A continuación se recoge una selección de problemas aparecidos en los exámenes del curso 2011-2012. Tras cada pregunta aparece la caja de respuesta, en el mismo tamaño que tenía en el examen, y dentro de ella la solución. Para leerla debes poner la hoja ante un espejo o mirarla por detrás al trasluz. Seguidamente viene la explicación de cómo se llega a esa respuesta, por si tienes dudas.

❑ Un servidor está esperando clientes en un socket TCP. Cada vez que uno llega, lo acepta, pero no lo atiende inmediatamente. Utiliza `select()` para saber cuándo el cliente le ha enviado datos, y atenderle sólo en ese caso.

El servidor tiene un array de sockets de datos en el que cada elemento del array es cero si no hay cliente en ese socket, o distinto de cero si hay un cliente conectado a él que podría enviar datos. Inicialmente todos los elementos del array son cero, pero a medida que llegan clientes y se aceptan, los elementos del array se van poblando. A medida que los clientes desconectan, se ponen de nuevo a cero.

El código siguiente hace uso de una serie de funciones auxiliares para simplificar el problema. Las funciones en cuestión no se muestran de momento y su misión es la siguiente:

- `InicializarSocketTCP(int)` Recibe un número de puerto y devuelve un socket pasivo TCP que escucha en ese puerto, listo para hacer `accept()` cuando un cliente intente conectarse.
- `Preparar_fd(fd_set *, int, int[])` Recibe en primer lugar la dirección de una estructura `fd_set` que debe inicializar, en segundo lugar el socket de escucha y en tercer lugar la dirección del array que contiene la lista de socket de datos. Lo que hace la función es introducir en el conjunto `fd_set` todos aquellos sockets que tengan cliente conectado, y también el socket de escucha.
- `Calcular_max(int, int[])` Recibe en primer lugar el socket de escucha y en segundo lugar la dirección del array de socket de datos. Devuelve el máximo entre todos los elementos del array y el socket de escucha.
- `Aceptar_cliente(int, int[])` Recibe en primer lugar el socket de escucha en el cual ya hay un cliente que acaba de conectar y espera ser aceptado, y en segundo lugar el array de sockets de datos. La función recorre el array buscando un elemento libre (igual a cero). Si no lo encuentra, acepta momentáneamente al cliente pero inmediatamente después cierra la conexión con él. Si lo encuentra, acepta el cliente y guarda el nuevo descriptor de socket de datos en la posición libre del array.
- `Atender_cliente(int)` Recibe como parámetro un socket de datos en el que un cliente ha enviado datos que esperan ser leídos. Lee los datos, y lleva a cabo el servicio necesario para ese cliente (en el cual no vamos a entrar). La función retorna cuántos bytes ha recibido del cliente. Observar que si el valor retornado es cero, esto implica que el cliente ha cerrado la conexión, pero la función `Atender_cliente()` no hace nada al respecto. Es el programa principal quien debe manejar este cierre.

```
1  #define MAX_SDAT 100
2  #define PUERTO   9999
3  int sdat[MAX_SDAT], sock_con;
4  struct sockaddr_in dir;
5  fd_set fd;
6  int max;
7
8  sock_con=InicializarSocketTCP(PUERTO);
9  for (i=0; i<MAX_SDAT; i++) sdat[i]=0;
10 while(1)
11 {
12   // Vamos a preparar el conjunto de sockets a observar
13   ____________(&fd, sock_con, _________);
14   // Calcula el máximo
15   max=Calcular_max(sock_con, sdat);
16   // Esperar a que lleguen clientes o datos
17   select(max+1, &fd, NULL, NULL, NULL);
18   if (_____________________________) {
19     Aceptar_cliente(sock_con, sdat);
20   }
21   for (i=0; i<MAX_SDAT; i++) {
22     if ((sdat[i]!=0) && (_____________________________))
23        cuenta=Atender_cliente(sdat[i]);
24        if (cuenta==0) {
25           ____________;
26           ____________;
27        }
28   }
29 }
```

— Escribe completa la línea 13, incluyendo lo que falta.

```
Preparar_fd (&fd, sock_con, sdat);
```

Explicación: Antes de llamar a `select` es necesario preparar la estructura `fd` con los sockets que nos interese observar. Esta preparación la hace la función `Preparar_fd` que es lo que hay que poner aquí. El segundo parámetro de esta función es el array que conteiene los sockets de datos a observar, por referencia. En el caso de un array, basta poner su nombre para obtener su referencia.

— ¿Qué falta en la línea 18?

```
FD_ISSET(sock_con, &fd)
```

Explicación: Al retorno de select es necesario averiguar, mediante `FD_ISSET`, qué socket es el que ha causado el retorno. En primer lugar comprobamos el socket de escucha, ya que vemos que la acción tomada en ese caso es `Aceptar_cliente`.

