EXAMEN DE SISTEMAS INFORMÁTICOS DE TIEMPO REAL

Julio 1999

1. Definir de forma breve qué es un Sistema Informático de Tiempo Real. Poner un ejemplo. Explicar las diferencias entre los Sistemas de Tiempo Real controlados por eventos o por tiempo.

Un sistema de informático de tiempo real es un sistema basado en un computador que ***debe responder ante estímulos generados por el entorno dentro de un periodo de tiempo finito***. Es decir, un sistema de tiempo real intereactúa con el entorno (mundo físico real) adquiriendo estímulos y estados del mismo, generando una acción sobre dicho entorno. Por otro lado el **tiempo de respuesta** es un aspecto clave, es decir, no importa solo que sea capaz de generar un resultado correcto sino que éste debe producirse en un tiempo determinado.

El control de procesos continuos es un ejemplo de aplicación de los sistemas de tiempo real. El computador debe mantener las variables de salida del sistema siguiendo una cierta función de referencia. Cuando detecta que las variables se alejan de dicha referencia el computador debe responder variando las entradas del sistema de acuerdo a una acción de control. Esta respuesta debe ejecutarse dentro de un periodo de tiempo fijado por la dinámica del sistema y del regulador empleado.

Los STR controlados ***por eventos o interrupciones*** establecen la ejecución de un componente o tarea basándose en la aparición de una interrupción o señal generada por un evento externo. Constituyen un mecanismo eficaz para responder ante eventos externos no regulares.

Los STR controlados ***por tiempo*** operan de acuerdo a los ciclos del reloj o relojes del sistema. Este tipo de arquitecturas operan tratando los pulsos regulares del reloj como si fueran señales de interrupción. Cuando el reloj alcanza ciertos valores predefinidos, una acción apropiada es seleccionada para su ejecución. Este tipo de sistemas se utiliza cuando es preciso la ejecución de tareas periódicas o la ejecución de tareas mediante temporizadores.

2. Explicar las diferencias entre las representaciones de tiempo denso y tiempo disperso. Campos de aplicaciones de tiempo real más apropiadas para cada tipo de representación.

La representación **densa del tiempo** se modela como un conjunto *denso:* es un conjunto ordenado (relación '<') tal que entre dos elementos cualesquiera existe siempre un tercer elemento. Ejemplos de representaciones densas son los número racionales ***Q*** o los números reales ***R***.

El uso de una métrica densa para la representación del tiempo se justifica en situaciones donde los tiempos de ocurrencia de cualquier pareja de eventos pueden ser arbitrariamente cercanos. Algunas situaciones típicas en las aplicaciones de tiempo real donde la ***métrica temporal densa*** puede ser apropiada son:

1. Sistemas distribuidos con relojes locales para cada nodo, donde es prácticamente imposible conseguir una sincronización perfecta entre los relojes.
2. Sistemas asíncronos donde, por definición, los eventos pueden ocurrir en tiempos arbitrarios.
3. Sistemas que exhiban un comportamiento analógico que tratan con dominios continuos de valores.

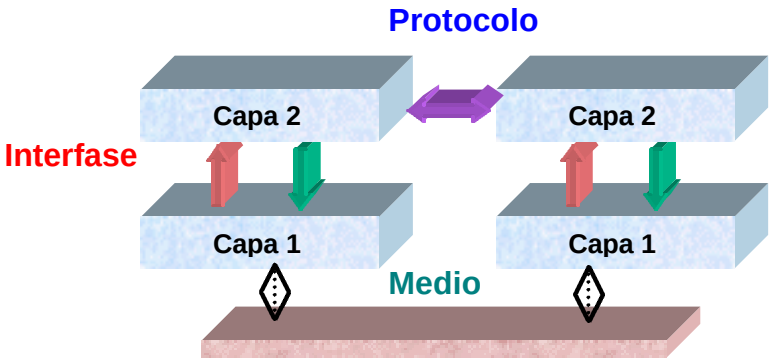
La representación dispersa del tiempo no cumple la condición anterior de forma que entre dos valores de tiempo consecutivos puede no haber ningún otro. Los números naturales ***N*** y los números enteros ***Z*** son ejemplos de conjuntos dispersos. Existen situaciones es las que es más adecuada una ***representación dispersa del tiempo***:

1. Sistemas monoprocesadores, y sistemas centralizados que utilizan un único reloj o un reloj maestro.
2. Sistemas síncronos, en los cuales los eventos ocurren en intervalos preestablecidos.
3. Sistemas que exhiben un comportamiento totalmente discreto.

