UNIVERSIDAD POLITÉCNICA DE MADRID
ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA Y DISEÑO INDSUTRIAL

Departamento de Ingeniería Eléctrica, Electrónica, Automática y Física Aplicada

APELLIDOS | **NOMBRE** | Nº Mat.

ASIGNATURA: SISTEMAS INFORMÁTICOS INDUSTRIALES

CURSO 4º **GRUPO** **Enero 2016**

Calificación

2. Problema de Análisis y Diseño Orientado a Objetos (10 puntos - 25 minutos)

Utilice una **clase interna** en la clase *VectorSTD* para implementar el interfaz *IVector*. Se pide.

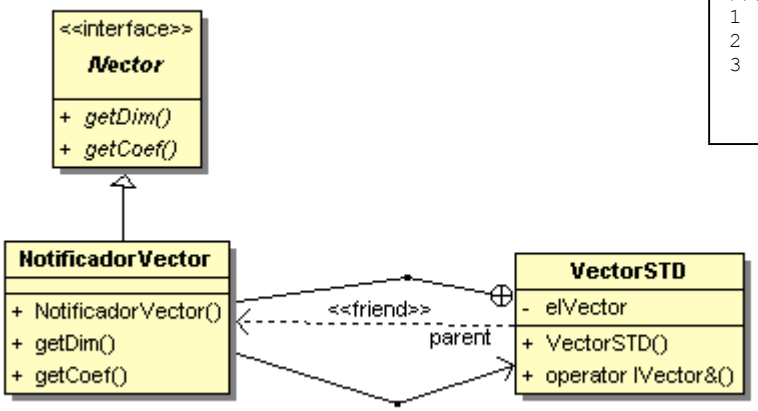
1. Diagrama Clase de Diseño e indique los patrones empleados (5 puntos).
2. Implementación de las clases *IVector* y *VectorSTD* (5 puntos).

```
#include <vector>
#include <iostream>
using namespace std;
class IVector{};
class VectorSTD{};

void showVector(IVector &elVector){
        for(int i=0; i<elVector.getDim();i++)
             cout << elVector.getCoef(i) <<endl;
}

int main(){
        float vector[]={1.0f,2.0f,3.0f};
        VectorSTD miVector(3,vector);
        showVector(miVector);
        return 0;
}

////////////////////////////////
Ejecución del programa
////////////////////////////////
1
2
3
```




```
class IVector{
public:
 virtual int getDim() = 0;
 virtual float getCoef(int) = 0;
};

class VectorSTD{
  vector<float> elVector;
public:
 class NotificadorVector;
 friend class VectorSTD::NotificadorVector;
 class NotificadorVector : public IVector {
   VectorSTD *parent;
   public:
   NotificadorVector(VectorSTD *p):parent(p){}
   int getDim()
     {return parent->elVector.size();}
   float getCoef(int i)
      {return parent->elVector[i];}
 } elNotificador;
 VectorSTD(int dim,float *pV)
   :elNotificador(this){
     for(int i=0;i<dim;i++)
       elVector.push_back(pV[i]);
   }
  operator IVector &(){return elNotificador;}
};
```

UNIVERSIDAD POLITÉCNICA DE MADRID
ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA Y DISEÑO INDSUTRIAL

Departamento de Ingeniería Eléctrica, Electrónica, Automática y Física Aplicada

APELLIDOS | **NOMBRE** | **Nº Mat.** | Calificación

ASIGNATURA: SISTEMAS INFORMÁTICOS INDUSTRIALES

CURSO 4º **GRUPO** **Enero 2016**

3. Problema de Análisis y Diseño Orientado a Objetos (10 puntos - 40 minutos)

Se desea realizar un programa para las taquillas de un cine. En esta primera versión se ha implementado dos tipos de ofertas: i) El día del espectador, los miércoles, con un descuento del 50% en cada entrada. ii) Cualquier día de la semana, se regala una entrada de cada tres. Siempre se aplica la oferta más ventajosa al cliente. Se pide.

