Ejercicios propuestos de sincronización entre procesos y concurrencia

Ejercicio 1. Escriba un programa que cree 100 procesos ligeros, creados como independientes, es decir, cuando acaban su ejecución liberan sus recursos y nadie puede esperar por ellos. Una vez creados todos los procesos se deberá esperar a la terminación de todos ellos.

Ejercicio 2. Escriba el código de dos procesos ligeros que deben alternar de forma estricta su ejecución. Resuelva el problema utilizando semáforos.

Ejercicio 3. Resuelva el ejercicio anterior utilizando mutex y variables condicionales.

Ejercicio 4. Se desea desarrollar una aplicación que debe realizar dos tareas que se pueden ejecutar de forma independiente. Los códigos de estas dos tareas se encuentran definidos en dos funciones cuyos prototipos en lenguaje de programación C, son los siguientes:

```
void tarea_A(void);
void tarea_B(void);
```

Se pide:

a) Programar la aplicación anterior utilizando tres modelos distintos: un programa secuencial ejecutado por un único proceso, un programa que crea procesos para desarrollar cada una de las tareas, y un programa que realiza las tareas anteriores utilizando procesos ligeros. En cualquiera de los tres casos la aplicación debe terminar cuando todas las tareas hayan acabado.

Ejercicio 5. Se desea usar un semáforo para asegurar que el inicio de una determinada actividad del proceso P2 comienza después de que finalice una actividad del proceso P1. ¿Qué primitiva de semáforos debe usar cada proceso y cuál debe ser el valor inicial del semáforo?

Ejercicio 6. Resuelva el ejercicio anterior utilizando tuberías.

Ejercicio 7. Resuelva el ejercicio anterior utilizando mutex y variables condicionales.

Ejercicio 8. Se desea comparar el rendimiento de un servidor de ficheros secuencial con uno multiflujo que utiliza procesos ligeros (threads). En ambos casos el servicio de una petición implica la ejecución con 10 ms de tiempo de cómputo y en el caso de que los datos pedidos no estén en la cache de bloques del servidor, otros 40 ms de acceso al disco. En el servidor secuencial, durante el acceso al disco se bloquea el proceso servidor. Sin embargo, en el segundo caso sólo se bloquea el thread correspondiente. Teniendo en cuenta que el disco sólo puede llevar a cabo una operación en cada momento y suponiendo que el servidor se ejecuta en un sistema monoprocesador se pide:

a) Suponiendo que no se producen aciertos en la cache, ¿Cuántas peticiones por segundo puede procesar el servidor secuencial? ¿y un servidor multiflujo?
b) Igual que el anterior apartado pero con un 100% de aciertos en la cache de bloques.
c) Igual para un 50% de aciertos

d) Considerar cómo afectaría a los resultados de los tres apartados anteriores el uso de un multiprocesador con 4 procesadores para ejecutar el servidor.

Ejercicio 9. Se dispone de dos procesos ligeros para la ejecución de un programa para escuchar música a través de *streaming*:

1. Un proceso ligero **A** que lee datos de un dispositivo de red y almacena los datos en un *buffer* compartido.
2. Un proceso ligero **B** que toma los datos de un *buffer* compartido y escribe esto en un dispositivo de audio.

Ambos procesos utilizan dos primitivas para acceder a los datos:

- cantidad = Leer (dispositivo)
  - En caso de que cantidad sea igual a 0, eso significa que no hay más datos.
- Escribir (dispositivo, cantidad)

  Siendo el parámetro: red, audio, o buffer

Codifique los procesos ligeros A y B, e indique que estructuras de datos son necesarias teniendo en cuenta los siguientes supuestos:

- La aplicación parte con el *buffer* vacio
- Cuando el *buffer* contiene datos, el proceso A debe avisar al proceso B de que existen datos suficientes en el buffer.
- Si el *buffer* se encuentra lleno al 100 %, el proceso A para automáticamente.
- Si el *buffer* se encuentra vacio, el proceso B debe esperar a que existan datos.
- Si no hay más datos que leer del dispositivo de red, el proceso A debe avisar al proceso B de que no hay más datos y que debe terminar cuando los datos del *buffer* sean procesados.

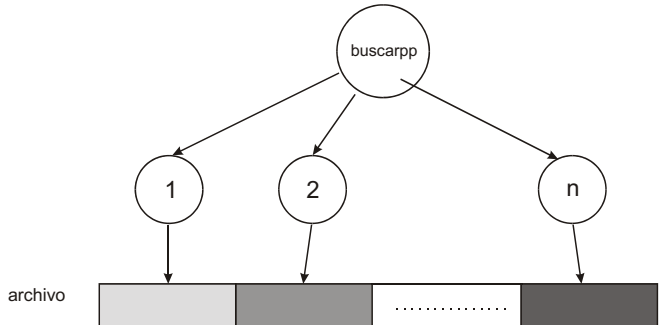
Para la resolución del problema utilice mutex y variables condicionales.

Ejercicio 10. Se desea desarrollar un programa en C, utilizando los servicios del estándar POSIX, que busque cuantas veces aparece un determinado carácter en un archivo. La sintaxis de este programa será la siguiente:

```
buscar  c file
```

donde `c` es el carácter a buscar y `file` el nombre del archivo. El programa debe imprimir el número de veces que aparece el carácter `c` en el fichero. En caso de que el carácter no se encuentre en el archivo el programa dará como resultado 0. El programa deberá realizar un correcto tratamiento de errores.
La versión a desarrollar debe utilizar procesos ligeros, de forma que la búsqueda en el fichero se haga en paralelo (ver figura adjunta). Este programa se encargará de:

1. Crear los procesos hijos encargados de realizar la búsqueda. Cada uno de estos procesos realizará la búsqueda en una porción del archivo (tal y como se muestra en la figura). Haga una versión para n= 4 procesos ligeros.
2. El programa debe finalizar cuando todos los procesos hayan acabado de procesar su parte. Utilice para el desarrollo de este programa procesos ligeros, mutex y variables condicionales.



Ejercicio 11. A continuación se muestra el código de un programa (pi.c) que calcula el número pi mediante el cálculo de la siguiente integral definida con el método de los trapecios:

$$\int_0^1 \sqrt{4\left(1-x^2\right)}\,dx = \frac{\pi}{2}$$

pi.c:

```
#include <stdio.h>
#include <math.h>
#include <stdio.h>
#include <sys/time.h>

#define N 1E7
#define d 1E-7

double PI;


void calcula(void){
    double result=0.0, x;
    int i;

    result = 0;
    for (i=0; i<N; i++) {
        x = d*i;
        result+=sqrt(4*(1-x*x));
    }

    PI = PI + d*2*result;
    return;

}

int main (int argc, char* argv[])
{
    int i;
    long t;
    struct timeval t1, t2;

    gettimeofday(&t1, 0);
    calcula();
    gettimeofday(&t2, 0);

    printf("PI = %.9f\n", PI);
```

```
    t = (t2.tv_sec-t1.tv_sec)*1000000 + t2.tv_usec-t1.tv_usec;
    printf("El tiempo es %f ms\n", (t / 1000.0));

    return 0;
}
```

Modifique el código del programa anterior para que el cálculo sea hecho por una serie de threads en paralelo. El programa permite obtener el tiempo de ejecución. Compruebe cómo es el tiempo de ejecución de la versión secuencial respecto a la versión paralela.

Ejercicios propuestos de paso de mensajes

Ejercicio 1. Desarrollar un servidor que permita obtener la hora, la fecha y el día de la semana en la que cae un día determinado. Diseñar y desarrollar el cliente y el servidor en los dos siguientes supuestos:

a) Se dispone de un sistema con los siguientes servicios:
   - int **Connect**(int pid): este servicio establece una conexión con un proceso con identificador pid. Devuelve un identificador de conexión.
   - int **Accept** (): este servicio acepta una conexión de un proceso que ejecute el servicio Connect. Devuelve un identificador de conexión.
   - **Send**(int ic, char *mensaje, int long): este servicio envía un mensaje de una determinada longitud a través del identificador de conexión ic.
   - **Receive(**int ic, char *mensaje, int long): este servicio recibe un mensaje de una determinada longitud de la conexión con identificador ic. pid.

