EXAMEN DE SISTEMAS INFORMÁTICOS DE TIEMPO REAL

Febrero 2001

SOLUCIÓN

1. **Explica detalladamente el proceso de creación y terminación de procesos en Unix. Poner un ejemplo.**

(1.5 puntos)

La respuesta corresponde al apartado 3.1.3 y los ejemplos 3.1 y 3.2 del libro de la asignatura

2. **Explica detalladamente qué mecanismo define POSIX para establecer los atributos de un thread. Pon un ejemplo (no a nivel de código) donde existan threads con distintos atributos.**

(1.5 puntos)

La respuesta se encuentra en el apartado 6.1 y 6.3 del capítulo 6 del libro de la asignatura

3. **Realizar un programa donde dos threads ejecuten dos funciones. Una función debe calcular la suma de los primeros 100 números naturales y la otra debe escribir el resultado en pantalla. Obviamente, la segunda función debe esperar a que la primera haya terminado para no imprimir un resultado erróneo.**

(2 puntos)

Para resolver el problema de la cooperación entre tareas (una tarea debe sincronizarse con otra para realizar un trabajo determinado), se utilizan las llamadas que permitan esperar a que un determinado thread haya concluido. En le caso de los threads POSIX, dicha llamada es *pthread_join* cuya sintaxis y ejemplo de utilización aparece en el capítulo 6 (pág. 101) del libro de la asignatura.

Así pues, la solución **óptima** del problema sería:

```
// Examen Febrero 2001
#include <stdio.h>
#include <stdlib.h>
#include <pthread.h>

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

/* Declaración de los threads */
pthread_t thread1, thread2, thmain;
pthread_attr_t attr; /*atributos de los threads*/

int num, suma;

/* Definición de las funciones func1 y func2 */
void *func1 (void *arg)
{
   int i;
   printf("Soy el thread 1 y voy a ejecutar func1 \n");
   for (i=1;i<num+1;i++)
         suma=suma+i;

  printf("Soy el thread 1 y he terminado de ejecutar la funcion
          1\n");
   pthread_exit(NULL);     /* Provoca la terminacion del
thread*/
}

void *func2 (void *arg)
{
   int err;
   if (err = pthread_join(thread1, NULL))
         perror("Error, no he podido esperar al thread1");
   printf("Soy el thread 2 y voy a ejecutar func2 \n");
   printf("La suma de los 100 primeros numeros es: %d \n",suma);
   printf("Soy el thread 2 y he terminado de ejecutar la funcion
           2\n");
   pthread_exit(NULL);     /* Provoca la terminacion del
thread*/

}

/*Funcion main*/
int main()
{
   num = 100;
   suma = 0;
   pthread_attr_init (&attr);
   printf("Soy la funcion main y voy a lanzar los dos threads
\n");
   pthread_create (&thread2, &attr, func2, NULL);
   pthread_create (&thread1, &attr, func1, NULL);
   printf("Soy main: he lanzado los dos threads y termino\n");
   pthread_exit(NULL);
}
```

Existe una solución alternativa pero **NO ÓPTIMA** empleando exclusión mutua (mutex). Dicha solución no es óptima y por tanto debe ser considerada como no válida debido ya que requiere más recursos y es más lenta que la propuesta anteriormente (notemos que existen infinidad de soluciones para resolver el problema, por ejemplo, que el thread que imprime el resultado espere una hora antes de hacerlo. Así, seguramente el thread que realiza los cálculos debe haber terminado....).

