UNIVERSIDAD CARLOS III DE MADRID
AREA DE ARQUITECTURA Y TECNOLOGÍA DE COMPUTADORES
GRADO EN INGENIERÍA INFORMÁTICA. SISTEMAS DISTRIBUIDOS

Para la realización del presente examen se dispondrá de **2 horas y 30 minutos. NO** se podrán utilizar libros, apuntes **ni** calculadoras de ningún tipo. **Responda** a los ejercicios en el **espacio reservado**.

Alumno: _____ **Grupo:** _____

Pregunta 1 (2 puntos): Conteste razonadamente a las siguientes preguntas:

a) Sea un sistema distribuido compuesto por N nodos. En este sistema una petición de servicio toma `t` unidades de tiempo. Considere `P` peticiones del mismo servicio que se ejecutan en `P*t` unidades de tiempo. Defina en qué consiste la escalabilidad de un sistema distribuido. ¿Es este sistema escalable? Justifique su respuesta.

b) Enumere las ventajas y los inconvenientes del uso de caché de bloques en los clientes de un servicio de ficheros distribuido con respecto a un cliente que no usa caché de bloques.

c) ¿En qué consiste la computación voluntaria? Cite algunos proyectos que usen este paradigma.

Solución:

a) Escalabilidad es la capacidad de mantener el rendimiento cuando el número de usuario o recursos de un sistema distribuido crecen significativamente. En el ejemplo descrito, el tiempo de respuesta aumenta de manera proporcional al número de peticiones, por tanto este sistema es escalable. Si el tiempo de respuesta x fuera mucho mayor que el tiempo lineal (x>>>P*t) entonces el sistema no escalaría, puesto que no se puede mantener un tiempo de respuesta similar al de una petición.

b) Ventajas:

- Mejora del rendimiento del sistema de ficheros distribuido (rendimiento cercano al de un sistema centralizado).
- Menor carga en el servidor y en la red. Se permiten transferencias más grandes por la red
- Facilita el crecimiento proporcional del rendimiento del sistema.

Inconvenientes:

- Dificultades relacionadas con el mantenimiento de la coherencia de caché.

c) La computación voluntaria (o computación pública) es un paradigma de computación distribuida en el que los usuarios (también llamados voluntarios) ceden sus recursos de computación (e.g. ciclos de CPU, almacenamiento, etc.) para resolver problemas de tipo científico o de ingeniería que requieren muchos recursos. Ejemplos: SETI@home, LHC@home, Rosetta@home.