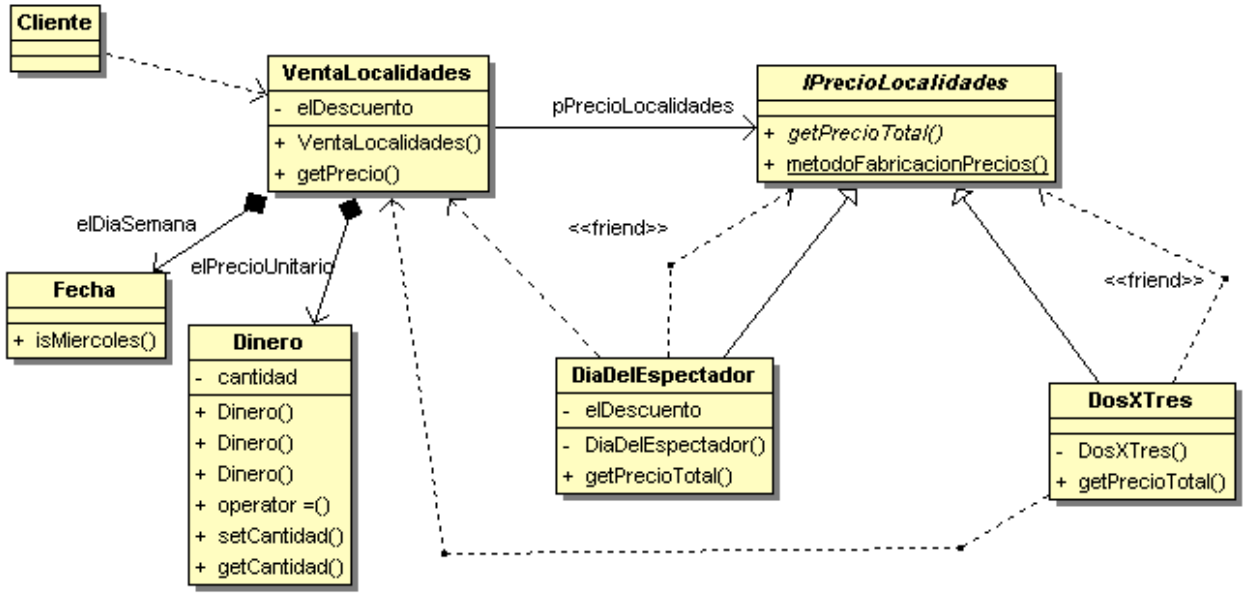
1. Diagrama Clase de Diseño e indique los patrones empleados (5 puntos).
2. Implementación de las clases *Dinero, Fecha, IPrecioLocalidades, DiaDelEspectador, DosXTres* y *VentaLocalidades* (5 puntos).

```
#include <iostream>
using namespace std;
#include "Dinero.h"
#include "Fecha.h"

class IPrecioLocalidades{…};
class VentaLocalidades{…};
class DiaDelEspectador {…};
class DosXTres{…};


int main()
{
 float precio;
 cout <<"\nPrecio unitario en euros: ";
 cin  >> precio;
 Dinero elPrecioUnitario(precio);
 VentaLocalidades ventanilla(elPrecioUnitario);
 cout << "\nNumero de localidades a vender:";
 int numeroLocalidades;
 cin  >> numeroLocalidades;
 Dinero elValorVenta=
       ventanilla.getPrecio(numeroLocalidades);
 cout << "A cobrar " << elValorVenta.getCantidad()
      <<" euros.";
 return 0;
}
////////////////////////////////
Ejecución del programa
Hoy no es miércoles
////////////////////////////////
Precio unitario en euros: 9
Numero de localidades a vender: 4
A cobrar 27 euros
```