   Asuma que los identificadores de los procesos son números enteros y que el servidor viene identificado por el número 1000.

b) Considere un sistema que utiliza colas de mensajes POSIX.

Ejercicio 2. Se desea diseñar un modelo de vector distribuido. Sobre un vector distribuido se definen los siguientes servicios:

- int **init** (char *nombre, int N). Este servicio permite inicializar un array distribuido de N números enteros. La función devuelve 1 cuando el array se se ha creado por primera vez. En caso de que el array ya esté creado, la función devuelve 0. La función devuelve -1 en caso de error.
- int **set**(char *nombre, int i, int valor). Este servicio inserta el valor en la posición i del array nombre.
- int **get**(char *nombre, int i, int *valor). Este servicio permite recuperar el valor del elemento i del array nombre.

Considere un sistema que utiliza colas de mensajes POSIX. Diseñe un sistema que implemente este servicio de vectores distribuidos

Ejercicio 3. Considere un sistema con los siguientes servicios:

- **Send**(int pid, char *mensaje, int long): este servicio envía un mensaje de una determinada longitud al proceso con identificador pid.
- **Receive(**int pid, char *mensaje, int long): este servicio recibe un mensaje de una determinada longitud.

Dados dos procesos con identificadores P1 y P2, implemente el código del procesos P1 y el del procesos P2. El proceso P1 se encarga de leer un determinado archivo y envía el contenido de este archivo al proceso P2. El procesos P2 se encarga de imprimir el archivo por pantalla. Considere que el tamaño máximo del mensaje que se puede utilizar en los servicios anteriores es de 8 KB. Por su parte, el archivo puede tener cualquier tamaño.

Ejercicio 4. Considere un sistema compuesto por N procesos. En este sistema se dispone de un mecanismo de comunicación basado en colas de mensajes. Sobre este sistema se quiere desarrollar un modelo de comunicación multicast. En este sistema cada proceso lleva asociado una cola de mensajes cuyo nombre coincide con el identificador del proceso asociado. Diseñe un mecanismo que permita que un proceso pueda mediante un servicio único que se denominará msend(), enviar un mensaje a todas las colas de mensajes del resto de procesos. Es decir, cuando un proceso invoca msend(buf, tamaño), el mensaje buf, es enviado a todas las colas de mensajes del resto de procesos.

Ejercicio 5. Considere un sistema compuesto por N procesos que no comparten memoria. Se quiere resolver el problema de la sección crítica utilizando una cola de mensajes al estilo de la de UNIX. Diseñe el sistema, indicando qué cola o colas hacen falta y cómo se resuelve el problema de la sección crítica para cada procesos.

Ejercicio 6. Se desea construir un programa distribuido formado por N procesos. Cada uno de estos procesos tendrá asociado un identificador de proceso, un número entero comprendido entre 0 y N-1. Cada proceso cuando comienza su ejecución recibe este identificador, que puede consultar en la variable PID. Asimismo puede conocer el número total de procesos (N) consultando la variable NP. Se asegura que no hay dos procesos con el mismo identificador. Se desea que estos procesos tengan acceso a los siguientes servicios:

- int **init**(void) Este servicio deben ejecutarlos todos los procesos al comienzo de su ejecución. Esta operación es imprescindible para utilizar el resto de servicios.
- int **send**(int n, char *buf, int len); Este servicio envía al proceso con identificador n (comprendido entre 0 y N-1) un mensaje (buf) de longitud len. El envío no es bloqueante.
- int **recv**(int n, char *buf, int len); Este servicio recibe del proceso n un mensaje en buf de longitud len. La recepción es bloqueante.
- int **broadcast**(int root, char *buf, int len); Con este servicio el proceso con identificador root, envía el mensaje buf de longitud len a todos los procesos del programa distribuido excepto al que ejecuta la llamada. Todos los procesos deben ejecutar la llamada.
- int **barrier**(void);Esta llamada bloquea a un proceso hasta que todos los procesos del programa distribuido la hayan ejecutado, es decir, la deben ejecutar todos los procesos para que puedan continuar su ejecución.

Se desea implementar estas funciones sobre un sistema que ofrece servicios de paso de mensajes basados en puertos . En este sistema cada proceso lleva asociado una dirección que es única en todo el sistema y que viene especificada por un número entero. Esta dirección se encuentra asociada al proceso en el momento de comenzar su ejecución. Los servicios disponibles en este sistema son los siguientes:

- int **obtener_direccion**(void); Este servicio permite a un proceso obtener su dirección que se asegura que es única en todo el sistema y que recibe en el momento de comenzar su ejecución.
- int **publicar_dirección**(int direccion, int n); Este servicio permite publicar en un servicio de nombres un valor asociado a una dirección.
- int **buscar_direccion**(int n); Este servicio devuelve la dirección que tiene asociada el servicio de nombres con el valor n.
- int **connect**(int direccion); Este servicio establece una conexión con el proceso que ejecuta en la dirección pasada como argumento. La llamada devuelve un descriptor de comunicación que se puede utilizar en las operaciones de envío y recepción.
- int **accpet**(int direccion); Este servicio bloquea al proceso que lo ejecute hasta que el proceso que ejecuta en la dirección pasada como argumento haga un connect. La llamada devuelve un descriptor de comunicación que se puede utilizar en las operaciones de envío y recepción.
- int **send_mess**(int fd, char *p, int len); Esta operación envía un mensaje por el canal con descriptor fd. El envío es no bloqueante.
- int **recv_mess**(int fd, char *p, int len); Esta operación recibe un mensaje por el canal con descriptor fd. La recepción es bloqueante.

Para que un par de procesos pueda conectarse en este sistema uno debe hacer un connect y el otro un accept.
Se pide:

1. Crear una estructura de información que permita asociar los identificadores de procesos (0 a N-1) con los descriptores de comunicación. Con esta estructura un proceso podrá conocer el descriptor de comunicación a utilizar para enviar o recibir mensajes de un proceso con identificador j (comprendido entre 0 y N-1).

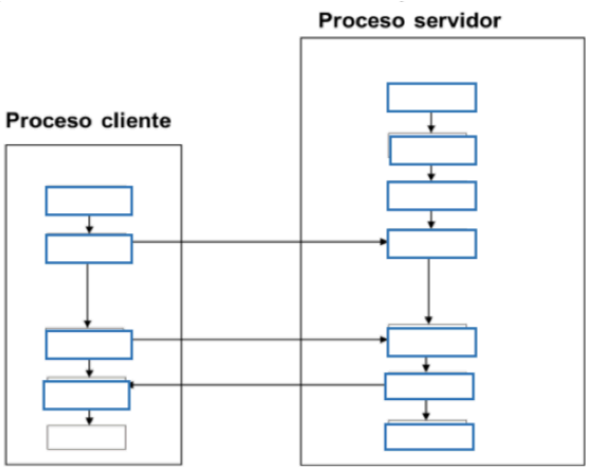
2. Implementar la operación init. Esta operación conectará a todos los procesos con todos. Cada par de procesos estará conectado mediante una única conexión que se podrá utilizar para enviar y recibir mensajes. Consejo: para cada par de procesos a considerar uno ejecutará connect y el otro accpet. Esta operación rellenará la estructura de información anterior. Implemente el resto de operaciones: send_p, recv_p, broacast y barrier.

Ejercicio 7 Se pretende implementar una barrera distribuida con las siguientes características. Un conjunto de N procesos cliente, que no comparten memoria entre sí, quieren utilizar una barrera de sincronización ofrecida por un proceso servidor. Los clientes iteran F fases cada una de ellas con M iteraciones (al igual que en el ejercicio propuesto 1). En cada iteración, cada proceso duerme un número aleatorio de segundos. Al final de cada fase, cada cliente se bloquea esperando la finalización del final de la fase en todos los clientes. Para ello cada cliente envía al servidor una petición indicándole que quiere ejecutar la barrera de sincronización. Por su parte, el servidor implementa la barrera, que simplemente desbloqueará a todos los clientes bloqueados cuando reciba la petición del último de los procesos clientes.
Se pide:

a) Diseñar la aplicación cliente-servidor anterior utilizando colas de mensajes POSIX, indicando y especificando las cuestiones de diseño necesarias.
b) De acuerdo al diseño elaborado, implemente en C, el código del cliente y del servidor. No es necesario hacer una implementación perfectamente detallada del código.