Pregunta 2 (1 punto): Dado el siguiente mensaje de petición SOAP:

```
<SOAP-ENV:Envelope   xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
  SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
  <SOAP-ENV:Body>
     <m:ConsultarPrecio xmlns:m="http://example.com/stockquote.xsd">
          <item>libro</item>
     </m: ConsultarPrecio >
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Se pide:

a) ¿Qué Lenguaje de Definición de Interfaz utilizan los servicios web basados en SOAP?

b) ¿Qué función encapsula este servicio web?

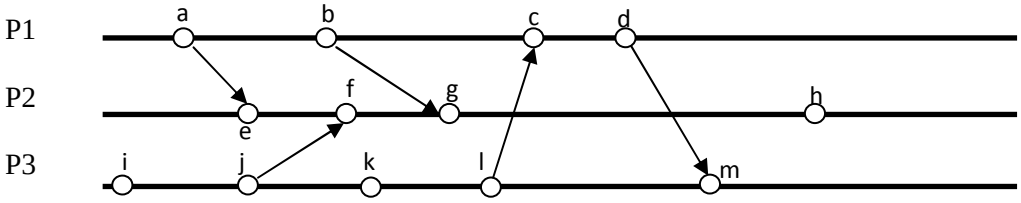
c) ¿Qué protocolo de transporte puede utilizarse para encapsular este mensaje?

d) Identificar los campos principales del mensaje SOAP.

Solución:

a) Web Services Description Language (WSDL).
b) El servicio web es *ConsultarPrecio.*
c) No se puede saber, podría estar encapsulado en cualquier protocolo del nivel de aplicación siguiendo las normas SOAP, como por ejemplo HTTP.
d) Los campos del mensaje SOAP son: el *envelope* (o contenedor) del mensaje SOAP, el cual contiene la descripción en XML del mensaje SOAP. El *envelope* del mensaje contiene a su vez otros dos campos: cabecera y cuerpo. La cabecera es opcional y no aparece en este ejemplo. El cuerpo del mensaje (body) contiene el mensaje de petición (en este caso) del servicio web que se invoca, en el ejemplo *ConsultarPrecio.* Cada uno de los campos del mensaje puede tener distintos atributos; en el mensaje aparece el espacio de nombres para el Envelope de SOAP y para el mensaje de petición.

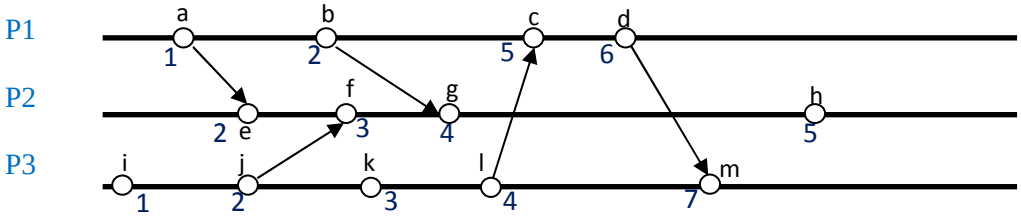
Pregunta 3 (1 punto): Considere los procesos P1, P2 y P3 que ejecutan en un sistema distribuido. Estos procesos generan los eventos marcados en la siguiente figura.



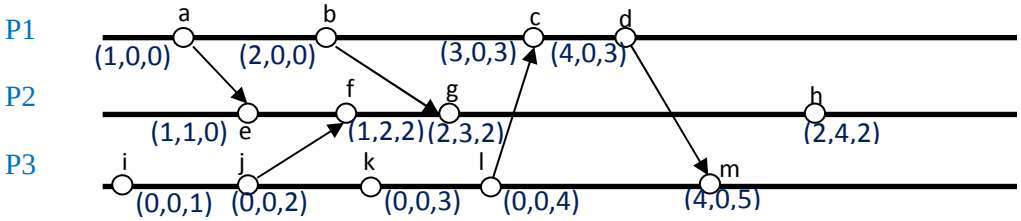
Se pide:
a) Indicar tres parejas de eventos ordenados utilizando las relaciones de causalidad potencial de Lamport.
b) Indicar tres parejas de eventos que sean concurrentes ¿Por qué son concurrentes?
c) Usando los relojes lógicos de Lamport, indique las marcas de tiempo para los eventos de los procesos anteriores.
d) Usando relojes vectoriales, defina las marcas de tiempo para los eventos de los procesos anteriores.
e) ¿Qué limitaciones tienen los relojes lógicos de Lamport? ¿Cómo tratan esas limitaciones los relojes vectoriales?

Solución:

a) a→b, b→g,d→m
b) a||i, h||d, b||l
c) Relojes lógicos de Lamport



d) Relojes vectoriales



e) Con relojes lógicos, dados dos eventos, a y b, si a→b entonces RL(a)<RL(b); sin embargo lo contrario no se puede garantizar, es decir si RL(a)<RL(b) no podemos saber si a→b. Con los relojes vectoriales, a→b sii RL(a)<RL(b), es decir lo contrario también se garantiza.

Pregunta 4 (2 puntos): Un sistema distribuido usa la siguiente interfaz para realizar el paso de mensajes:

- `send(i,msg)` – envía el mensaje `msg` al proceso con identificador `i`
- `receive(i,&msg)` – recibe el mensaje `msg` del proceso con identificador `i`

El sistema distribuido está compuesto por N procesos, donde los identificadores de proceso **i** oscilan entre 0 y N-1, y donde cada proceso ejecuta el siguiente código.