Patrón GoF Estrategia.



```
typedef enum {EURO, DOLAR, YEN}TipoDinero;

class Dinero{
 TipoDinero elTipoMoneda;
 float cantidad;
public:
 Dinero(): elTipoMoneda(EURO), cantidad(0) {}
 Dinero(float valor): elTipoMoneda(EURO),
   cantidad(valor) {}
 Dinero(Dinero &elValor){
   elTipoMoneda = elValor.elTipoMoneda;
    cantidad = elValor.cantidad;
 }
 Dinero& operator= (Dinero &elValor){
   elTipoMoneda = elValor.elTipoMoneda;
   cantidad = elValor.cantidad;
   return(*this);
 }
 void setCantidad(floatlaCantidad)
  {cantidad=laCantidad;}
 float getCantidad(void){return cantidad;}
};

#include <ctime>

class Fecha {
 public:
  bool isMiercoles(){
   time_t tSac = time(NULL);// instante actual
   struct tm* tmP = localtime(&tSac);
   return tmP->tm_wday == 3 ? true:false;
 }
};

class VentaLocalidades;
class IPrecioLocalidades{
public:
 virtual Dinero getPrecioTotal
 (int numLocalidades, Dinero elPrecioUnitario) = 0;
 static IPrecioLocalidades *
   metodoFabricacionPrecios(VentaLocalidades *);
};

class VentaLocalidades{
 Fecha elDiaSemana;
 Dinero elPrecioUnitario;
 float descuentoDiaDelEspectador;
 IPrecioLocalidades *pPrecioLocalidades;
public:
 VentaLocalidades(Dinero elPrecio) :
  elPrecioUnitario(elPrecio)
  {descuentoDiaDelEspectador=50/100;}
 Dinero getPrecio(int numeroLocalidades){
  pPrecioLocalidades =
 IPrecioLocalidades::metodoFabricacionPrecios(this);
  return ( pPrecioLocalidades>getPrecioTotal
    (numeroLocalidades,elPrecioUnitario) );
 }
 ~VentaLocalidades() {
 if(pPrecioLocalidades) delete pPrecioLocalidades;
 }
 float getDescuentoDiaEspectador() {
   return descuentoDiaDelEspectador;
 }
 bool isMiercoles() {
  return elDiaSemana.isMiercoles();
 }
};
```

```
class DiaDelEspectador : public IPrecioLocalidades
{
 friend class IPrecioLocalidades;
 VentaLocalidades *pContexto;
 DiaDelEspectador(VentaLocalidades *pC):
       pContexto(pC) {}
 public:
 virtual Dinero getPrecioTotal
 (int numeroLocalidades, Dinero elPrecioUnitario){
  Dinero elValorTotal;
  elValorTotal.setCantidad
  (elPrecioUnitario.getCantidad()*
   numeroLocalidades*
   pContexto->getDescuentoDiaEspectador());
  return (elValorTotal);
  }
};

class DosXTres : public IPrecioLocalidades
{
 friend class IPrecioLocalidades;
 VentaLocalidades *pContexto;
 DosXTres(VentaLocalidades *pC):
    pContexto(pC) {}
 public:
virtual Dinero getPrecioTotal
(int numeroLocalidades, Dinero elPrecioUnitario){
 Dinero elValorTotal;
 int multiplosDeTres = 0;
 for(int i=1;i<=numeroLocalidades;i++)
       if( i%3 == 0)
              multiplosDeTres++;
 int localidadesSueltas =
  numeroLocalidades - (multiplosDeTres*3);
 elValorTotal.setCantidad
  (elPrecioUnitario.getCantidad()*
  (localidadesSueltas+(multiplosDeTres*2)));
  return (elValorTotal);
 }
};

IPrecioLocalidades *
IPrecioLocalidades::metodoFabricacionPrecios
(VentaLocalidades *pContexto){
 if (pContexto->isMiercoles() == true )
   return new DiaDelEspectador(pContexto);
  else return new DosXTres(pContexto);
}
```

UNIVERSIDAD POLITÉCNICA DE MADRID
ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA Y DISEÑO INDSUTRIAL

Departamento de Ingeniería Eléctrica, Electrónica, Automática y Física Aplicada

APELLIDOS

NOMBRE Nº Mat.