— ¿Qué falta en la línea 22?

```
FD_ISSET(sdat[i], &fd)
```

Explicación: Vemos que se está recorriendo el array de los sockets de datos y si el elemento es distinto de cero y además se da la condición que tenemos que escribir, entonces se llama a `Atender_cliente`. Esto implica que la condición que falta debe comprobar si el cliente ha enviado algo, antes de antenderlo, es decir, si ha habido actividad en el socket `sdat[i]`. Esto se comprueba con `FD_ISSET(sdat[i], &fd)`

— ¿Qué falta en los huecos de la línea 25 y siguiente?

```
close(sdat[i]);
sdat[i]=0;
```

Explicación: Si `cuenta==0` esto significa que el cliente ha cerrado la conexión. Por tanto debemos cerrar también el socket de datos que nos unía a él (`close(sdat[i])`) y eliminar este socket de la lista de sockets que vigilamos. Para esto último basta poner un cero en `sdat[i]` ya que, cuando volvamos a ejecutar la función `Preparar_fd`, ya que ésta sólo considera los que son distintos de cero, el socket .eliminadoza no será vigilado.

Seguidamente se muestra la implementación de las funciones auxiliares que se describieron en el enunciado.

```
1  int InicializarSocketTCP(int p)
2  {
3     int s;
4     struct sockaddr_in d;
5
6     s=socket(PF_INET, SOCK_DGRAM, 0);
7     // Comprobación de errores omitida
8     // Inicializar la estructura sockaddr_in
9     ____________________________________
10    ____________________________________
11    ____________________________________
12
13    // Asignar dirección y puerto al socket
14    bind(_____________________________________);
15    // Comprobación de errores omitida
16    listen(s, SOMAXCONN);
17    return s;
18 } //-----------------------------------------------
```

```
19 void Preparar_fd(fd_set &f, int sc, int sd[])
20 {
21   int i;
22   FD_ZERO(&f);
23   FD_SET(sc, &f);
24   for (i=0; i<MAX_SDAT; i++) {
25    if (              ) FD_SET(              );
26   }
27 }//-----------------------------------------
28 int Calcular_max(int sc, int sd[])
29 {
30   int i;
31   int max=      ;
32
33   for (i=0; i<MAX_SDAT; i++)
34     if (sd[i]>max) max=sd[i];
35   return max;
36 }//-----------------------------------------
37 void AceptarCliente(int sc, int sd[])
38 {
39   int i;
40   int aux;
41
42   aux=          (     , NULL, NULL);
43   for (i=0; i<MAX_SDAT; i++) {
44     if (sdat[i]==0) {
45         ;
46       return;
47     }
48   }
49   close(aux);
50 }//-----------------------------------------
```

— Escribe el código que falta en las líneas 9 y siguientes que inicializa la estructura `sockaddr_in`.

```
                        d.sin_family=AF_INET;
                        d.sin_port=htons(p);
    d.sin_addr.s_addr=htonl(INADDR_ANY);
```

— Completa el `bind` de la línea 14.

```
bind(s, (struct sockaddr *) &d, sizeof(d));
```

— Reescribe la línea 25 completando lo que falta.

```
            if (sd[i]!=0) FD_SET(sd[i], &f);
```

Explicación: Hay que ir poniendo en la estructura `fd_set` (variable `f`) todos los sockets en los que esperamos algo. Estos son, por un lado el socket de conexión (pero esto ya lo hace la función antes de entrar al bucle `for`), y por otro todos los sockets de datos almacenados en el array *y que sean distintos de cero*, ya que los que son cero son los que no tienen cliente conectado.
Por tanto la condición del `if` es que `sd[i]!=0`, y los parámetros a pasarle a `FD_SET` son `sd[i], &f`.

— ¿Cómo habría que inicializar la variable `max` en la línea 31 para que la función realice correctamente su cometido?

```
                                        sc
```

Explicación: Ya que la función debe calcular el máximo entre todos los sockets del array más el socket de conexión (`sc`) y, por lo que vemos en el resto del código de esta función, se va comparando cada elemento del array con la variable `max` para actualizarla, la forma de que tenga también al socket `sc` en cuenta es inicializarla con el valor de dicho socket, y no con cero como podría ser más habitual.

— Completa la línea 42.

```
                    aux=accept(sc, NULL, 0);
```

Explicación: La misión de la función `Aceptar_cliente` es aceptar la conexión que un cliente está intentando sobre el socket de escucha, el cual la función recibe a través del parámetro `sc`. Por tanto se trata simplemente de hacer `accept` sobre dicho socket.
Observar que el nuevo socket de datos que es creado como consecuencia del `accept` se guarda provisionalmente en la variable `aux`, pero más adelante se cierra ese socket si no se encuentra sitio para el nuevo cliente en el array.

— ¿Qué falta en la línea 45?

```
                                sd[i]=aux;
```

Explicación: Vemos que el bucle `for` va revisando todos los elementos del array a la búsqueda de uno que sea cero, lo que indicaría que está "libre". Cuando lo encuentra, tal como dice el enunciado, ha de usarlo para guardar en él el nuevo socket de datos que acaba de ser creado, el cual está provisionalmente en la variable `aux`, por tanto lo que falta es simplemente `sd[i]=aux`.