```
1:    mi_funcion(int i){
2:        if (i==0)
3:              send(i+1,msg);

4:         while(1){
5:              if (i==0)  j=N;
6:              else  j=i;
7:              receive(j-1,&msg)
8:              // Código del proceso id
9:              if (i==N-1) j=-1;
10:             else j=i;
11:             send(j+1,msg);
12:       }
13:   }
```

Conteste razonadamente a las siguientes cuestiones:

a) ¿Cuántas veces como máximo puede ejecutarse concurrentemente la operación `receive` de la línea 7?
b) ¿Cuántas veces como máximo puede ejecutarse concurrentemente el código en la línea 8?
c) ¿Cómo están ordenados lógicamente los procesos que ejecutan en este sistema distribuido?
d) ¿Qué aplicación tiene este algoritmo?

Solución:

a) La operación `receive` podría ejecutarse concurrentemente tantas veces como procesos en el sistema distribuido, por tanto N veces.
b) El código de la línea 8 sólo puede ser ejecutado concurrentemente por 1 proceso, dado que sólo hay un proceso que puede hacer `receive` sin bloquearse. El proceso que ejecuta inicialmente este código es el proceso con identificador 1, dado que el proceso con identificador 0 hace `send` al siguiente proceso, es decir al proceso 1 (Líneas 2 y 3).
c) Los procesos están ordenados lógicamente en forma de anillo: el proceso i envía al proceso i+1 y él recibe del proceso i-1. El primer y último proceso se conectan entre sí para cerrar el anillo.
d) Este algoritmo puede ser aplicado para resolver el problema de la sección crítica en un sistema distribuido de manera descentralizada.

Pregunta 5 (4 puntos): Se desea implementar una aplicación cliente-servidor para la consulta de archivos que contienen imágenes médicas. El servicio a implementar incluye las siguientes operaciones:

- Búsqueda de una imagen (archivo) en un servidor de imágenes. El cliente pregunta al servidor si una imagen existe en el servidor. El cliente enviará el nombre de la imagen (nombre del archivo) y el servidor enviará un código indicando si existe o no esa imagen.

- Obtener una imagen desde el servidor de imágenes. El cliente puede recuperar del servidor una imagen (archivo). El cliente enviará al servidor el nombre de la imagen y el servidor enviará la imagen al cliente para que este almacene la imagen en un fichero local.

- Almacenar una imagen en el servidor de imágenes. El cliente que accede al servicio puede almacenar en el servidor una nueva imagen. Una imagen viene dada por su nombre, su fecha de creación y el archivo de la imagen. El contenido de este archivo se enviará al servidor para su almacenamiento.

Tenga en cuenta que el archivo de la imagen puede tener cualquier tamaño.

Se pide:

a) Diseñar la aplicación cliente-servidor, indicando y especificando todos los aspectos necesarios para su diseño. Como parte del diseño, describa detalladamente el protocolo de servicio.

b) De acuerdo al diseño anterior, implementar en el lenguaje de programación C el código del servidor.

Solución:

a) En una aplicación cliente-servidor necesitamos tener en cuenta al menos los siguientes aspectos de diseño:

Para diseñar una aplicación cliente-servidor necesitamos considerar aspectos comunes a las aplicaciones distribuidas:

- **Nombrado**: necesitamos saber cómo identificar la máquina donde ejecuta la aplicación servidora. Para ello usamos direcciones IP estáticas y asumiremos que el cliente conoce la IP (o el nombre) y el puerto donde ejecuta el servidor, en el ejemplo el puerto 4200.

- **Escalabilidad**: para proporcionar un servicio escalable (que siga siendo efectivo mientras el número de peticiones aumenta) vamos a diseñar un servidor concurrente, es decir, aquel que puede atender simultáneamente varias peticiones de servicio al mismo tiempo. El servidor concurrente podría ser implementado con procesos convencionales o procesos ligeros, siendo ésta última la opción más eficiente. Dado que vamos a considerar un servidor concurrente necesitamos proteger aquellas variables compartidas que pudieran dar lugar a condiciones de carrera y por tanto llevar a resultados incorrectos en la aplicación (aspectos de concurrencia y sincronización).

- **Heterogeneidad:** dado que las máquinas cliente y servidor pueden tener arquitecturas de hardware distintas, para realizar el envío de los datos a la red deben traducirse al estándar de red (*network order o big endian*); análogamente en la recepción los datos desde la red deben pasarse al formato de host. Estas dos operaciones se denominan *marshalling* y *unmarshalling* respectivamente.