Ejercicios propuestos de aplicaciones clientes servidor y sockets

Ejercicio 1. Dado el siguiente esquema de comunicación entre un cliente y un servidor:



Se pide:

a) Indique sobre el dibujo anterior los servicios que faltan.
b) La estructura anterior representa a un servidor secuencial. Convierta el esquema anterior en otro en el que el servidor sea concurrente.
c) Si el cliente ejecuta en la dirección IP 163.117.148.200 y el servidor en la dirección 163.117.148.10 y puerto 80, ¿cuándo se asigna un puerto al cliente? ¿qué puerto se le asigna?

Ejercicio 2. Desarrolle un servidor que permita obtener la hora, la fecha y el día de la semana en la que cae un día determinado. Diseñar y desarrollar el cliente y el servidor.

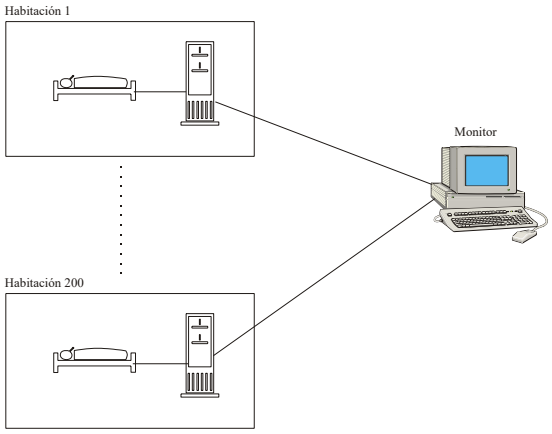
Ejercicio 3. Diseñe utilizando sockets el mecanismo de comunicación de las colas de mensajes POSIX.

Ejercicio 4. Dado el siguiente fragmento de código, indique qué problema(s) plantea. Considere que s representa un descriptor de socket correctamente definido.

```
struct mensaje {
     int a;
     float b;
     char c;
};

int n;
struct mensaje m;

m.a = 4;
m.b = 2.3;
m.c = '1';

n = write(s, (const void *) &m, sizeof(struct mensaje));
if (n != sizeof(struct mensaje)) {
     printf("Error en el envío del mensaje\n");
```

```
            close(s);
        }
```

Ejercicio 5. Un hospital, con 200 habitaciones, desea construir una aplicación que le permita monitorizar las constantes vitales de los pacientes. Para ello dispone de un sistema como el que se muestra en la siguiente figura.



Cada paciente está conectado a un computador que toma cada 500 ms la temperatura, las pulsaciones y la presión arterial de dicho paciente. Se dispone de un computador que hace de monitor y que visualiza las constantes vitales de todos los pacientes. El funcionamiento de la aplicación es el siguiente: cuando llega un paciente, el computador al que se conecta el paciente envía un mensaje al monitor indicándole que va a comenzar el proceso de monitorización. Cada 500 ms envía al monitor las constantes vitales del paciente. Cuando el paciente abandona la habitación, el computador al que estaba conectado envía un mensaje al monitor indicándole que deja de enviarle datos. Un requisito importante en esta aplicación es que el servidor debe recibir cada 500 ms los datos de un paciente con una alta fiabilidad.

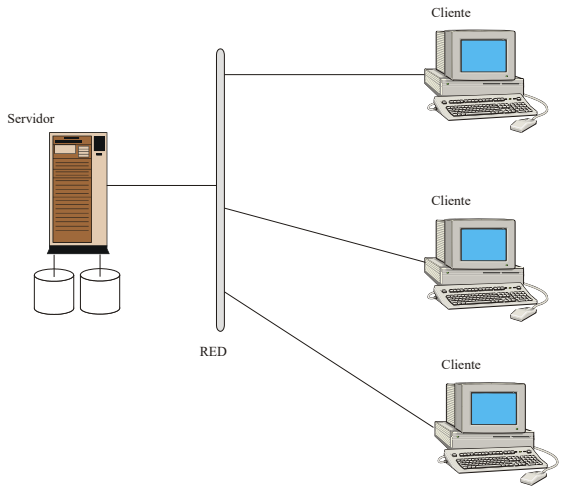
Haga un diseño de la aplicación, haciendo todas las consideraciones que crea oportuna.

Suponga que se desea emplear RPC para esta aplicación con las siguientes características:

- La sobrecarga por llamada es de 0,1 ms, tanto en el cliente como en el servidor
- Se emplea una red Ethernet de 10 Mbit/s
- La latencia de las operaciones, tanto de envío como de recepción de datos, es de 1ms.
- Los computadores de las 200 habitaciones ya han realizado el proceso de conexión y se encuentran en la fase de envío de datos al monitor.

g) Indique los pasos que debe realizar el suplente del cliente.
h) ¿Cuál es tamaño máximo de la información por llamada, que se puede intercambiar entre el cliente y el servidor para que se pueda monitorizar correctamente a todos los pacientes del hospital cada 500 ms?

Ejercicio 6. Se desea diseñar un sistema de video bajo demanda como el que se muestra en la siguiente figura:



En un sistema de este tipo el servidor debe enviar las películas a los clientes de forma que éstos las visualicen con una cierta calidad de servicio, sin que se aprecien pérdidas significativas.
El funcionamiento del sistema es el siguiente:

1. El servidor almacena en los discos una serie de películas.
2. Cuando un cliente desea ver una película envía un mensaje al servidor indicando el nombre de la película que desea ver.
3. El servidor envía un mensaje al cliente indicando si puede o no ver la película. El cliente no podrá ver la película si existen muchos clientes en el sistema y no le puede ofrecer la película con calidad.
4. Si el cliente recibe un mensaje de confirmación, se quedará a la espera de recibir la película.
5. El servidor envía de forma periódica a todos los clientes activos paquetes con fragmentos de la película. El tamaño de los paquetes es variable, pero nunca mayor de 8 KB.

En este sistema, el servidor envía a cada uno de los clientes activos un flujo continuo de paquetes que deben mostrarse por pantalla sin que se produzcan pérdidas significativas en las imágenes. Es importante que el sistema sea rápido para garantizar un flujo continuo a los clientes, sin embargo, los clientes pueden tolerar la pérdida de algunos paquetes sin que se produzca una degradación en la visualización de la película.

Suponiendo que se desea construir una aplicación cliente-servidor, se pide:

a) ¿Qué tipo de sockets emplearía para esta aplicación? ¿Por qué?
b) Haga un diseño de esta aplicación, especificando el formato de los mensajes (a intercambiar entre el cliente y el servidor) que se emplearía, así como la secuencia de mensajes a

intercambiar entre el cliente y el servidor. Haga una estimación del tamaño que ocuparían los mensajes.

c) De acuerdo al tipo de sockets empleado en el apartado *a*, indique qué llamadas de la biblioteca de sockets utilizaría en el cliente y en el servidor, así como el orden en el que usarían dichas llamadas los clientes y el servidor.

d) ¿Cómo podría indicar el servidor al cliente el final de una película?

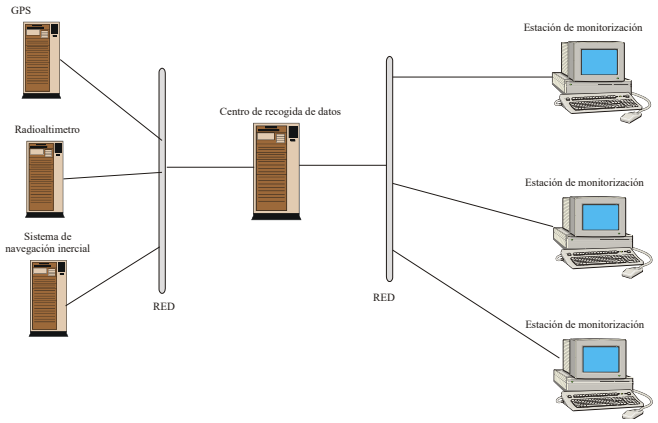
Suponga que se dispone de un sistema con las siguientes características:

- Se emplea una red Fast-Ethernet de 100 Mbit/s
- El tiempo medio de acceso a disco para un bloque de 8 KB es de 7ms

e) ¿Cuál es el número máximo de clientes que pueden ver las películas considerando que el ancho de banda óptimo que necesita cada uno de los clientes para ver una película es de 320 KBit/s.