ASIGNATURA: SISTEMAS INFORMÁTICOS INDUSTRIALES

CURSO 4º GRUPO Enero 2016

Calificación

4. Problema de Sistemas Operativos (20 puntos - 45 minutos)

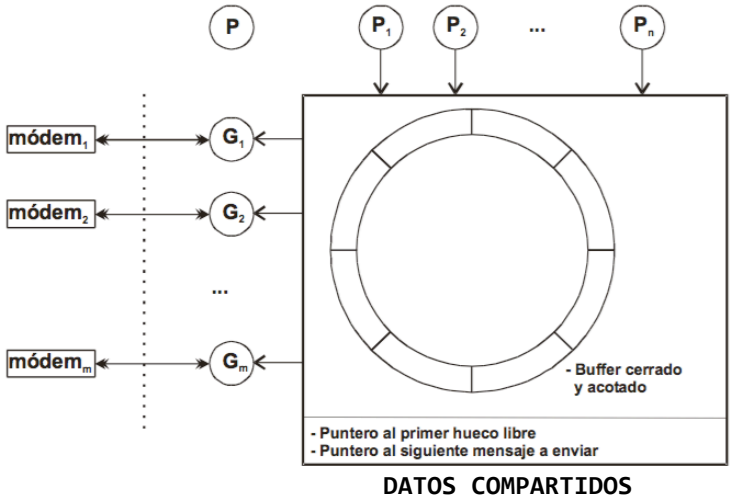
Se desea realizar un sistema de envío de mensajes SMS a través de una serie de módems GSM.

Todos los SMS a ser enviados deben almacenarse en un buffer circular acotado de tamaño N mensajes. De este modo, los procesos que vayan a enviar un mensaje deben rellenar el siguiente hueco libre del buffer con la siguiente información:

```
struct mensaje {
     int tfnoDestino;     // Número de teléfono de destino
     int tfnoOrigen;      // Número de teléfono de origen
     char sms[161];       // Cuerpo del mensaje
}
```

Para implementar este sistema se plantea la estructura de procesos de la figura. El esquema pretende utilizar un **fichero proyectado en memoria** con la información compartida, la cual podría implementarse mediante la estructura siguiente:

```
struct compartido {
       mensaje buffer[MAX_BUFFER];
       int sigHueco;
       int sigElemento;
}
```



Los procesos se describen a continuación:

- El proceso general P se encarga de crear todos los mecanismos de sincronización necesarios y de crear la zona donde se proyecta el fichero en memoria que comparte la información.

- Los procesos P_i son los que desean enviar mensajes. Para enviar uno, P_i debe rellenar el siguiente hueco vacío del buffer. P_i recibe los datos del mensaje a enviar como parámetros de salida de la siguiente función (supóngase ya implementada):

  ```
  datosMensaje (int * tfnoDestino, int * tfnoOrigen, char * sms)
  ```

- Hay un proceso G_i asociado a cada uno de los módems, que son los que envían los mensajes. Estos procesos procesos recogen los mensajes pendientes de envío del buffer y se lo envían a su correspondiente módem. Cada uno de estos procesos abre el dispositivo "/dev/modem$_i$" y, por cada mensaje dado, se llama a la siguiente función bloqueante (supóngase ya implementada):

  ```
  enviarMensaje(int descriptorModem, int tfnoDestino, int tfnoOrigen, char * sms)
  ```

  Nota: Para abrir el dispositivo "/dev/modem$_i$" utilizad la función *open* que devuelve el descriptor de fichero asociado.

Se debe suponer que todos los procesos anteriores son **independientes entre sí** y que todos ellos conocen los nombres de los mecanismos de comunicación y sincronización creados por el proceso P.

Se pide:

1. Según el sistema planteado, los punteros al primer hueco libre y al siguiente mensaje a enviar están en e fichero proyectado en memoria. ¿Sería más eficiente tener almacenados estos valores en cada uno de los procesos? En caso afirmativo, ¿en cuál de ellos? Razónelo (2 puntos)

   Los punteros deben ser compartidos por todos los procesos. Por tanto, es necesario que estén almacenados en algún objeto compartido que pueda ser accedido por todos los procesos. No puede estar en los propios procesos porque dificultaría la actualización de los valores de los punteros en todos los procesos.

   Tener los puntos en el objeto compartido exige que los procesos accedan en exclusión mutua para modificar dichos punteros.

