EXAMEN DE SISTEMAS INFORMÁTICOS DE TIEMPO REAL

SOLUCIÓN **Febrero 2002**

1. Enumera y comenta brevemente las ventajas e inconvenientes de los threads de núcleo frente a los threads de usuario.

(1 punto)

Las ventajas de threads de núcleo frente a los de usuario son:

- Son totalmente visibles por el núcleo y por tanto planificables, lo cual implica que pueden competir por todos los recursos del sistema y aprovechar así la ejecución paralela.
- No bloquea la ejecución de la aplicación cuando realiza llamadas al sistema bloqueantes ya que el planificador puede suspenderlo y continuar ejecutando otro thread.

Como inconveniente se puede citar:

- La creación y manejo de estos threads son casi tan costosas como la creación y manejo de procesos aunque la sincronización y el manejo de datos compartidos es mucho más ligera que en éstos. Por supuesto, la creación y manejo de threads de usuario es mucho más ligera (tanto en tiempo como en recursos)
-

2. A un sistema llegan tres procesos cuyo comportamiento se describe en la tabla siguiente:

Proceso	Instante de llegada	Tiempo de CPU
P1	0	12
P2	1	5
P3	1	9

Se pide:

a) (0.5 puntos) Dibujar el cronograma de la vida de los mismos en el sistema suponiendo que el algoritmo de planificación empleado es FCFS

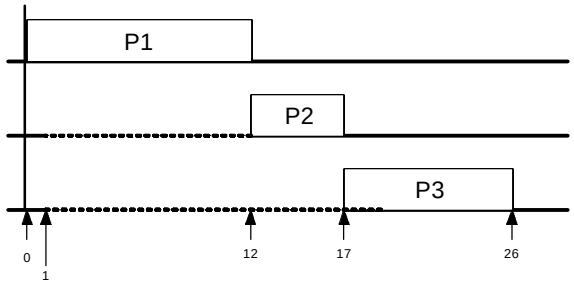


Figura 1. Cronograma del apartado a) (FCFS)

b) (0.5 puntos) Idem al caso anterior pero empleando en este caso un algoritmo de planificación RR con un valor de *quantum* de 4 unidades de tiempo.

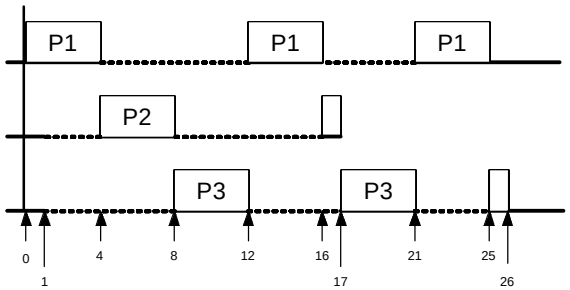


Figura 2. Cronograma del apartado b) (RR)

c) (0.5 puntos) Hallar el tiempo medio de espera para ambos casos y comentar el resultado obtenido.

(1.5 puntos)

Apartado a): t. medio de espera = (0+11+16)/3 = 9 u.t.
Apartado b): t. medio de espera = (13+11+16)/3 = 13.33 u.t.

Como se puede apreciar, en este caso, el tiempo medio de espera utilizando el algoritmo FCFS es menor que el obtenido utilizando RR (en contra de lo que en principio se pueda suponer). Esto es debido a que el algoritmo FCFS produce tiempos de espera altos cuando los procesos tienen tiempos de ejecución bastante dispares (no es el caso) y además los últimos en llegar tienen poco tiempo de ejecución). Este algoritmo sería recomendable en un entorno de proceso de trabajo por lotes.

Sin embargo, en un entorno interactivo (por ejemplo un entorno multiusuario pseudoparalelo) la segunda opción es mejor ya que el usuario puede percibir resultados antes de que todos los procesos anteriores al suyo hayan terminado de ejecutarse.