No obstante, la anterior solución se transcribe a continuación y ha sido puntuada (con menor puntuación) en el ejercicio.

```
// Examen Febrero 2001-Solución con mutex
#include <stdio.h>
#include <stdlib.h>
#include <pthread.h>

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

/* Declaración de los threads */
pthread_t thread1, thread2, thmain;
pthread_attr_t attr;       /*atributos de los threads*/
pthread_mutex_t mutex;

int num, suma;

/* Definición de las funciones func1 y func2 */
void *func1 (void *arg)
{
       int i;
       printf("Soy el thread 1 y voy a ejecutar func1 \n");
       for (i=1;i<num+1;i++)
              suma=suma+i;

       printf("Soy el thread 1 y he terminado de ejecutar la funcion 1\n");
       pthread_mutex_unlock(&mutex);
       pthread_exit(NULL); /* Provoca la terminacion del thread*/
}

void *func2 (void *arg)
{
       int err;
       pthread_mutex_lock(&mutex);
       printf("Soy el thread 2 y voy a ejecutar func2 \n");
       printf("La suma de los 100 primeros numeros es: %d \n",suma);
       printf("Soy el thread 2 y he terminado de ejecutar la funcion 2\n");
       pthread_exit(NULL); /* Provoca la terminacion del thread*/

}

/*Funcion main*/
int main()
{
num = 100;
suma = 0;
pthread_attr_init (&attr);
pthread_mutex_init(&mutex,NULL);
pthread_mutex_lock(&mutex); /* Pone el mutex a 0; si se hace una op. P, el
proceso se bloquea*/
printf("Soy la funcion main y voy a lanzar los dos threads \n");
pthread_create (&thread2, &attr, func2, NULL);
pthread_create (&thread1, &attr, func1, NULL);
printf("Soy main: he lanzado los dos threads y termino\n");
pthread_exit(NULL);
}
```

4. Definir de forma clara y concisa qué es un Sistema Informático de Tiempo Real. Enumerar sus características fundamentales.

(1 punto)

(Capítulo 1 del libro)

Un Sistema Informático de Tiempo Real se define como un sistema basado en computador el cual, ***debe responder ante estímulos generados por el entorno dentro de un periodo de tiempo finito***. No importa solo que sea capaz de generar un resultado correcto sino que éste se produzca en un tiempo determinado, asimismo intereactúa con el entorno (mundo físico real) adquiriendo estímulos y estados del entorno, y generando una acción sobre dicho entorno.

Características fundamentales:

- Grandes y complejos
- Rápida respuesta
- Operación continua
- Reacción ante estímulos físicos
- Tolerancia a fallos en los componentes o en sus conexiones
- Fiable y seguro.
- Concurrencia entre sus componentes
- Implementación eficiente

5. Describir de forma clara y concisa las características fundamentales de un Sistema Operativo de Tiempo Real.

(1 punto)

(Capítulo 2 del libro)

- Determinismo: capacidad de estimar los tiempos de ejecución de cada tarea.
- Concurrencia: capacidad de mantener varias tareas en estados intermedios de su ejecución.
- Multiproceso: ejecución concurrente de procesos. Instancias de ejecución de un programa con recursos independientes (registros, pilas, memoria,...) y capacidad de planificación (asignación de tiempo de ejecución).
- Multihebra (mutlithread): ejecución concurrente de hebras (unidad de planificación sin recursos) que permita un cambio de contexto muy rápido.
- Manejo de excepciones: para la respuesta controlada ante fallos en el sistema.
- Mecanismos de Planificación: asignación de uso de CPU entre las diferentes tareas determinista con manejo de prioridades.
- Mecanismos de Temporización: que permita la ejecución de tareas en tiempos prefijados.
- Mecanismos de comunicación entre tareas: seguros frente a bloqueos, y gestión de sistemas distribuidos
- Acceso a los recursos hardware del sistema: dispositivos, interrupciones, ... para controlar la interacción con el entorno.

6. Definir qué es una Señal POSIX. Describir gráficamente los diferentes estados es los que se puede encontrar una señal. ¿Para qué sirve la máscara de bloqueo de una señal?

(1 punto)

(Apuntes de señales POSIX apartados 1 y 5.1)

Una **señal POSIX** es una notificación por software a un proceso o thread de la ocurrencia de un evento. Los eventos pueden ser generados por la ejecución del propio proceso (**síncronos**) o bien por otros procesos o el entorno exterior (**asíncronos**).

Estados en los que se puede encontrar una señal:

1. Una señal está asociada a un evento, por lo que cuando dicho evento se produce se dice que la señal se ha *generado*.

2. Se dice que la señal está *depositada* cuando el proceso asociado emprende una acción con base a ella.

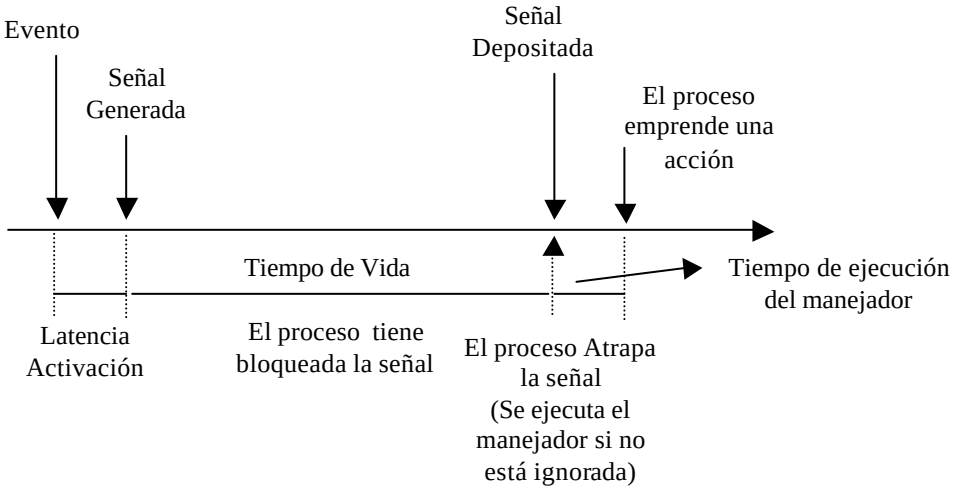
3. El *tiempo de vida* de una señal es el intervalo entre la generación y el depósito de ésta.

4. Se dice que una señal está *pendiente* si ha sido generada pero todavía no está depositada.

5. Un proceso *atrapa* una señal si éste ejecuta el manejador de señal cuando se deposita.

6. Una señal puede ser *ignorada* por el proceso, es decir que no sea ejecutado ningún manejador al ser depositada. Destacar que aunque no se ejecute ningún manejador la señal si que llega al proceso y éste puede determinar alguna acción en función de ello. El que una señal sea atrapada o ignorada es especificado en la configuración de la señal.

7. La acción que se emprende cuando se genera una señal depende de la *máscara* de la señal. La máscara contiene una lista de señales que en un determinado momento están *bloqueadas*. Si se genera una señal y está bloqueada no se pierde, queda pendiente de ser depositada hasta que sea desbloqueda.



Máscara de bloqueo de una señal: especifica para cada proceso o thread si una señal determinada que ha sido generada puede ser depositada o debe quedar pendiente de depósito. Una máscara es almacenada por el S.O. por cada thread. Si el bit correspondiente a la señal está activado (1) la señal está bloqueada para dicho thread, si está desactivado (0) la señal no está bloqueada.

La máscara de bloqueo permite por tanto especificar el thread dentro de un proceso que debe atender una señal asíncrona (notificada a todos los threads).

7. Para la red mostrada en la figura se pide:

a) Asignar una dirección IP de Red de clase C (cualquiera que sea válida).

b) Especificar la máscara de subred adecuada para cada una de las subredes

c) Asignar las direcciones IP de cada subred.

d) Asignar las direcciones IP a cada nodo (Estaciones y Routers).

e) Asignar la dirección IP de la pasarela (router) por defecto de cada nodo.

(2 puntos)

Nota: la selección de las direcciones concretas es libre siempre que cumpla los requerimientos del problema y sean direcciones IP válidas (no privadas).

Utilizar direcciones correlativas para los nodos en cada subred. Basta con indicar la primera y última dirección de cada nodo.

La solución se puede entregar sobre el gráfico del enunciado o bien sobre una reproducción correcta del mismo.

(Capítulo 11 del libro)

a) Una red de clase C se identifica por la secuencia 110xxxxx en los bits más significativos de la dirección IP por lo que el rango de direcciones IP será;

192.0.0.0 a 223.255.255.0

Excluyendo el rango de direcciones privadas 192.168.00 – 192.168.255.0 podemos elegir cualquiera.

