, 4)  devuelve 0xFFF0FFFF
ponCeros(0xFFFFFFFF, 43, 6)  devuelve 0xFFFFFFFF
ponCeros(0xFFFFFFFF, 24, 50) devuelve 0x00FFFFFF
ponCeros(0xFFFFFFFF,  7, 11) devuelve 0xFFFC007F
```

Dado que la función pedida debe poner a 0 un conjunto de bits del valor de entrada, la forma de lograrlo es creando una máscara de tantos bits como el entero, que tenga unos desde el bit 0 hasta el indicado por **prim**, luego tantos ceros como indique **num** y de nuevo unos hasta el bit más significativo. El resultado pedido será entonces el resultado de realizar una operación lógica **and** entre **dato** y dicha máscara. Esto se puede realizar de muchas maneras. A continuación se presentan dos. La primera construye la máscara añadiendo unos de la forma indicada, mediante dos bucles. La segunda, más eficaz pero menos evidente, construye la máscara directamente mediante operaciones binarias. Ambas funciones verifican en primer lugar la coherencia de los parámetros **prim** y **num**. El código aparece comentado y explicado.

```
int poncerosV1(int dato, unsigned char prim, unsigned char num)
{
   int masc, i;

   if ((prim >= 8*sizeof(int)) || !num) // Si num es 0 o mayor que el numero
        return dato;                    // de bits devolvemos dato sin tocar
   if (num + prim > 8*sizeof(int))      // Si num + prim van mas alla del ultimo
        num = 8*sizeof(int) - prim;     // bit ajustamos num
   masc = 0;                            // Creamos la mascara y ponemos unos hasta
   for(i = 0; i < prim; i++)            // prim
        masc = masc | (1 << i);         // Luego los ponemos desde num + prim
   for (i = num + prim; i < 8*sizeof(int); i++)
        masc = masc | (1 << i);         // hasta el final
   return dato & masc;                  // Devolvemos dato and mascara
}

int poncerosV2(int dato, unsigned char prim, unsigned char num)
{
   int masc;