3. Se desea realizar un programa que evalúe las expresiones del tipo *(a*b)+(c*d)* de forma eficiente. Para ello se debe realizar un programa con threads que cumpla las siguientes características:

   - Los datos *a*, *b*, *c* y *d* se deben pedir por teclado
   - Se dispone de tres funciones:
     - *func1*: calcula el producto *(a*b)* y añade el cálculo a la variable global *resultado*
     - *func2*: calcula el producto *(c*d)* y añade el cálculo a la variable global *resultado*
     - *total*: imprime el contenido de la variable *resultado*
   - Cada una de las funciones debe ser ejecutada por un thread: *func1* por el thread *th1*, *func2* por el thread *th2* y *total* por el thread *result*
   - Los datos *a*, *b*, *c* y *d* **no son variables globales** y se deben pasar como parámetros a las respectivas funciones

(2.5 puntos)

El código del programa pedido se muestra a continuación.

```
#include <stdio.h>
#include <stdlib.h>
#include <sys/types.h>
#include <pthread.h>

// Declaración de prototipos
void * func1 (void *);
void * func2 (void *);
void * total (void *);

// Estructura para el paso de parámetros
typedef struct
{
  int dato1, dato2;
}datos;

// Declaración de threads
pthread_t th1, th2, result;

// Declaración de variables globales
pthread_attr_t attr; // Objeto atributo
static pthread_mutex_t exmut;  // Mutex
int resultado;

// Definición de funciones
void * func1 (void * arg)
{
  int a,b;
  datos *p = (datos *) (arg); // Cast
  a = (p->dato1);
  b = (p->dato2);
```



```
  printf("\nSoy el thread 1 y voy a sumar (a*b) a resultado");
  pthread_mutex_lock(&exmut);
  resultado = resultado + (a*b);
  pthread_mutex_unlock(&exmut);

  pthread_exit(NULL);
}

void * func2 (void * arg)
{
  int a,b;
  datos *p = (datos *) (arg);
  a = (p->dato1);
  b = (p->dato2);

  printf("\nSoy el thread 2 y voy a sumar (c*d) a resultado");
  pthread_mutex_lock(&exmut);
  resultado = resultado + (a*b);
  pthread_mutex_unlock(&exmut);

  pthread_exit(NULL);
}

void * total (void * arg)
{
  int err;

  // Hay que comprobar que ambos threads han terminado
  if (err = pthread_join(th1, NULL))
      printf("\nError: Soy result y no he podido esperar a th1");
  else
      printf("\nSoy result y th1 a terminado");

  if (err = pthread_join(th2, NULL))
      printf("\nError: Soy result y no he podido esperar a th2 ");
  else
      printf("\nSoy result y th2 a terminado \n");

 printf("\nSoy result y el resultado es : %d\n", resultado);

  pthread_exit(NULL);
}


int main (void)
{
      datos datos1, datos2 ;
      resultado = 0;

      pthread_attr_init (&attr); // Inicialización del objeto atributo
      pthread_mutex_init(&exmut,NULL); // Inicialización del mutex

      printf("\nIntroduce el valor de a: ");
      scanf("%d",   &datos1.dato1);
      printf("\nIntroduce el valor de b: ");
      scanf("%d",   &datos1.dato2);
      printf("\nIntroduce el valor de c: ");
```

```
        scanf("%d",   &datos2.dato1);
        printf("\nIntroduce el valor de d: ");
        scanf("%d",   &datos2.dato2);

        pthread_create(&th1, &attr, func1, &datos1);
        pthread_create(&th2, &attr, func2, &datos2);
        pthread_create(&result, &attr, total, NULL);

        printf("\nSoy main, he lanzado la ejecución y salgo\n");

        pthread_exit(NULL);
}
```


4. Explicar detalladamente las diferencias entre los SITR controlados por eventos y los controlados por tiempo. Indicar ventajas e inconvenientes, así como ejemplos de aplicación.

(1 punto)

La ***arquitectura controlada por eventos o interrupciones*** establece la ejecución de un componente o tarea basándose en la aparición de una interrupción o señal generada por un evento externo. El sistema genera de este modo una interrupción del computador de forma que pueda ser atendido el evento por la rutina encargada de dicho control. Por ejemplo, cuando un robot móvil se mueve por un espacio rodeado de obstáculos ante la aparición de una colisión detectada por un sensor externo el sistema puede parar el comportamiento actual y activar una nueva tarea, '*dormida*' hasta entonces, encargada de evitar el obstáculo.

La ***arquitectura controlada por tiempo*** opera 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.

Ambos tipos de arquitectura son comunes en las aplicaciones prácticas y tienen sus ventajas y sus inconvenientes. Los sistemas manejados por tiempo son más sencillos ya que los conceptos que determinan su diseño son más explícitos. También son considerados más robustos ya que permiten interfaces entre subsistemas muy controlados y con un funcionamiento independiente basado en relojes locales. Por otra parte, los sistemas basados en interrupciones constituyen un mecanismo mucho más eficaz para responder ante eventos externos no regulares.