2. Implementar el proceso P siguiendo el esquema planteado. (4 puntos)

   Los mecanismos de comunicación y sincronización que se deben crear por P deben ser con nombre puesto que todos los procesos de la solución son independientes entre sí. El problema es un caso típico de procesos productores y consumidores. En este caso se plantea la solución con el uso de semáforos para el acceso al buffer (elementos y huecos) y otro para el acceso a la sección crítica (sembin).

```
#define MAX_BUFFER 1000
#define MAX_MENSAJES 90000

struct mensaje {
     int tfnoDestino;     // Número de teléfono de destino
     int tfnoOrigen;      // Número de teléfono de origen
     char sms[161];       // Cuerpo del mensaje
};

struct compartido {
       mensaje buffer[MAX_BUFFER];
       int sigHueco;
       int sigElemento;
};

semt_t *elementos, *huecos, *sembin; // Declara los semáforos

int main (void) {
       int fd;
       char *direccion;
       struct compartido * objCompartido;

       // El proceso P crea el fichero a proyectar en memoria */
       fd = open("/tmp/datos.txt", O_CREAT | O_RDWR | O_TRUNC, 0666);

       // Proyectar el objeto compartido en su espacio de direcciones
       direccion = (char *)mmap(NULL, sizeof(objCompartido), PROT_READ|PROT_WRITE, MAP_SHARED, fd,
       0);
       close(fd);
       objCompartido=(struct compartido*)direccion;

       // Inicializa punteros de huecos y elementos utilizando el puntero
       objCompartido->sigHueco = 0;
       objCompartido->sigElemento = 0;

       // Inicializa los semáforos
       elementos = sem_open("ELEMENTOS", O_CREAT, 0700, 0);
       huecos = sem_open("HUECOS", O_CREAT, 0700, MAX_BUFFER);
       sembin = sem_open("SEMBIN", O_CREAT, 0700, 1);
}
```

3. Implementar uno de los procesos P_i siguiendo el esquema planteado. (7 puntos)

```
#define MAX_BUFFER 1000
#define MAX_MENSAJES 90000

struct mensaje {
     int tfnoDestino;     // Número de teléfono de destino
     int tfnoOrigen;      // Número de teléfono de origen
     char sms[161];       // Cuerpo del mensaje
};

struct compartido {
       mensaje buffer[MAX_BUFFER];
       int sigHueco;
       int sigElemento;
};

semt_t *elementos, *huecos, *sembin; // Declara los semáforos


void Pi (void) {

       int i, fd;
       char *direccion;
       struct compartido * objCompartido;
       int tfnoOrigen, tfnoDestino; char sms[160];

       /* Proyecta el fichero sobre su espacio de direcciones */
       fd = open("/tmp/datos.txt", O_RDWR);
       direccion = (char *)mmap(NULL, sizeof(objCompartido), PROT_READ|PROT_WRITE, MAP_SHARED, fd,
       0);
       close(fd);
       objCompartido=(struct compartido*)direccion;

       /* Abre los semáforos */
       elementos = sem_open("ELEMENTOS", 0);
       huecos = sem_open("HUECOS", 0);
       sembin = sem_open("SEMBIN", 0);

       for(i=0; i < MAX_MENSAJES; i++ ) {
              // Se lee el mensaje a enviar del buffer
              datosMensaje(&destino, &origen, sms);
              sem_wait(huecos); // Un hueco menos
              sem_wait(sembin); // Acceso con exclusión mutua
              // El mensaje se inserta en el objeto compartido
              (objCompartido -> buffer[objCompartido->sigHueco]).tfnoDestino = tfnoDestino;
              (objCompartido -> buffer[objCompartido->sigHueco]).tfnoOrigen = tfnoOrigen;
              strcpy(sms ,(objCompartido -> buffer[objCompartido->pHueco]).sms);
              objCompartido->sigHueco = (objCompartido->sigHueco + 1) % MAX_BUFFER;
              sem_post(sembin);  // Se sale de la sección crítica
              sem_post(elementos); // Un elemento más
       }
}
```