Tomaremos por ejemplo la dirección de red **194.100.100.0** que está en el rango de direcciones Europeas (194.0.0.0 – 195.255.255.0)

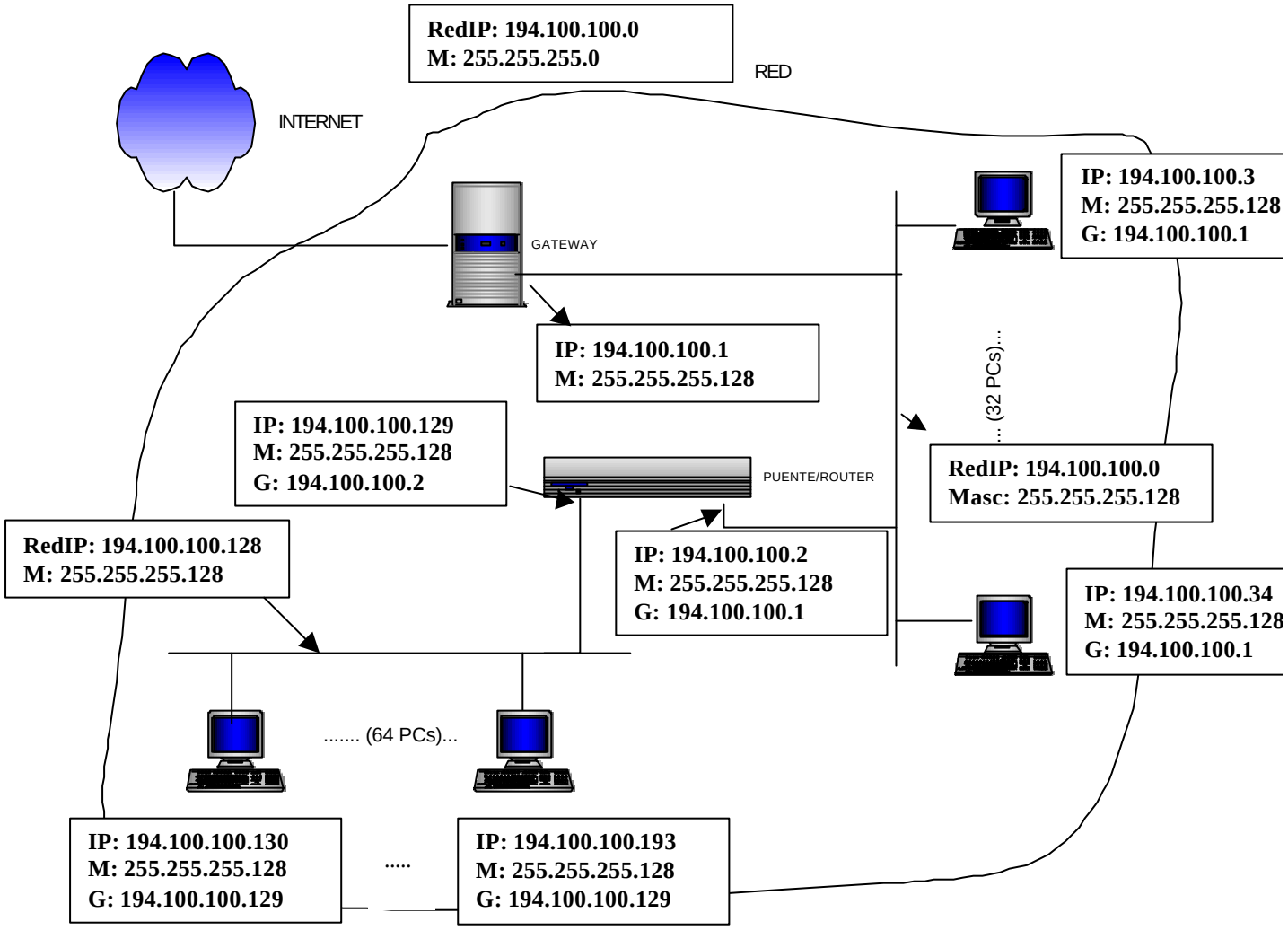
b) Ya que nuestra red está formada por dos subredes utilizaremos el bit más significativo del campo de subred/nodo (cuarto byte) para direccionar subredes. Para ello activaremos el bit más significativo de la máscara de subred en este campo (cuarto byte).

Los 7 bits restantes nos permiten direccionar $\mathbf{2^7-2=126}$ nodos, suficientes para los equipos presentes en cada subred. De este modo la máscara para las dos subredes quedará:

255.255.255.128

c) IP de cada subred:
Derecha (32PCs + router+ pasarela): **194.100.100.0**
Inferior (64 PCs + router): **194.100.100.128**

d) y e) sobre el gráfico:



Nota: **IP:** dirección IP
M: máscara de subred
G: pasarela por defecto

Duración del Examen: 2 ½horas