Ejercicio 7. Se desea desarrollar un sistema de adquisición de datos para los diferentes instrumentos de un avión. Para ello se pretende realizar un primer prototipo que incluya la adquisición de datos de tres instrumentos: el sistema de navegación inercial, el radioaltímetro y el GPS (véase la figura). El sistema de navegación inercial envía un paquete de datos formado por tres datos de tipo entero y tres de tipo real. El radioaltímetro envía un paquete formado por 10 datos de tipo real. El GPS envía un paquete de datos formado por cinco datos de tipo real y un dato de tipo booleano. Cada instrumento genera un nuevo paquete cada 100 ms.

La siguiente figura ilustra la arquitectura final del sistema. Cada **instrumento** se puede considerar un computador en sí mismo. El **centro de recogida de datos** es un computador que recoge los datos enviados por los diferentes instrumentos (recuerde que cada instrumento genera un nuevo paquete de datos cada 100 ms). Las **estaciones de monitorización** son computadores en las que se pueden visualizar los datos que generan los diferentes instrumentos. En cada estación de monitorización existe un usuario que puede visualizar los datos de uno, de dos o de todos los instrumentos. Cada usuario en cada estación de monitorización puede visualizar los datos de los diferentes instrumentos a una tasa distinta. Por ejemplo, en la estación nº 1 se puede visualizar los datos del GPS cada 500 ms y los datos del radioaltímetro cada segundo. Por su parte en la estación nº 2 se pueden visualizar los datos del GPS cada 2 segundos y los datos del sistema de navegación inercial cada 300 ms.



Para el desarrollo de esta aplicación se propone el diseño de dos soluciones distintas:

1. En el primer modelo, los instrumentos envían los datos al centro de recogida de datos a su correspondiente tasa de envío. Por su parte, las diferentes estaciones de monitorización recogen del centro de recogida de datos los datos que deseen visualizar según la tasa definida.
2. En el segundo modelo, los instrumentos envían los datos al centro de recogida de datos a su correspondiente tasa de envío. En este segundo modelo, el comportamiento de las estaciones de monitorización es diferente. Cuando una estación desea monitorizar un determinado instrumento, envía una petición al centro de recogida de datos para indicarle que quiere visualizar los datos de ese instrumento a una determinada tasa. El servidor registra esta información y se encarga posteriormente de enviar los datos directamente a la estación de monitorización para su visualización. Cuando una estación de monitorización no desea recibir más información de un instrumento envía un mensaje al centro de recogida de datos para indicarlo.

Se pide, para cada uno de los modelos anteriores:

a) ¿Qué tipo de sockets emplearía? ¿Por qué?
b) Haga un diseño de la aplicación, indicando qué componentes (cliente, servidor) residen en cada computador.
c) Especifique el formato de los mensajes (a intercambiar entre los distintos elementos del sistema) que se emplearía, así como la secuencia de mensajes a intercambiar entre los distintos elementos del sistema. Haga una estimación del tamaño que ocuparían los mensajes.
d) De acuerdo al tipo de sockets empleado en el apartado *a*, indique qué llamadas de la biblioteca de sockets utilizaría en los distintos elementos de su aplicación.
e) ¿Qué solución cree que sería la mejor? ¿Por qué?

Ejercicio 8. Se desea desarrollar un sistema de monitorización de los nodos de un cluster. En cada nodo del cluster ejecuta un servicio de monitorización, que acepta dos operaciones enviadas desde un nodo central que se utiliza como monitor:

- *start*: que inicia el proceso de monitorización en el nodo.
- *stop*: que para el proceso de monitorización en el nodo.

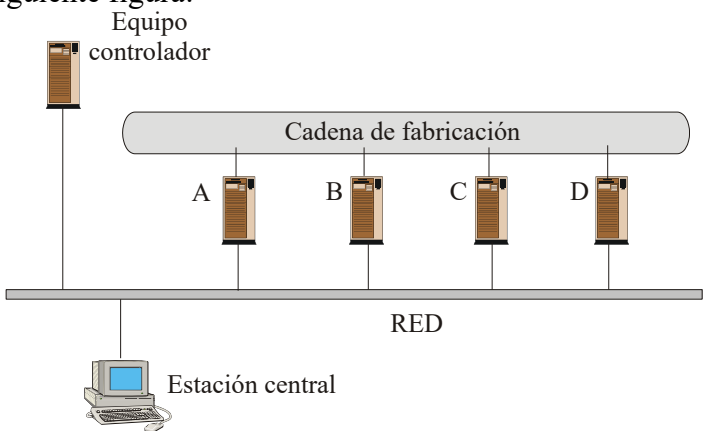
Una vez arrancado el proceso de monitorización en un nodo, este enviará cada segundo al monitor central la siguiente información: número de procesos arrancados en la máquina, porcentaje de memoria libre, temperatura del procesador, operaciones de E/S realizadas en el último segundo. Cada vez que el monitor central recibe información de un nodo registra dicha información en una base de datos utilizando un servicio denominado registrar, que tiene la siguiente interfaz:

- Registrar(char *IP, int num_procesos, float mem_libre, float temp, int iop)

Donde IP representa la IP del nodo del cluster, num_procesos el número de procesos arrancados en la máquina, mem_libre el porcentaje de memoria libre en el nodo, temp la temperatura del procesador e iop el número de operaciones de E/S realizadas en el último segundo.
Considerando que se quiere desarrollar la aplicación utilizando sockets, se pide:

a) Haga un diseño de la aplicación, haciendo todas las consideraciones que crea oportunas.

Ejercicio 9. Se quiere diseñar una aplicación distribuida que controle una cadena de fabricación como la que se muestra en la siguiente figura:



Esta cadena consta de los siguientes componentes:

a) **Estaciones de supervisión** de la cadena (equipos A, B, C y D de la figura anterior). Se encargan de comprobar el correcto funcionamiento de la cadena en el punto en el que están instalados.
b) Un **equipo controlador** que se encarga de poner en marcha la cadena, pararla y activar el sistema antiincendios.
c) Una **estación central** en la que se puede monitorizar el estado de la cadena. Desde esta estación se pone en marcha la cadena y también se para.
d) Una **red de área local** que conecta a todos los equipos anteriores.

El funcionamiento del sistema es el siguiente:

a) Cuando el sistema comienza a funcionar, la estación central indica al equipo controlador que ponga en marcha la cadena.
b) Durante el funcionamiento de la cadena de fabricación, la estación central interroga cada minuto a las estaciones de supervisión sobre su estado. El estado de una estación de supervisión puede ser:
    1. CORRECTO, si la cadena funciona correctamente.
    2. INCORRECTO, si hay algún error en la cadena.
    3. FUEGO, si la estación supervisora detecta fuego.
c) Con independencia de la monitorización realizada cada minuto por la estación central, si en algún momento una de las estaciones supervisoras detecta un funcionamiento incorrecto o un fuego, notifica a la estación central dicho problema. En este caso la estación supervisora no espera a que la estación central le pregunte su estado.

d) Cuando se detecta un funcionamiento incorrecto o un fuego, la estación central indica al equipo controlador dicho evento. En caso de que el funcionamiento sea incorrecto, el equipo controlador para la cadena. En caso de fuego activa el sistema antiincendios.
e) Cuando el sistema deja de funcionar, la estación central indica al equipo controlador que pare la cadena.

La aplicación se va a realizar utilizando sockets. Para realizar un correcto diseño de la misma debe responder a las siguientes preguntas:

1. ¿Qué tipo de sockets emplearía? ¿Por qué?
2. Identifique todos los mensajes del sistema, indicando: el formato del mismo, su tamaño en bytes, quién genera el mensaje y quién lo recibe y procesa
3. Indique la estructura en pseudocódigo que tendrían los procesos que ejecutan en los distintos componentes del sistema, indicando las llamadas de la biblioteca de sockets que hay que utilizar en los distintos elementos de su aplicación.