3. Explicar los conceptos de protocolo e interfase en la arquitectura OSI. Enumerar y describir brevemente la funcionalidad de las capas del modelo OSI.

Protocolo es el conjunto de reglas que regulan la comunicación entre dos entidades del mismo nivel en dos equipos diferentes. Establece una comunicación lógica a nivel horizontal.

La **interfase** es el conjunto de servicios que las entidades de un nivel ofrecen a las entidades del nivel superior dentro del mismo equipo. Establecen una comunicación física a nivel vertical.



Las capas del modelo de referencia **OSI** son las siguientes:

- **Capa Física**:

Se encarga de transmitir una ristra de bits a través de un canal de comunicación. Especifica aspectos tales como: conexión mecánica, interfase eléctrica. Topología, modulación, velocidad de transmisión, sincronización, fragmentación, ...

- **Capa de Enlace**:

Dada una ristra de bits que le proporciona el nivel físico, lo convierte en una línea de comunicación que parezca libre de errores. Se encarga de la fragmentación, control de errores, control de flujo,...

- **Capa de Red**:

Controla la operatividad de la subred estableciendo las rutas desde un nodo fuente a su destino. Establece las rutas óptimas, controla la congestión de la red, realiza el control de flujo, la fragmentación y reagrupamiento, la traducción de direcciones y la interconexión entre redes hetereogéneas.

- **Capa de Transporte**:

Proporciona un canal de comunicación *extremo a extremo* (nodo origen a nodo destino) libre de errores. Se encarga de identificar al proceso destino de la información, la secuenciación. y el control de flujo.

- **Capa Sesión**:

Permite establecer sesiones de comunicación entre usuarios. Administra un testigo, evitando que se ejecuten acciones de comunicación simultáneas y establece puntos de chequeo en la información.

- **Capa Presentación**:

Resuelve le problema de la representación de los datos, la compresión y el cifrado de la información.

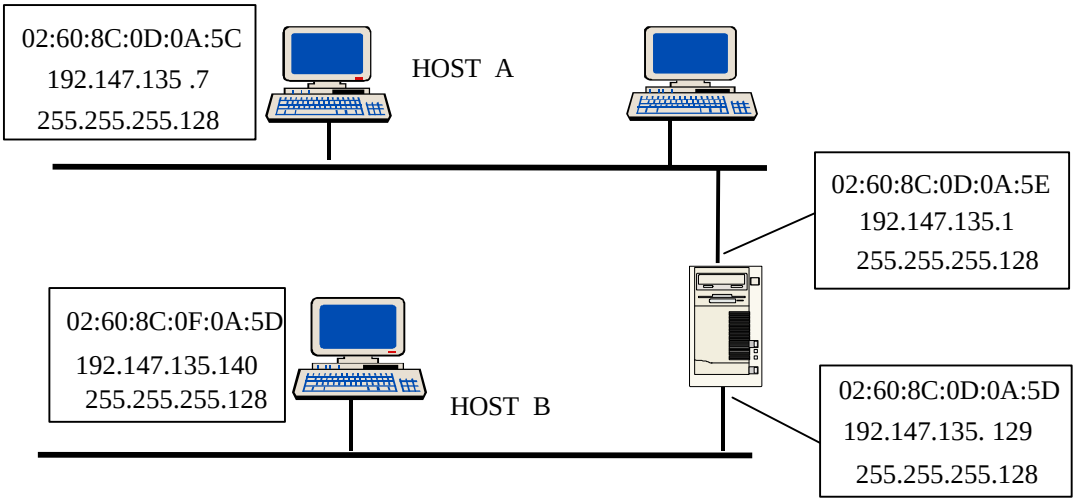
- **Capa aplicación**:

Conjunto de protocolos que interactúan con las aplicaciones o el usuario final.

4. Describir de forma breve las características fundamentales del protocolo TCP.

- Se trata de un protocolo orientado a la conexión, precisando cubrir tres etapas en toda transmisión: establecimiento de la conexión, transmisión y desconexión. Es por tanto un protocolo lento.
- Realiza un control de errores reenviando aquellos segmentos incorrectos o perdidos.
- Realiza también un control de flujo adecuando un receptor lento a un emisor rápido.
- Garantiza el secuenciamiento de la información
- Es un protocolo de tipo flujo de datos, no existen fronteras entre los mensajes de una conexión.

5. Describir la secuencia de mensajes y protocolos involucrados en una transmisión UDP entre el nodo A y el nodo B descritos en la figura. Para cada nodo se indican las direcciones MAC, IP y máscara de red.