- El protocolo de transporte a utilizar va a depender de los requisitos de fiabilidad de la aplicación. Los protocolos de transporte implementados sobre IP son TCP y UDP. Ambas aplicaciones (cliente y servidor) deben estar de acuerdo en el protocolo de transporte a utilizar. Seleccionamos TCP como protocolo de transporte a usar en la aplicación. No se necesita implementar aspectos de calidad de servicio (QoS). El servidor por tanto, será orientado a conexión dado que el protocolo de transporte que vamos a utilizar es TCP. Esto implica que deberá existir un establecimiento de la conexión entre el cliente y servidor previo al intercambio de los datos. Además, el servidor será sin estado es decir, no es necesario mantener información global ni información de cada cliente sobre peticiones anteriores del servicio.

- El protocolo de servicio: El protocolo de servicio define el intercambio de mensajes entre la aplicación cliente y servidor y además el formato de los mensajes. Para cada petición se va a establecer una conexión, se va a enviar los argumentos de la petición y se va a recibir el resultado. Posteriormente se cierra la conexión. Para cada servicio (buscar, obtener y almacenar) vamos a definir los mensajes de petición y de respuesta. Existen tres servicios que debe ofrecer el servidor:
  - Buscar una imagen
  - Obtener una imagen

- Almacenar una imagen

Para poder identificar cada petición, se puede utilizar un byte que se envía del cliente al servidor al establecer la conexión. Se puede utilizar 0 para búsqueda, 1 para obtener imagen y 2 para almacenar una imagen

NOTA: Notación E: Emisor; R: Receptor.

Petición **Búsqueda**:

Se envían los siguientes mensajes:

- Mensaje E1: El cliente envía el identificador de la petición: un número entero que identifica la operación a realizar (e.g. 0).
- Mensaje E2:
  - Tamaño del nombre del archivo de imagen: longitud en bytes del nombre del archivo de imagen a buscar.
  - Nombre de la imagen: cadena de caracteres de hasta un máximo determinado (según la longitud indicada en el mensaje E2) indicando el nombre de la imagen a buscar en el servidor.

El mensaje de respuesta podría ser:

- Mensaje R3: Código de respuesta: un número entero, que indica si la imagen existe o no en el servidor (por ejemplo un 1 si existe o un 0 en caso contrario).

Petición **Obtener una imagen**:

Se envían los siguientes datos:

- Mensaje E1: El cliente envía el identificador de la petición: un número entero que identifica la operación a realizar (e.g. 1).
- Mensaje E2:
  - Tamaño del nombre del archivo de imagen: longitud en bytes del nombre del archivo de imagen a buscar.
  - Nombre de la imagen: cadena de caracteres de hasta un máximo determinado (según la longitud indicada en el mensaje E2) que representa el nombre de la imagen a buscar en el servidor.

El mensaje de respuesta podría ser:

- Mensaje R3: Código de respuesta: un número entero, que indica éxito o error en la operación. 1 éxito, 0 error.
- Mensaje R4:
  - Tamaño del archivo de imagen: un número entero que indica el tamaño en bytes de la imagen que va a enviar el servidor al cliente.
  - El archivo de imagen: la imagen se enviará en bloques de un determinado tamaño, por ejemplo 8 KB, hasta llegar al fin de fichero.

Petición **Almacenar una imagen**:

Se envían los siguientes datos:

- Mensaje E1: El cliente envía el identificador de la petición: un número entero que identifica la operación a realizar (e.g. 2).
- Mensaje E2:
  - Tamaño del nombre del archivo de imagen: longitud en bytes del nombre del archivo de imagen a buscar.

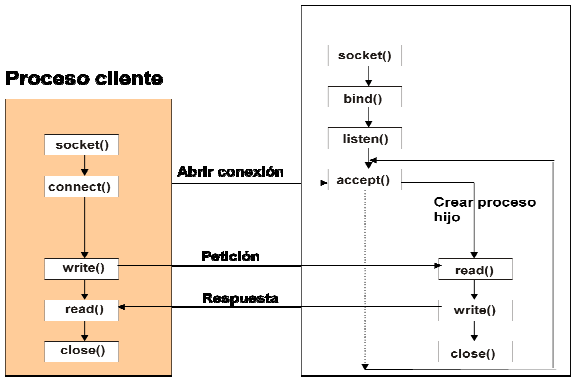
- Nombre de la imagen: cadena de caracteres de hasta un máximo determinado (de acuerdo al tamaño enviado en el mensaje E2) indicando el nombre de la imagen a buscar en el servidor.

- Mensaje E4:
  - Tamaño de la imagen: un número entero que indica el tamaño en bytes de la imagen.
  - El archivo de imagen: la imagen se enviará del cliente al servidor en en bloques de un determinado tamaño, por ejemplo 8 KB, hasta llegar al fin de fichero.

El mensaje de respuesta podría ser:

- Mensaje R3: Código de respuesta: un número entero, que indica éxito o error en la operación. Este mensaje se recibiría justo después de enviado el mensaje E2 y antes de los mensajes E4. El valor 1 indica éxito, 0 error.

El modelo de comunicación TCP es el siguiente:



a) De acuerdo al diseño anterior, implementar en el lenguaje de programación C el código del servidor.