Ejercicio 10. Se quiere diseñar una aplicación que permita ejecutar comandos en máquinas remotas. Esta aplicación se ejecutará de la siguiente forma:

```
remoteshell dirección-IP-remota
```

Al programa se le pasará la dirección IP de la máquina remota en la que se quieren ejecutar los comandos. El programa a continuación mostrará al usuario un *prompt* de la siguiente forma:

>

El usuario tecleará comandos de UNIX, como por ejemplo, “ls –l”, “ps”, “who”. Estos comandos se ejecutarán en la máquina remota y la máquina remota devolverá el resultado de la ejecución de estos comandos al programa remoteshell que mostrará la salida por pantalla. El programa finalizará su ejecución cuando se pulse fin de archivo (Control-D en Unix/Linux).

La aplicación se va a realizar utilizando sockets. Para realizar un correcto diseño de la misma debe responder a las siguientes preguntas:

1. ¿Qué tipo de sockets emplearía? ¿Por qué?
2. Identifique la parte cliente y servidora de esta aplicación.
3. Identifique todos los mensajes del sistema, indicando: el formato del mismo, su tamaño en bytes, quién genera el mensaje y quién lo recibe y procesa
4. Indique la estructura en pseudocódigo que tendrían los procesos que ejecutan en la parte cliente y servidora, indicando las llamadas de la biblioteca de sockets que hay que utilizar en los distintos elementos de su aplicación.

NOTA: Para la ejecución de comandos en la máquina remota se puede utilizar la llamada system que permite ejecutar un comando desde un programa. Una posibilidad que se puede utilizar en la máquina remota es que la salida del comando se vuelque a un fichero y que el contenido de este fichero se envíe al programa remoteshell para que lo muestre por pantalla.

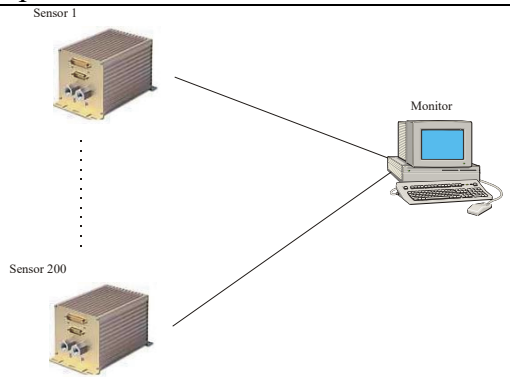
Ejercicio 11. Se dispone de un parking con tres puertas de acceso y dos máquinas para que los usuarios del parking realicen el pago. Cada puerta permite la entrada y salida de vehículos y está dotada de un pulsador y un expendedor de tickets. Además dispone de un letrero luminoso donde se puede escribir: *libre*, *completo* o *cerrado*. Se quiere diseñar un sistema informático distribuido para controlar dicho parking. Para ello se instala un computador central (CC) que va a controlar el sistema completo y equipos sencillos en las puertas y en las máquinas donde se realiza el pago. El funcionamiento del sistema es el siguiente:

- Cuando se abre el parking se reinicia el sistema y el CC envía un mensaje a los computadores de las puertas para que escriban *libre* en el letrero luminoso de entrada.
- Cuando el parking se cierra el CC envía el mensaje *cerrado* a todas las puertas.
- Cuando un vehículo llega a una puerta el conductor presiona el pulsador. Si el parking está abierto y no está completo, el controlador envía al computador central un mensaje que indica que un vehículo quiere entrar. Si el parking está cerrado o completo, el controlador no envía ninguna información al equipo central. Cuando el CC recibe un mensaje de entrada incrementa el contador de plazas ocupadas y responde al computador de la puerta adecuada con un mensaje indicando la hora y fecha de entrada, un código (un número entero) que identifica al vehículo y el nuevo estado del parking (*libre* o *completo*). . Con la fecha, hora y código, el computador de la puerta imprime el ticket. Si el CC detecta que el parking pasa a estar completo, envía un mensaje al resto de puertas indicando que el parking pasa a estar *completo*. En caso de que el vehículo no pueda entrar debido a que se ha llenado ya el parking el CC responde con el mensaje completo e indica que el vehículo no puede pasar (de esta forma el computador de la puerta no imprime el ticket de entrada)
- Cuando un conductor realiza el pago en una de las máquinas introduce el ticket. Con la información codificada en el ticket, el computador de la máquina envía un mensaje al CC indicando la hora, fecha y código del vehículo a retirar. El CC realiza el cálculo del importe a pagar y se lo devuelve a la máquina de pago. La máquina de pago indica el precio al conductor, cobra e imprime un ticket para que se proceda a la retirada del vehículo.
- Cuando un conductor sale del parking introduce el ticket de salida en el computador de la puerta y éste envía un mensaje al CC indicando la retirada del vehículo. Si el parking pasa a estar *libre* se informa a todas las puertas de dicho evento.

Se pide:

1. ¿Qué tipo de sockets emplearía? ¿Por qué?
2. Identifique todos los mensajes del sistema, indicando: el formato del mismo, su tamaño en bytes, quién genera el mensaje y quién lo recibe y procesa.
3. Indique la estructura en pseudocódigo que tendrían los procesos que ejecutan en los distintos componentes del sistema, indicando las llamadas de la biblioteca de sockets que hay que utilizar en los distintos elementos de su aplicación.

Ejercicio 12. Un edificio, con 200 despachos, desea construir una aplicación que le permita monitorizar la temperatura y humedad de cada uno de los despachos.



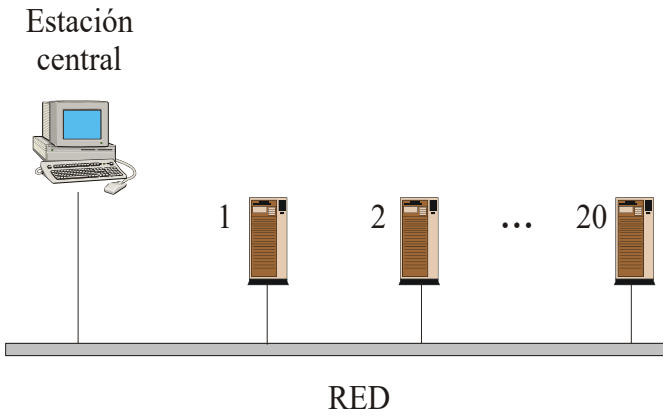
Cada despacho consta de un sensor que toma cada segundo la temperatura y humedad del despacho. Se dispone de un computador que hace de monitor y que permite visualizar los datos de todos los despachos. El funcionamiento de la aplicación es el siguiente: cuando un despacho se abre por las mañanas, el sensor situado en ese despacho envía un mensaje al monitor indicándole que va a comenzar el proceso de monitorización. Cada segundo envía al monitor los datos de temperatura y humedad del despacho. Cuando el despacho se cierra por las tardes, el sensor envía un mensaje al monitor indicándole que deja de enviarle datos.

Suponiendo que se desea construir una aplicación cliente-servidor utilizando sockets, se pide:

a) Identifique en qué computador reside el cliente y en cuál el servidor.
b) ¿Qué tipo de sockets emplearía para esta aplicación? ¿Por qué?
c) Especifique el formato de los mensajes a intercambiar entre el cliente y el servidor y la secuencia de mensajes a intercambiar entre ellos. Haga una estimación del tamaño que ocuparían los mensajes.

De acuerdo al tipo de sockets empleado en el apartado *b*, indique qué llamadas de la biblioteca de sockets utilizaría en el cliente y en el servidor .

Ejercicio 13. Se dispone de **20 máquinas cliente** y una máquina central, denominada **estación central** tal y como se muestra en la siguiente figura:



En cada una de las 20 máquinas (así como la estación central) se ejecuta un demonio que ofrece un servicio denominado servicio **daytime** que atiende a través del puerto 13. Para acceder a la información solo es necesario establecer una conexión con la máquina y puerto indicado, a lo que el servicio responde enviando cadena de caracteres terminada en el carácter de retorno de carro (‘\n’) y cierra la conexión. Como ejemplo, dicho servicio ofrecería la siguiente información cuando se es ejecutado:

23 JUN 2005 15:27:37 CEST

Donde se tiene la fecha 23 de junio de 2005, a las 15 horas, 27 minutos y 37 segundos tomando el sistema CEST como referencia.