- El datagrama **UDP** genera un único datagrama **IP**.
- No se especifica el tamaño del datagrama por lo que este puede generar uno o más fragmentos.
- El protocolo IP comprueba las direcciones IP y máscaras de los nodos origen y destino determinando que se encuentra en una subred diferente, por tanto **envía el datagrama al router** (192.147.135.1).
- Para enviar el datagrama al router se evalúa la tabla **ARP** generando, si fuera necesario, un mensaje **ARP request** que sería respondido con un **ARP reply** por parte del router indicando su dirección MAC (02:60:8C:0D:0A:5E).
- Conocida la dirección MAC se encapsula el datgrama en una trama **Ethernet** y se transmite al router.
- El router recibe la trama y la pasa al nivel **IP**. Comprueba en la tabla de enrutamiento la dirección IP destino enviando el datagrama a la subred 128 (192.147.135. 129).
- El destino final del datgrama se encuentra ya en la subred actual, por lo que se evalúa la dirección MAC mediante **ARP**. Se genera un mensaje **ARP request** que sería respondido con un **ARP reply** por parte de la máquina destino (02:60:8C:0F:0A:5D)
- Finalmente se enviaría el datagrama **IP** al nodo destino (192.147.135.140) encapsulado en una trama **Ethernet**.
- Si existiera fragmentación se recompondrían los fragmentos formando un único datagrama **UDP**

6. **Explicar en que consiste el paralelismo y el pseudoparalelismo. Comentar cual es el algoritmo de planificación más adecuado para implementar el pseudoparalelismo.**

(1 punto)

- Consultar el capítulo 2 de los apuntes, epígrafe 2.1 (Procesos) y el capítulo 4, epígrafe 4.3.5 (Planificación por turno rotatorio)

7. Explica detalladamente los **modelos de ejecución de threads**. Ventajas e inconvenientes de cada uno.

(1.5 puntos)

- Consultar el capítulo 2 de los apuntes, epígrafe 2.2.2 (Modelos de control de threads)

8. **Semáforos**: definición y funcionamiento (ejemplo)

(1 punto)

- Consultar el capítulo 6 de los apuntes, epígrafe 6.3 (Semáforos)

9. Realizar un programa con **pthreads** que escriba los cuadrados de los 100 primeros números naturales. El programa debe contener **dos threads**. El primero calculará y escribirá en pantalla los cuadrados de los enteros impares y el segundo el de los pares. La salida por pantalla debe ser ordenada (impar, par, impar, par,....).

(1.5 puntos)

```
#include <stdio.h>
#include <stdlib.h>
#include <pthread.h>
#include <sched.h>

/* Prototipos de las funciones que ejecutan los threads */
void *func1 (void *);
void *func2 (void *);

/* Declaraci¢n de los threads */
pthread_t thread1, thread2, thmain;
pthread_attr_t attr;    /*atributos de los threads*/
static pthread_mutex_t exmut; /*declaración del mutex*/

/* Definici¢n de las funciones func1 y func2 */
void *func1 (void *arg)
{
      int i;
      for (i=1; i<100; i = i+2)
            {
                  pthread_mutex_lock(&exmut); /*Operación P*/
                  printf("Th.1: %d\t%d\n", i, i*i);
                  pthread_mutex_unlock(&exmut); /*Operación V*/
                  sched_yield();    /*Fuerza la salida del procesador*/


            }
```

```
      pthread_exit(NULL);     /* Provoca la terminación del thread*/
}

void *func2 (void *arg)
{
      int i;
      for (i=2; i<=100; i = i+2)
            {
                  pthread_mutex_lock(&exmut);   /*Operación P*/
                  printf("Th.2: %d\t%d\n", i, i*i);
                  pthread_mutex_unlock(&exmut); /*Operación V*/
                  sched_yield();    /*Fuerza la salida del procesador*/

            }
      pthread_exit(NULL);     /* Provoca la terminación del thread*/

}

/*Funci¢n main*/
int main(void)
{


      pthread_mutex_init(&exmut,NULL); /Inicialización del mutex*/

      pthread_attr_init (&attr);    /*Inicialización de atributos*/

      printf("Soy la función main y voy a lanzar los dos threads \n");
      pthread_create (&thread1, &attr, func1, NULL);
      pthread_create (&thread2, &attr, func2, NULL);
      printf("Soy main: he lanzado los dos threads y termino\n");
      pthread_exit(NULL);
}
```