```
#include <sys/types.h>
#include <sys/socket.h>

#define MAX_NONMBRE      1024
#define MAX_BLOQUE       8192
#define TRUE             1
#define FALSE            0


pthread_mutex_t   m;
pthread_cond_t    c;
int               busy = FALSE;


int enviar (int socket, char *mensaje, int longitud) {
   int r;
   int l = longitud;

   do {
         r = write (socket, mensaje, l);
         l = l - r;
         mensaje = mensaje + r;
   }while ((l>0) && (r>=0));

   if (r < 0)
         return (-1); /* fallo */
   else
         return (longitud);
}

int recibir (int socket, char *mensaje, int longitud) {
   int r;
   int l = longitud;
```

```
    do {
          r = read (socket, mensaje, l);
          l = l – r;
          mensaje = mensaje + r;
    }while ((l>0) && (r>=0));

    if (r < 0)
          return (-1); /* fallo */
    else
          return (longitud);
}


void main(int argc, char *argv[])
{
    struct sockaddr_in server_addr,  client_addr;
    int sd, sc;
    int len;
    pthread_attr_t attr;


    if ((sd =  socket(AF_INET, SOCK_STREAM, IPPROTO_TCP))<0) {
          printf (“SERVER: Error en el socket”);
          exit(1);
    }

    /* se asocia la dirección al puerto */
    bzero((char *)&server_addr, sizeof(server_addr));
    server_addr.sin_family    = AF_INET;
    server_addr.sin_addr.s_addr = INADDR_ANY;
    server_addr.sin_port      = htons(4200);

    if (bind(sd, &server_addr, sizeof(server_addr)) < 0) {
          printf (“SERVER: Error en el bind”);
          exit(1);
    }



    pthread_mutex_init(&m, NULL);
    pthread_cond_init(&c, NULL);

    pthread_attr_init(&attr);
    pthread_attr_setdetachstate(&attr, PTHREAD_CREATE_DETACHED);

    for(;;) {
          sc = accept(sd, (struct sockaddr *)  &cliente, &len);
          if (sc < 0){
                break;
          }

          /* se crea un thread para atender la petición*/
          if (pthread_create(&thid, &attr, tratar_peticion, &sc) != 0){
                close (sc);
                continue;
          }

          /* esperar a que el hijo copie el descriptor */
          pthread_mutex_lock(&m);
          while(busy == TRUE)
                pthread_cond_wait(&m, &c);
          busy = TRUE;
          pthread_mutex_unlock(&m);
    }
    exit(0);

}
```

```
void tratar_peticion(int * s)
{
	int s_local;
	int peticion;

	pthread_mutex_lock(&m);
	s_local = *s;
	busy = FALSE;
	pthread_cond_signal(&c);
	pthread_mutex_unlock(&m);

	// recibe la petición
	if (receive(s_local, &peticion, sizeof(int)) < 0) {
		/* error */
		close(s_local);
	}
	else
	{
		peticion = ntohl(peticion); // enteros en formato de red
		switch (peticion){
			case 0: busqueda(s_local);
				break;

			case 1: obtener_imagen(s_local);
				 break;

			case 2: almacenar_imagen(s_local);
				 break;
		 }
		close(s_local);
	}
	pthread_exit(NULL);
}

int busqueda(int s)
{
	int size;
	char nombre[MAX_NOMBRE];
	int df;
	int respuesta;