También ejecuta un servicio denominado **setdaytime** que atiende por el puerto 14 y que permite cambiar la hora de la máquina. Para ello se establece la conexión, se envía una cadena de caracteres con exactamente el mismo formato que el utilizado en el servicio **daytime** y se cierra la conexión.

Se quiere diseñar una aplicación distribuida que se encargue de gestionar la configuración de 20 máquinas, en concreto de gestionar de forma precisa la fecha y hora de las mismas.

Dicha gestión se realiza desde la **estación central** con la aplicación pedida, de forma que realiza los siguientes pasos generales:

1. Se lee de un fichero las direcciones IP de las 20 máquinas.
2. Se solicita al servicio de cada una de las máquinas la fecha y hora.
3. Si la diferencia con la hora de referencia (la de la estación central) se desvía de 5 segundos, entonces se procede a establecer la hora de cada una de las máquinas afectadas.

La aplicación se va a realizar utilizando *sockets*. Para realizar un correcto diseño de la misma debe responder a las siguientes preguntas:

1. Identifique todos los mensajes del sistema, indicando quién genera el mensaje y quién lo recibe y procesa.

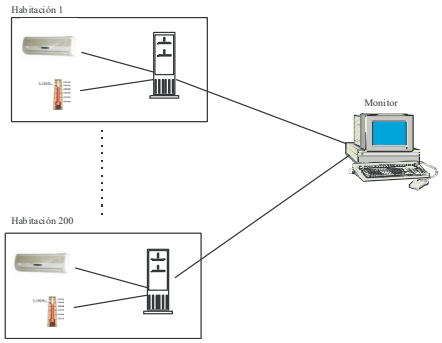
2. ¿Es posible diseñar dicho servicio con *sockets* no orientados a conexión? Proponga un ejemplo con el protocolo que seguiría para ofrecer el servicio con *sockets* no orientados a conexión.

3. Indique la estructura en pseudocódigo que tendrían los procesos que ejecutan en los distintos componentes del sistema, indicando las llamadas de la biblioteca de *sockets* que hay que utilizar en los distintos elementos de su aplicación.
   Realice este estudio en el caso de utilizarse *sockets* orientados a conexión y en el caso de utilizarse *sockets* no orientados a conexión.

4. ¿La aplicación puede ejecutarse en cualquiera de las 21 máquinas? ¿En caso negativo, qué sería necesario para que pudiera ejecutarse en cualquier máquina?

5. Si se desea que la aplicación pueda descubrir las direcciones IP de las 20 máquinas sin necesidad del fichero de configuración, ¿Qué servicio o servicios necesitaría? ¿Cómo funcionaría el sistema?

Ejercicio 14. Un hotel, con 200 habitaciones, desea construir una aplicación que le permita controlar la temperatura de sus habitaciones. Para ello dispone de un sistema como el que se muestra en la siguiente figura. Cada habitación dispone de un sensor de temperatura que está conectado a un computador que toma cada 500 ms la temperatura y humedad de dicha habitación. Se dispone de un computador que hace de monitor y que controla las medidas de las distintas habitaciones del hotel. El funcionamiento de la aplicación es el siguiente: cuando llega un cliente a la habitación, el computador de la habitación envía un mensaje al monitor indicándole que va a comenzar el proceso de muestreo y control. Cada 500 ms envía al monitor la temperatura y humedad. Cuando el cliente abandona la habitación, el computador envía un mensaje al monitor indicándole que deja de enviarle datos. La habitación dispone de un aparato de aire acondicionado que se enciende cuando el cliente entra en la habitación y que se apaga de forma automática cuando la abandona. Cuando el monitor detecta que la temperatura es superior a 25 grados lo enciende de forma remota. Cuando la temperatura cae por debajo de 22 grados lo apaga de forma remota



Se pide:

Haga un diseño de la aplicación utilizando *sockets*. Para ello:

1. Identifique los distintos tipos de mensajes.
2. Identifique el formato de dichos mensajes.
3. Indique el tipo de *sockets* a emplear.
4. Identifique el protocolo o secuencia de intercambio de mensajes necesario, indicando las partes servidoras y clientes

Ejercicio 15. Se dispone de una red de **N** ordenadores distribuida en las cuatro plantas de un edificio. Se desea implementar una aplicación cliente-servidor que consiste en monitorizar la temperatura y humedad del edificio usando los distintos ordenadores de la red como *servidores* de temperatura y humedad (0..N-1). Cada uno de los ordenadores tiene conectado un sensor de temperatura y de humedad. Para obtener el valor exacto de la temperatura y de la humedad en un punto, cada uno de los servidores usa una función *get_temperatura()* y *get_humedad()* respectivamente implementadas en una biblioteca de funciones externa. Todos los servidores escuchan en el puerto 2000. Por otro lado, existe un ordenador en el edificio que actúa como *cliente.* El programa cliente ejecuta un *shell de temperatura*

que recibe un mandato por cada función a implementar e invoca la función apropiada en el servidor. Las operaciones que se quieren implementar son:

- Solicitar el valor de temperatura del servidor X. El valor de retorno será el valor de temperatura devuelto por el servidor X.
- Solicitar el valor de humedad del servidor X. El valor de retorno será el valor de humedad devuelto por el servidor X.
- Solicitar el valor de temperatura de todos los servidores de la red. El valor de retorno será la media de los valores de temperatura recibidos.
- Solicitar el valor de humedad de todos los servidores de la red. El valor de retorno será la media de los valores de humedad recibidos.

Se pide:

1. Utilizando el lenguaje C y el mecanismo de comunicación de *sockets de TCP*, implemente el servidor del enunciado.
2. Utilizando el lenguaje C y el mecanismo de comunicación de *sockets de TCP*, implemente la función del programa cliente para solicitar el valor de temperatura a todos los servidores, cuyo prototipo es:

   float get_broadcast(char *fichero_maquinas);

   donde fichero_maquinas es un fichero de texto que contiene la dirección IP de las **N** máquinas de la red.

   NOTA:
   Considere que existe una función denominada: leer_maquina(char *fichero_maquinas); En cada invocación, devuelve la dirección IP de la siguiente máquina en el fichero.
3. Programe usando las RPC de SUN, la interfaz de comunicación entre cliente y servidor (temperatura.x).

Ejercicio 16. Una aplicación cliente genera una petición a un servidor web de tamaño 1 MB. Si los paquetes que se envían por la red tienen longitud máxima de 64 KB calcule:

- Número de paquetes que la aplicación cliente necesita enviar al servidor.
- Tiempo de transmisión de un mensaje asumiendo que el ancho de banda de la red es 10 Mbps y un retardo de 1 ms por paquete.
- Tiempo total para enviar la petición completa.

Ejercicio 17. La empresa Panelillos S.A. está especializada en el desarrollo e instalación de placas solares para grandes superficies. El CEO de la compañía quiere sacar al mercado una nueva solución para mejorar la eficiencia de absorción de rayos solares. Para ello, se quiere implementar un sistema distribuido que permita la gestión autónoma de las placas solares. El sistema estará compuesto por los siguientes elementos:

- Placas solares: cada placa solar dispone un computador autónomo que contiene un sensor que recoge una serie de datos ambientales: temperatura, humedad, luminosidad y niveles de radiación ultravioleta (estos valores se representarán como números en coma flotante en simple precisión), y un actuador que se encarga de mover la placa. El computador se encarga de leer los datos del sensor, descrito anteriormente, y enviará de forma periódica estos datos a una estación central. Para reducir costes se plantea desplegar una red inalámbrica para poder comunicar las placas y la estación central.

- Estación central: consistirá en un computador que recibirá los datos del computador de cada placa solar y monitorizará el estado de las placas solares con el objetivo de maximizar la energía generada. Por ello, el sistema contará con un sistema inteligente que moverá las placas solares en la orientación más adecuada. La estación central puede realizar dos tipos de llamadas sobre una placa solar dada:
  - Obtener_Energia: mediante esta llamada se obtendrá la energía generada por una determinada placa solar (valor entero).
  - Girar_Placa: mediante esta llamada se podrá mover una determinada placa sobre dos ángulos (coronal y sagital). Los ángulos vendrán dados por números enteros.