5. Explicar, para el mecanismo de Señales POSIX, los siguientes conceptos:

a) Generación de una señal: mecanismos y funciones básicas. (0.5p)

Una **señal** es una notificación por software a un proceso/thread de la ocurrencia de un evento. Es el propio sistema operativo el que se encarga de su generación en función del evento asociado. Estos eventos pueden tener un origen externo, como la activación de un pin de la CPU o de un determinado registro o puerto del computador, También pueden estar asociados a interrupciones hardware del propio computador, como por ejemplo cuando el contador de un reloj alcanza cierto valor.

Aparte de este tipo de eventos, una señal puede ser generada por los propios procesos que se están ejecutando en el computador (software). Permite de este modo un mecanismo para que los diferentes procesos se comuniquen entre sí la aparición de eventos que determinan la ejecución del programa.

Funciones básicas para la generación una señal desde otro proceso:

```
int kill(pid_t pid, int sig);
int sigsend(idtype_t idtype, id_t id, int sig);
int  sigqueue(pid_t  pid,  int  signo, const  union  sigval
value);
```

b) Diferencias entre señales síncronas y asíncronas. (0.5p)

1. **Señales síncronas**: son aquellas que son generadas por la ejecución del código, y por tanto son generadas por un thread concreto. Por ejemplo SIGBUS o SIGFPE son generadas por errores del código ejecutado. En este caso la señal puede ser depositada únicamente por el thread que generó el evento

2. **Señales asíncronas**: el resto de señales producidas por llamas explícitas (kill) o por eventos no asociados al código en ejecución. En este caso la señal va dirigida a todos los threads del proceso, aunque solamente uno de ellos puede depositarla.

c) Bloqueo de una señal: definición, estructuras y funciones utilizadas (0.5p)

El bloqueo de una señal supone que dicha señal quede pendiente de depósito en el caso de que sea generada, es decir la señal no se pierde sino que su atención es postergada. El bloqueo de una señal por parte de un proceso es específico de cada thread del proceso. Por lo tanto, dentro de un proceso pueden existir threads que hayan bloqueado la señal y threads que no la hayan bloqueado. Estos últimos son los únicos que pueden depositar la señal.

La gestión del bloqueo de señales se realiza mediante lo que se denomina **máscara de señal** esta máscara se maneja mediante una estructura de tipo ***sigset_t*** almacenada por el S.O. para cada thread. La máscara de señal indica la configuración de cada señal (bloqueada o no bloqueada). Adicionalmente podemos definir nuevos **conjuntos de señales** de tipo ***sigset_t*** que nos permitirán activar o desactivar grupos de señales en cada momento.

Funciones para el manejo de conjuntos de señales:

sigemptyset(sigset_t *set***)**: inicializa un conjunto excluyendo todas las señales
sigfillset(sigset_t *set***)**: inicializa un conjunto incluyendo todas las señales
sigaddset(sigset_t *set***)**: incluye la señal indicada en el conjunto
sigdelset(sigset_t *set***)**: excluye la señal indicada en el conjunto

funciones para el bloqueo de señales:

int **sigprocmask**(int how, const sigset_t *set, sigset_t *oset);

int **pthread_sigmask**(int how, const sigset_t *set, sigset_t *oset);

Parámetros:

- ***how***: indica la acción a realizar: **SIG_BLOCK**, (para bloquear un conjunto de señales), **SIG_UNBLOCK** para desbloquearlas, **SIG_SETMASK** copia el conjunto dato en ***set*** en la máscara.
- ***set***: es un puntero al conjunto de señales que queremos bloquear o desbloquear (tal como se describió anteriormente)
- ***oset***: si no es NULL almacena el valor de la máscara anterior a la operación

d) Manejador de una señal: definición y funciones. (0.5p)

El manejador es una función que es ejecutada cuando una señal es depositada por un thread. Su asignación se realiza mediante la función ***sigaction()***. Podemos elegir tres opciones:

1. Ignorar la señal: no se ejecutaría ningún manejador (a pesar de ello la señal si que llegaría al proceso que podría ejecutar una acción)

2. Asignar un manejador por defecto

3. Asignar nuestro propio manejador

Para evitar que se deposite una señal mientras se está configurando se debe, en primer lugar, bloquear la señal en todo el proceso antes de llamar a la función ***sigaction()***.

```
int sigaction(int sig, const struct sigaction *act,
                  struct sigaction *oact);
```

6. Para interconectar las redes de una empresa situadas en dos oficinas distantes se propone el esquema indicado en la figura anexa. Se utiliza un enlace RDSI para unir ambas oficinas mediante dos routers (1 y 2) (*la conexión a la red RDSI se realiza mediante un enlace punto a punto a través de los terminales de red TR1 estandarizados*). En la oficina A se dispone de dos subredes interconectadas a través del router 3. Asimismo, en esta última oficina se dispone de una conexión con acceso a internet a traves del router 4 (gateway) que controla el tráfico hacia internet de todas las oficinas.