	// recibe el tamaño del archivo
	if (receive(s, &size, sizeof(size))<0){
		return(-1);
	}
	size = ntohl(size);   // de formato de red a formato de host

	// recibe el nombre del archive
	// se asume que no es mayor de MAX_NOMBRE bytes
	if (receive(s, &nombre, size) < 0) {
		return(-1);
	}

	/* se abre el archive para comprobar si existe */
	df=open(nombre, O_RDONLY);
	if (df < 0)
		respuesta=0;   // fallo
	else{
		respuesta=1;   // exito
		close(df);
	}
	// se envía la respuesta, pasar a formato de red
```

```
        respuesta = htonl(respuesta);
        if (send(s, (char*)&respuesta, sizeof(int)) < 0) {
                return(-1);
        }
        return(0);

}
int obtener_imagen(int s){
        int size;
        char nombre[MAX_NOMBRE];
        char buffer[MAX_BLOQUE]
        int df;
        int respuesta;
        int num=0, totales=0, tamanyo=0;

        // recibe el tamaño del nombre del archivo
        if (receive(s, &size, sizeof(size)) < 0) {
                return(-1);
        }
        size = ntohl(size); // pasar a formato de host

        // recibe el nombre del archivo
        // se asume que no es mayor de MAX_NOMBRE bytes
        if (receive(s, &nombre, size) < 0) {
                return(-1);
        }




        df=open(nombre, O_RDONLY);

        if (df<0){
                respuesta =  0;  // fallo
        }
        else
                respueta = 1;  // exito

        // pasar respuesta a formato de red
        respuesta = htonl(respuesta);
        if (send(s, (char*)&respuesta, sizeof(int)) < 0) {
                return(-1);
        }

        // obtiene el tamaño del archivo y lo envía

        tamanyo = lseek(df, 0, SEEK_END);
        // pasar a formato de red
        tamayo = htonl(tamanyo);

        if (send, (char*)&tamanyo, sizeof(int)) < 0){
                return(-1);
        }
        lseek (df, 0, SEEK_SET); // poner el puntero del fichero de nuevo al principio

        while (totales < tamanyo){

                if (tamanyo – totales >= MAX_BLOQUE)
                        num=read(df, buffer, MAX_BLOQUE);
                else
                        num=read(df, buffer, tamanyo-totales);


                if (num < 0){
```

```
                        return(-1);
                }

                totales=totales+num;
                // envía el bloque al cliente
                if (enviar(s, (char*)&buffer, num) < 0) {
                        return(-1);
                }

        }
        close(df);
        return (0);
}

int almacenar_imagen(int s)
{
        int size;
        char nombre[MAX_NOMBRE];
        char buffer[MAX_BLOQUE]
        int df;
        int respuesta;
        int num=0, totales=0, tamanyo=0;

        // recibe el tamaño del nombre del archivo
        if (receive(s, &size, sizeof(size)) < 0){
                return(-1);
        }

        size = ntohl(size);  // pasar a formato de host
        // recibe el nombre del archivo
        if (receive(s, &nombre, size) < 0) {
                return(-1);
        }


        df=open(nombre, O_CREAT|O_WRONLY|O_TRUNC, 0777);
        if (df<0)
                respuesta = 0;   // error
        else
                respuesta = 1;

        // envía la respuesta
        respuesta = htonl(respuesta); // pasar a formato de red
        if (send(s, (char*)&respuesta, sizeof(int)) < 0){
                return(-1);

        }

        if (respuesta == 0 ){
                return (-1);
        }

        // recibe el tamaño del archivo

        if (receive(s, (char*)&tamanyo, sizeof(int)) < 0){
                return(-1);
        }
        tamanyo = ntohl(tamanyo);  // pasar a formato de host


        while (totales < tamanyo){

                if (tamanyo - totales >= MAX_BLOQUE)
                        num=recibir(s, (char*)&buffer, MAX_BLOQUE);
```

```
                else
                        num=recibir(s, (char*)&buffer, tamanyo – totales);

                if (num < 0) {
                        return(-1);
                }

                if (write(df, buffer, num) < 0){
                        return(-1);
                }

                totales = totales +  num;
        }
        close(df);
        return(0);
}
```