   if ((prim >= 8*sizeof(int)) || !num) // Si num es 0 o mayor que el numero
        return dato;                    // de bits devolvemos dato sin tocar
   if (num + prim > 8*sizeof(int))      // Si num + prim van mas alla del ultimo
        num = 8*sizeof(int) - prim;     // bit ajustamos num
   masc = (1 << prim) - 1;              // Creamos los bits bajos a uno (2^prim - 1)
   masc = masc | ((~masc) << num);      // Al negar y desplazar ponemos num ceros
                                        // hasta prim + num, y el resto unos.
                                        // Al hacer el or con la mascara original
                                        // restablecemos los unos hasta prim
   return dato & masc;                  // Devolvemos dato and mascara
}
```

Pregunta 6. **(2,5 puntos)** - Se desea conectar un microcontrolador a ciertas entradas y salidas digitales. La tabla inferior indica las características eléctricas de todos los pines de entrada/salida del microcontrolador, que se alimenta a 5V.

V_{IHmin}	3,8V	V_{OHmin}	4,2V	I_{Imax}	12 µA
V_{ILmax}	1,2V	V_{OLmax}	0,8V	I_{Omax}	23 mA

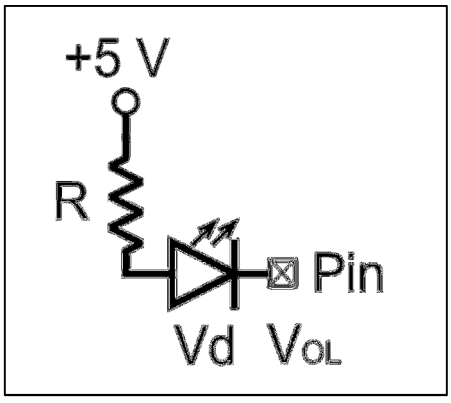
Para las interconexiones, que se proponen más abajo, se dispone de resistencias de ¼ W de cualquier valor, diodos de silicio estándar (Vd = 0,7V), diodos leds (Vd = 1,2V) y transistores de dos tipos. Los primeros, T1, tienen I_{Cmax} = 100 mA y una ganancia superior a 300; los segundos, T2, tienen I_{Cmax} de 2 A y una ganancia superior a 50. De ambos tipos, cuya V_{CEsat} es de 0,2V y V_{BE} es de 0,6V, se tienen versiones PNP y NPN con las características especificadas.

Se pide realizar los siguientes diseños y cálculos, sabiendo que se dispone, además de los 5V para alimentar el microcontrolador, de otra fuente de tensión de 24V.

- Conectar a una salida un diodo led que se encienda a nivel bajo, con una corriente mínima de 15 mA. Indicad cómo se conectaría y dad el valor máximo de la corriente que soportaría el led según dicha conexión.
- Añadir la circuitería necesaria para activar desde una salida del microcontrolador un relé que conmuta con una tensión mínima de 18V -la máxima es superior a los 24V de que se dispone- y consume como máximo 1,5 A.
- Conectar a una entrada del microcontrolador una señal digital que, en su estado alto, garantiza una tensión entre 20V y 24V y es capaz de suministrar una corriente de 300 mA. En su estado bajo la tensión es prácticamente 0V.

Diseñad los circuitos pedidos teniendo en cuenta los valores de corriente, tensión y potencia adecuados según lo que se especifica de cada uno de los componentes o señales.

Veamos en primer lugar la conexión del diodo. Como se ha de encender a nivel bajo, conectaremos su ánodo al pin de activación y su cátodo a +5V a través de la resistencia que hay que diseñar, como aparece en la figura. Con esta configuración cuando el led se encienda tendremos la tensión de salida del nivel bajo, V_{OL}, en el pin. Siendo el máximo valor V_{OLmax} de 0,8V, y queriendo que como mínimo circulen 15 mA a través del diodo, se tiene la ecuación que nos da el valor de R.

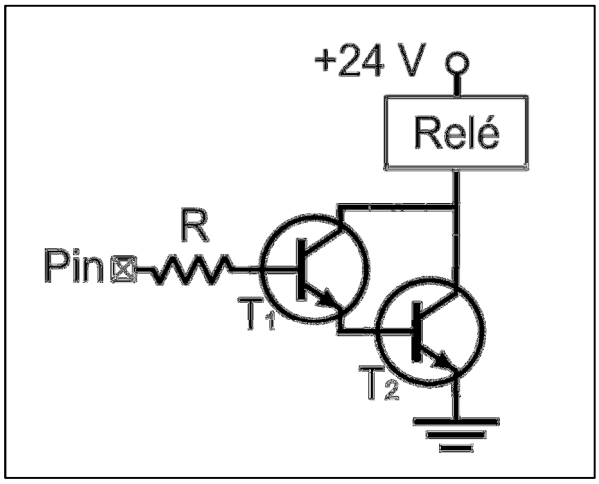


$$R = \frac{5 - (V_d + V_{OLmax})}{15\ mA} = \frac{3\ V}{15\ mA} = 200\ \Omega$$

Con este valor para R la corriente máxima se calcula sustituyendo en la ecuación anterior el valor de V_{OLmax} por el menor posible para V_{OL}, es decir 0V. Entonces esta intensidad será

$$I = \frac{5 - (V_d + 0)}{200\ \Omega} = \frac{3{,}8\ V}{200\ \Omega} = 19\ mA$$

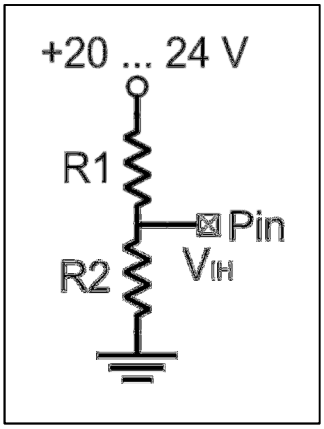
La conexión del relé resulta algo más complicada. Su consumo puede ser de hasta 1,5 A, por lo que será necesario utilizar un transistor para activarlo, en particular el modelo T2 de los disponibles, puesto que T1 solo soporta 100 mA. Sin embargo, para saturar T2 será necesaria una corriente superior a los 30 mA debido a que su ganancia mínima garantizada es de 50. Dado que la máxima corriente suministrada por una salida digital, I_{Omax}, es de 23 mA, el microcontrolador no podrá saturar directamente este transistor T2. Así pues, para poder saturarlo habrá que utilizar otro, de tipo T1, que suministre la corriente necesaria. Se puede realizar el montaje de diversas formas, en este caso utilizaremos una configuración darlington como la de la figura. A efectos prácticos, podemos suponer el conjunto T1-T2 como un transistor con una ganancia equivalente al producto de ambas, es decir, superior a 15000. De este modo, la corriente de base de T1 mínima para saturar el par será de 1,5 A / 15000, es decir 0,1 mA. Escogeremos una corriente diez veces superior para garantizar la saturación, y diseñaremos R para que en el peor caso circule 1 mA. De esta forma tenemos



$$R = \frac{V_{OHmin} - 2 * V_{BE}}{1\ mA} = \frac{3\ V}{1\ mA} = 3\ K\Omega$$

Como se sabe, en un par darlington el transistor de potencia, T2 en nuestro caso, no se satura dado que V_{CE} se ve limitada por la suma de V_{BE} y V_{CE} del otro, T1 en este caso. Como esta caída es menor de 1 V y el relé funciona a partir de 18 V, podemos garantizar que este montaje hará conmutar el relé al escribir un 1 lógico en el pin de activación.

Por último vamos a adaptar la tensión de entrada de entre 20 V y 24 V a los niveles lógicos de nuestro microcontrolador. El montaje será un simple divisor como el de la figura, al que se podrían añadir diodos de protección, aunque no se solicita en el enunciado nada al respecto.



La tensión en el pin será $V_{IH} = \frac{V * R_2}{R_1 + R_2}$, con V entre 20 y 24 V. Como a menor valor de V menor será el de V_{IH}, calcularemos las resistencias para que con 20 V de entrada obtengamos una tensión V_{IHmin}, 3,8 V. Con esto nuestra ecuación se transforma en

$$3{,}8 = \frac{20 * R_2}{R_1 + R_2} \Rightarrow R_1 = 4{,}2\ R_2$$

Hemos aproximado el resultado por defecto para que R2 sea en todo caso mayor y la tensión V_{IH} no sea nunca inferior a V_{IHmin}. Sabemos además que la salida digital puede darnos hasta 300 mA, lo que unido a la corriente de entrada del pin I_I, prácticamente despreciable, nos permite fijar la corriente que atravesará el divisor con un margen muy amplio. Elegimos 0,1 mA para reducir el consumo del sistema, con lo que tenemos

$$R_1 + R_2 = \frac{20\ V}{0{,}1\ mA} = 200\ K\Omega$$

Con esto obtenemos un valor de 39 K para R2 y 161 K para R1, ajustando R2 por exceso. Con estos valores recalculamos la tensión del pin en los extremos del rango de entradas, obteniendo 3,9 V cuando la entrada es de 20 V y 4,68 V cuando es de 24. Como vemos, ambos valores están entre V_{IHmin} y V_{CC} por lo que el diseño de las resistencias es correcto.