   El conjunto completo constituye la red de la empresa. El enlace RDSI debe considerarse como un enlace punto a punto (protocolo PPP) entre los routers 1 y 2. Para direccionar las subredes indicadas se ha asignado la dirección IP 194.160.132.0, reservándose la dirección IP 194.160.132.1 para el Gateway.

   Se pide (razonando brevemente cada una de las respuestas):
   a) Asignar las direcciones IP y máscaras de subred a cada una de las subredes y nodos. (0.5)
   b) Tabla de enrutamiento para cada uno de los routers presentes exceptuando la pasarela a Internet. (0.5)
   c) Pasarela por defecto para cada uno de los nodos. (0.5)
   d) Indicar la secuencia de mensajes y protocolos involucrados en una transmisión UDP entre un PC de la oficina B y el servidor ubicado en la oficina A. (0.5)

   (2 puntos)

a) Se trata de Una red de clase C (se identifica por la secuencia 110xxxxx en los bits más significativos de la dirección IP) Nuestra red está formada por cuatro subredes:
   2 redes en la oficina A,
   1 red en la oficina B
   1 red Punto a punto (RDSI) entre ambas oficinas

   El número máximo de nodos por subred es de 21 en la oficina B por lo que bastan 5 bits por subred (**2^5-2=30** nodos como máximo).

   Usaremos los tres bits más significativos para dire3ccionar subredes (**2^3=8 subredes)**

   De este modo la máscara para todas las subredes quedará:

   255.255.255.224

   El resto de direcciones IP sobre el gráfico

b) Tabla enrutamiento Router 1:

Destino	Máscara	Gateway
0.0.0.0 (Default)	0.0.0.0	194.160.132.1
194.160.132.32	255.255.255.224	194.160.132.2
194.160.132.0	255.255.255.224	194.160.132.3
194.160.132.128	255.255.255.224	194.160.132.98
194.160.132.98	255.255.255.224	194.160.132.97

Tabla enrutamiento Router 2:

Destino	Máscara	Gateway
0.0.0.0 (Default)	0.0.0.0	194.160.132.97
194.160.132.97	255.255.255.224	194.160.132.98
194.160.132.128	255.255.255.224	194.160.132.129

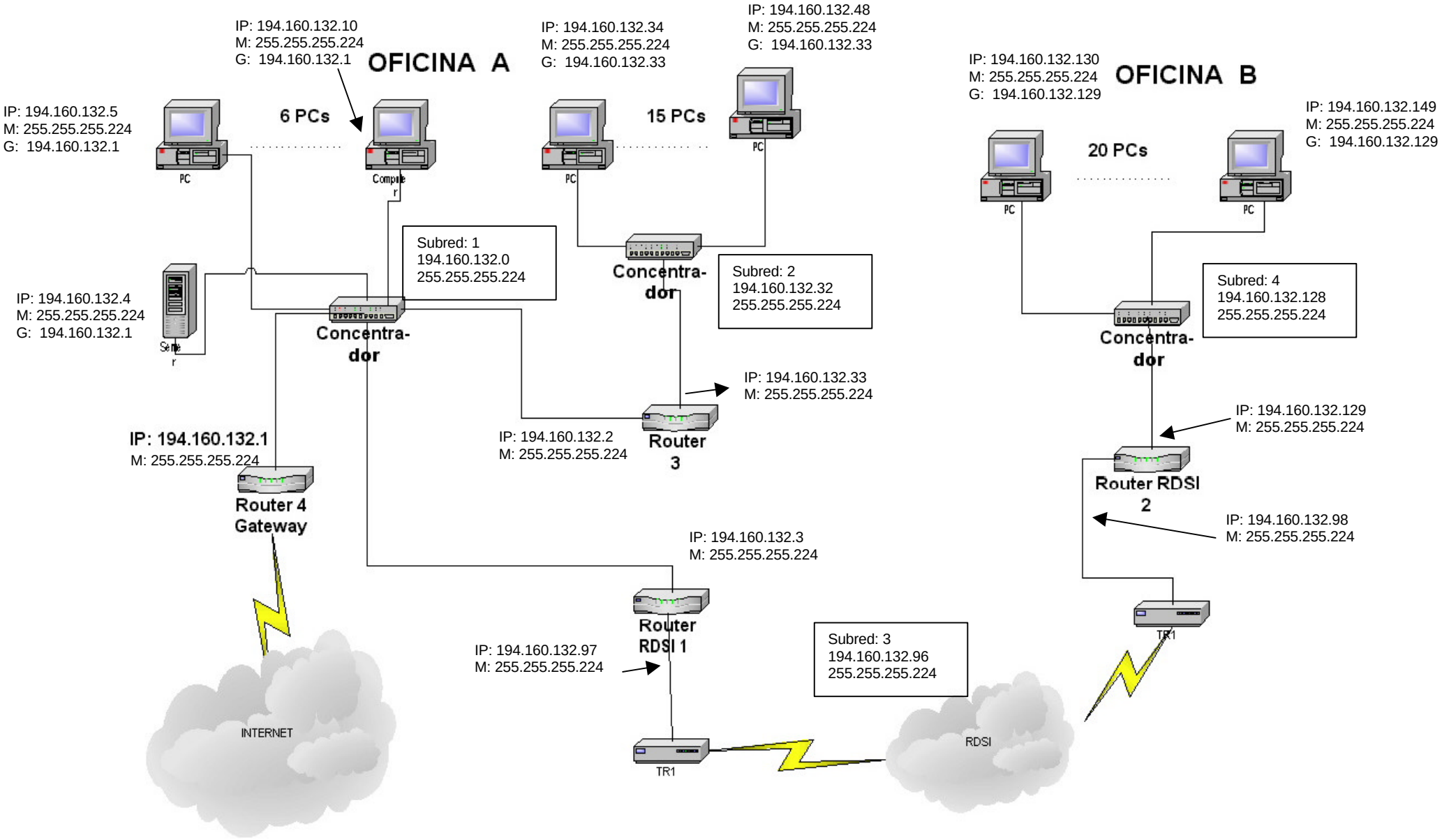
Tabla enrutamiento Router 3:

Destino	Máscara	Gateway
0.0.0.0 (Default)	0.0.0.0	194.160.132.1
194.160.132.32	255.255.255.224	194.160.132.33
194.160.132.0	255.255.255.224	194.160.132.2
194.160.132.128	255.255.255.224	194.160.132.3

c) en el gráfico.

d) Secuencia de tramas y protocolos:

- 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** (194.160.132.129).
- 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
- Conocida la dirección MAC se encapsula el datagrama en una trama **Ethernet** y se transmite al router.
- El router 2 recibe la trama y la pasa al nivel **IP**. Comprueba en la tabla de enrutamiento la dirección IP destino enviando el datagrama al siguiente router (194.160.132. 97).
- El router 1 recibe la trama y la pasa al nivel **IP**. Comprueba en la tabla de enrutamiento la dirección IP destino enviando el datagrama al interfaz (194.160.132. 4).
- El destino final del datagrama 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
- Finalmente se enviaría el datagrama **IP** al nodo destino (194.160.132.4) encapsulado en una trama **Ethernet**.
- Si existiera fragmentación se recompondrían los fragmentos formando un único datagrama **UDP**


OFICINA A
IP: 194.160.132.10
M: 255.255.255.224
G: 194.160.132.1
IP: 194.160.132.34
M: 255.255.255.224
G: 194.160.132.33
IP: 194.160.132.48
M: 255.255.255.224
G: 194.160.132.33
IP: 194.160.132.5
M: 255.255.255.224
G: 194.160.132.1
6 PCs
15 PCs
PC
Compute r
Subred: 1
194.160.132.0
255.255.255.224
Subred: 2
194.160.132.32
255.255.255.224
Concentra-dor
IP: 194.160.132.4
M: 255.255.255.224
G: 194.160.132.1
Server
IP: 194.160.132.33
M: 255.255.255.224
IP: 194.160.132.2
M: 255.255.255.224
Router 3
IP: 194.160.132.1
M: 255.255.255.224
Router 4 Gateway
IP: 194.160.132.3
M: 255.255.255.224
Router RDSI 1
IP: 194.160.132.97
M: 255.255.255.224
Subred: 3
194.160.132.96
255.255.255.224
TR1
INTERNET
RDSI
OFICINA B
IP: 194.160.132.130
M: 255.255.255.224
G: 194.160.132.129
IP: 194.160.132.149
M: 255.255.255.224
G: 194.160.132.129
20 PCs
Subred: 4
194.160.132.128
255.255.255.224
IP: 194.160.132.129
M: 255.255.255.224
Router RDSI 2
IP: 194.160.132.98
M: 255.255.255.224
TR1