La estación central ajustará la orientación de la placa solar de forma periódica cada 5 minutos. Para obtener la nueva posición la estación hará uso de los datos ambientales recibidos de cada placa. La estancia central dispone de un método local, denominado Predice_Angulo, que recibe los parámetros ambientales y el valor de energía de una placa y devuelve los dos ángulos, que permiten girar la placa.

Considerando que se quiere desarrollar la aplicación utilizando sockets, se pide:

a) Haga un diseño detallado del sistema, haciendo todas las consideraciones que crea oportunas, teniendo en cuenta las características de cada uno de los componentes ofrecidos.
b) De acuerdo al protocolo diseñado, se pide implementar (código C) el siguiente servicio mediante el API de sockets: el código que permite desde el computador central realizar la invocación del método Girar_Placa sobre una determinada placa.

Ejercicio 18. Considere los siguientes fragmentos de programa cliente y servidor:

Cliente:

```
int main(int argc, char* argv[]) {
  int sockd;
  struct sockaddr_in serv_name;
  struct hostent *hp;
  int err;

  if (argc < 3) {
    fprintf(stderr, "Uso: %s Direccion_IP nombre_fichero\n", argv[0]);
    exit(1);
  }
   // en argv[1]  está la dirección IP del servidor
   // en argv[2]  está el nombre del fichero a transferir

  sockd = socket(AF_INET, SOCK_STREAM, 0); /* crea el socket */
  if (sockd == -1) {
    perror("Error en socket");
    exit(1);
  }

  /* direccion del servidor*/
  bzero((char *)&serv_name, sizeof(serv_name));
  hp = gethostbyname (argv[1]);
  memcpy (&(serv_name.sin_addr), hp->h_addr, hp->h_length);
  serv_name.sin_family = AF_INET;
  serv_name.sin_port = htons(1200);

  status = connect(sockd, (struct sockaddr*)&serv_name, sizeof(serv_name));
  if (err == -1) {
    perror("Conexión con el servidor");
    exit(1);
  }

  /* * * * * * * * * PARTE CLIENTE  * * * * * * * * * * * * * *
```

Servidor:

```
int main() {
  int sockd,sockd2;
  struct sockaddr_in dir, cliente;
  int err, len;

  sockd = socket(AF_INET, SOCK_STREAM, 0);
  if (sockd == -1){
    perror("Error en socket");
    exit(1);
  }

  dir.sin_family = AF_INET;
  dir.sin_addr.s_addr = INADDR_ANY;
  dir.sin_port = htons(1200);
  err = bind(sockd, (struct sockaddr*)&dir, sizeof(dir));
  if (err == -1){
    perror("Error en bind");
    exit(1);
  }

  err = listen(sockd, 5);
  if (err == -1){
    perror("error en listen");
    exit(1);
  }

  len = sizeof(peer_name);
  sockd2 = accept(sockd, (struct sockaddr*)&client, &len);
  if (sockd2 == -1) {
      perror("Error en accept");
      exit(1);
  }
  /* * * * * * * * * PARTE SERVIDOR    * * * * * * * * * * * * * *
    Rellenar el código del servidor necesario para recibir el contenido
    de un fichero enviado por el cliente. El servidor debe crear un fichero
    y escribir en él el contenido recibido por el cliente.
   */
  exit(0);
}
```

El programa cliente debe transferir el contenido de un fichero al servidor. El nombre del fichero lo recibe el cliente en la línea de mandatos (en `argv[2]`). El servidor debe crear un fichero con nombre igual y escribir en él el contenido del fichero enviado por el cliente. Se pide:

a) ¿Cuál es el puerto asignado al servidor?
b) ¿Cuál es el puerto asignado al cliente? ¿Cuándo se asigna el puerto al cliente?
c) Escriba en el cliente (en PARTE CLIENTE) y en el servidor (en PARTE SERVIDOR) el código necesario para transferir el fichero del cliente al servidor. El servidor debe crear y escribir el contenido del fichero transmitido por el cliente en un fichero con igual nombre.
(**No** se podrá hacer uso del servicio `sendfile`).

Ejercicio 19. Se desea implementar un servicio de control de incendios mediante una arquitectura distribuida cliente-servidor. Este servicio dispone de una centralita, un sensor que recoge información de la temperatura del ambiente, un sistema anti-incendios y un actuador conectado a un botón. La centralita solicita al sensor la temperatura y humedad registrada cada minuto. En caso de que la centraliza detecte una temperatura mayor de 60 grados, la centralita envía una solicitud de activación al sistema anti-incendios. Cuando la temperatura del sensor baja de los 30 grados, la centralita envía una solicitud de parada del sistema anti-incendios. Si en algún momento un usuario pulsa el botón de alarma, el actuador que controla este botón envía un mensaje a la centralita indicando que hay un incendio. De igual forma, la centralita envía una solicitud de activación al sistema anti-incendios. Cuando la temperatura del sensor baja de los 30 grados, la centralita envía una solicitud de parada del sistema anti-incendios.

1. Realice un diseño completo de la aplicación distribuida, utilizando sockets, describiendo de forma detallada el protocolo de la aplicación. Haga todas las consideraciones que considere oportunas.
2. Indique la estructura en pseudocódigo que tendrían los procesos que ejecutan en los distintos componentes del sistema, indicando las llamadas de la biblioteca de *sockets* que hay que utilizar en los distintos elementos de su aplicación.

Ejercicio 20. Un cliente envía una petición de longitud M bytes a un servidor con un único procesador. El servidor procesa la petición en un tiempo t y devuelve una respuesta de longitud M/2 bytes. El tiempo t de procesamiento de una petición sólo supone tiempo de CPU. La red dispone de una latencia L s. y un ancho de banda de B bytes/s.
Responda razonadamente a las siguientes preguntas:

a) ¿Cuál es el tiempo de respuesta percibido por el cliente?
b) Si el servidor es secuencial, ¿cuántas peticiones como máximo podría atender en T segundos?
c) ¿Y si el servidor es concurrente?
d) Asuma que el servidor dispone de 4 procesadores, ¿cuál sería el tiempo percibido por el cliente? ¿cuántas peticiones como máximo podría atender en T segundos ?

Ejercicio 21. Se quiere desarrollar un sistema de comunicación y sincronización entre un reloj de pulsera inteligente (llamado iCrono) y un teléfono móvil. El reloj inteligente permite interactuar con el teléfono móvil de los usuarios de forma autónoma y a través de sencillos actuadores, como un botón y el movimiento de la muñeca del usuario. Para ello, el usuario puede desde el reloj de pulsera realizar los siguientes servicios:

- *consultarEmail*: Mediante este servicio el usuario puede consultar su correo electrónico. Este método se encargará de consultar al teléfono móvil los últimos 10 correos recibidos,

devolviendo, por cada correo, el emisor del correo y la cabecera del correo. En caso de que haya menos se recibirán los que haya.

- *consultarAgenda*: Mediante este servicio la pulsera solicitará al teléfono los eventos programados para el día en curso. Para ello, se devolverá por cada evento programado, la hora, el lugar, el asunto y el número de participantes en la reunión.

Adicionalmente, el reloj inteligente puede recibir de forma autónoma mensajes que han sido recibidos por el teléfono móvil, con la idea de notificar al usuario de su llegada a través de su pantalla. Para ello se contará con un servicio de notificación que implementará el siguiente servicio:

- *enviarMensaje*: Mediante este servicio se enviará al reloj inteligente la siguiente información: identificador del mensaje, remitente del mensaje y el texto del mensaje con un límite de 100 caracteres.

Considerando que se quiere desarrollar la aplicación utilizando sockets, se pide:

a) Haga un diseño detallado de la aplicación, haciendo todas las consideraciones que crea oportunas, teniendo en cuentas las características de cada uno de los servicios ofrecidos, con el objetivo de maximizar el ahorro de energía de todos los dispositivos presentados.

b) Se pide implementar los siguientes servicios mediante el API de socket en lenguaje C:
   a. El servicio *consultarEmail* en el lado del cliente.
   b. El servicio *consultarAgenda* en la parte cliente.
   c. El sistema de espera de notificaciones del servicio de envío de mensajes en la parte del cliente.

c) Represente los tres servicios presentados anteriormente mediante XDR de SUN. En caso de usarse este tipo de RPC, describa además el proceso y los pasos necesarios para desplegar el sistema en ambos dispositivos.

Ejercicios propuestos de RPC

Ejercicio 1. Dada la siguiente especificación de procedimiento remoto:

```
procedure REMOTO (in int A, inout int B, out int C);
```

Suponiendo que se dispone de las primitivas de comunicación cliente/servidor con las siguientes especificaciones y semánticas:

```
void sendrecv(port p, char *buffer, int *bufsize);
```

Permite a un proceso enviar al puerto p un mensaje con lo que hay en el buffer de tamaño bufsize. A continuación el proceso se queda bloqueado en espera del resultado del servicio. Cuando el proceso se desbloquea se encuentra el resultado en buffer y bufsize indica el tamaño del buffer resultado.

```
void receive(por p, char *buffer, int *bufsize);
```

Permite a un proceso recibir un mensaje por el puerto p. El proceso se bloquea hasta que el mensaje llegue. El mensaje se almacena en buffer y bufsize indica el tamaño del buffer resultado.

```
void reply(char *buffer, int *bufsize);
```

Permite devolver un resultado en buffer con tamaño bufsize al proceso del que se recibió el último mensaje.

```
port createport();
```

Permite obtener un puerto único identificable en todo el sistema distribuido.
Suponiendo que el *binder* (servidor de nombres, enlazador) ofrece tres RPC cuyas especificaciones son:

```
registrar(in char nombre[longmax], int port p);
buscar(in char nombre[longmax], out port p);
borrar(in char nombre[longmax);
```

Sabiendo que nos encontramos en un sistema homogéneo, se pide:

a) Diseñar y programar el suplente del cliente del procedimiento REMOTO, especificando la secuencia de funciones que debe realizar y las estructuras de datos que utiliza.

b) Diseñar y programar el suplente del servidor, especificando la secuencia de funciones que debe realizar y las estructuras de datos que utiliza. El diseño del servidor sólo debe considerar que se atiende una petición en cada momento.
c) Indicar si ha sido necesario un cambio de representación de los parámetros y porqué.
d) ¿Se puede utilizar un servidor concurrente?

NOTA: no es necesario considerar aspectos de seguridad ni tratamientos de error.

Ejercicio 2. Se quiere desarrollar un servidor de almacenamiento remoto, que permita realizar las operaciones siguientes sobre el sistema de ficheros remoto:

- Operaciones sobre ficheros:
  - Abrir
  - Crear
  - Leer datos
  - Escribir datos
  - Cerrar
  - Renombrar
- Operaciones sobre directorios:
  - Crear
  - Borrar
  - Listar el contenido

Se pide:

a) Diseñe la aplicación cliente-servidor anterior utilizando sockets, indicando y especificando todos los aspectos necesarios para su diseño.
b) De acuerdo al diseño anterior, indique qué llamadas a la biblioteca de sockets utilizaría en el cliente y en el servidor y en qué orden.
c) Especifique la misma interfaz utilizando las RPC de Sun.
d) Enumere los pasos necesarios para realizar el cliente de RPC para invocar un procedimiento remoto usando las RPC de Sun.
e) Defina las operaciones realizadas por los *stub* del cliente y del servidor para las operaciones de lectura y escritura de datos en un fichero.
f) Defina las operaciones realizadas por los *stub* del cliente y del servidor para la tarea de listar de un directorio.

Ejercicio 3. En relación a las llamadas a procedimientos remotos responda a las siguientes preguntas:

a) Dado un cliente, un servidor y un *binder*, indique el intercambio de mensajes necesario para registrar un servicio de RPC y el intercambio de mensajes necesario para localizar un servicio de RPC.

b) La siguiente salida corresponde a la ejecución del mandato `rpcinfo -p guernika.lab.inf.uc3m.es`:

```
    100000    2   tcp   111  portmapper
    100000    2   udp  111  portmapper
    100003    2   tcp   2049  nfs
    100003    3   tcp   2049  nfs
    100003    4   tcp   2049  nfs
    100003    2   udp   2049  nfs
    100003    3   udp   2049  nfs
    100003    4   udp   2049  nfs
    200030    2   tcp   9000   calculadora
    200030    2   udp  9000   calculadora
```

¿Cuál es la función del servicio *portmapper*? Si Un cliente intenta acceder a `guernika.lab.inf.uc3m.es`, puerto 9000, servicio "`calculadora`" y versión "1". ¿Se ejecutaría correctamente una llamada a RPC de ese servicio? Justificar la respuesta.

Ejercicio 4. En relación a las llamadas a procedimientos remotos responda a las siguientes preguntas:

a) Dado un cliente, un servidor y un *binder*, indique el intercambio de mensajes necesario para registrar un servicio de RPC y el intercambio de mensajes necesario para localizar un servicio de RPC.

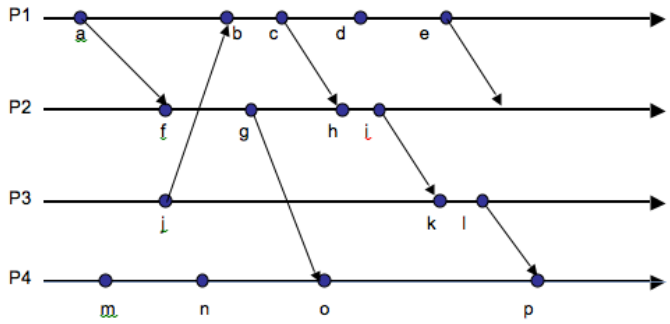
b) La siguiente salida corresponde a la ejecución del mandato `rpcinfo -p guernika.lab.inf.uc3m.es`:

```
100000    2   tcp   111  portmapper
100000    2   udp  111  portmapper
100003    2   tcp   2049  nfs
100003    3   tcp   2049  nfs
100003    4   tcp   2049  nfs
100003    2   udp   2049  nfs
100003    3   udp   2049  nfs
100003    4   udp   2049  nfs
200030    2   tcp   9000   calculadora
200030    2   udp  9000   calculadora
```

¿Cuál es la función del servicio *portmapper*? Si Un cliente intenta acceder a `guernika.lab.inf.uc3m.es`, puerto 9000, servicio "`calculadora`" y versión "1". ¿Se ejecutaría correctamente una llamada a RPC de ese servicio? Justificar la respuesta.

Otros ejercicios propuestos

Ejercicio 1. El siguiente diagrama muestra una serie de eventos que ocurren entre cuatro procesos de una aplicación distribuida.



Se pide:

a) Asignar los valores de los relojes vectoriales, en la forma (v1,v2,v3, V4) a los siguientes eventos del gráfico anterior
b) Para los siguientes pares de eventos diga si se cumple la relación $\rightarrow$ o $\|$ entre ellos:
   - (j,o)
   - (j,p)

   Razone su respuesta

Ejercicio 2. Una aplicación cliente genera una petición a un servidor web de tamaño 1 MB. Si los paquetes que se envían por la red tienen longitud máxima de 64 KB calcule:

a. Número de paquetes que la aplicación cliente necesita enviar al servidor.
b. Tiempo de transmisión de un mensaje asumiendo que el ancho de banda de la red es 100 Mbps y su latencia es de 1 ms por paquete.
c. Tiempo total para enviar la petición completa.
d. Considerando que el servidor conectado a esta red cuenta con recursos suficientes, ¿cuántas transferencias de 1MB podría realizar en 1 segundo?
e.

Ejercicio 3. Un cliente envía una petición de longitud L bytes a un servidor con un único procesador. El servidor procesa la petición en un tiempo t y devuelve una respuesta de longitud L/2 bytes. El tiempo t de procesamiento de una petición sólo supone tiempo de CPU. La red dispone de un ancho de banda de 1Mbit/s. Responda razonadamente a las siguientes preguntas:

a) ¿Cuál es el tiempo de respuesta percibido por el cliente?
b) Si el servidor es secuencial, ¿cuántas peticiones como máximo podría atender en el intervalo de tiempo [1,N]?
c) ¿Y si el servidor es concurrente?
d) Asuma que el servidor dispone de 4 procesadores, ¿cuál sería el tiempo percibido por el cliente?
e) Ventajas e inconvenientes de un servidor secuencial frente a un servidor concurrente.