4. Implementar uno de los proceos G_i siguiendo el esquema planteado. (7 puntos)

```
#define MAX_BUFFER 1000
#define MAX_MENSAJES 90000

struct mensaje {
     int tfnoDestino;     // Número de teléfono de destino
     int tfnoOrigen;      // Número de teléfono de origen
     char sms[161];       // Cuerpo del mensaje
};

struct compartido {
       mensaje buffer[MAX_BUFFER];
       int sigHueco;
       int sigElemento;
};

semt_t *elementos, *huecos, *sembin; // Declara los semáforos

void Gi (void) {

       int i, fd, fdModem;
       char *direccion;
       struct compartido * objCompartido;
       int tfnoOrigen, tfnoDestino; char sms[160];

       /* Proyecta el fichero sobre su espacio de direcciones */
       fd = open("/tmp/datos.txt", O_RDWR);
       direccion = (char *)mmap(NULL, sizeof(objCompartido), PROT_READ|PROT_WRITE, MAP_SHARED, fd,
       0);
       close(fd);
       objCompartido=(struct compartido*)direccion;

       /* Abre los semáforos */
       elementos = sem_open("ELEMENTOS", 0);
       huecos = sem_open("HUECOS", 0);
       sembin = sem_open("SEMBIN", 0);

       // El consumidor tiene que abrir el módem GSM relacionado
       fdModem = open(“/dev/modemi”);

       for(i=0; i < MAX_MENSAJES; i++ ) {
              sem_wait(elementos); // Un elemento menos
              sem_wait(sembin); // Acceso con exclusión mutua
              // Se lee el mensaje del objeto compartido
              tfnoDestino = (objCompartido -> buffer[objCompartido->sigElemento]).tfnoDestino;
              tfnoOrigen = (objCompartido -> buffer[objCompartido->sigElemento]).tfnoOrigen;
              strcpy(sms, (objCompartido -> buffer[objCompartido->sigElemento]).sms);
              objCompartido->sigElemento = (objCompartido->sigElemento + 1) % MAX_BUFFER;
              sem_post(sembin);  // Se sale de la sección crítica
              sem_post(huecos); // Un hueco más
              // Se lee el mensaje a enviar del buffer
              enviarMensaje (fdModem, tfnoDestino, tfnoOrigen, sms);
       }
}
```

UNIVERSIDAD POLITÉCNICA DE MADRID
ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA Y DISEÑO INDSUTRIAL

Departamento de Ingeniería Eléctrica, Electrónica, Automática y Física Aplicada

APELLIDOS

NOMBRE Nº Mat.

ASIGNATURA: SISTEMAS INFORMÁTICOS INDUSTRIALES

CURSO 4º GRUPO Enero 2016

Calificación

5. Problema de Sistemas Operativos (10 puntos - 20 minutos)

Se proporciona el código de un programa cliente donde se pide un número de DNI y se envía al servidor para que éste le devuelva la letra asociada a dicho número. Se pide implementar el programa **servidor concurrente** utilizando sockets de tipo stream que permita atender a N clientes que funcionen con el código proporcionado. Supóngase que ya existe una función implementada con el siguiente prototipo:

```
char obtenerLetra (int dni);
```

```
int main(void)
{
        int sd;
        struct sockaddr_in server_addr;
        struct hostent *hp;
        int dni;
        char letra;

        sd = socket(AF_INET, SOCK_STREAM, 0);
        if (sd < 0){
                printf("Error en socket\n");
                return 1;
        }

        hp = gethostbyname ("138.100.100.1");
        if (hp == NULL){
                printf("Error en la llamada gethostbyname\n");
                return 1;
        }
        memcpy (&(server_addr.sin_addr), hp->h_addr, hp->h_length);

        server_addr.sin_family = AF_INET;
        server_addr.sin_port = htons(4200);

        if (connect(sd, (struct sockaddr *) &server_addr,
sizeof(server_addr)) < 0) {
                printf("Error en la llamada connect");
                return 1;
        }
        printf("Introduce el DNI\n");
        scanf ("%d", &dni);
        dni=htonl(dni);

        write(sd, (char *)&dni, sizeof(int));

        read(sd, (char *)&letra, sizeof(int));

        printf("Letra: %c \n", letra);
        close (sd);
        return 0;
}
```

Código del servidor (solución basada en procesos pesados):

```
int main(void)
{
  struct sockaddr_in server_addr,client_addr;
  int sd, sc;
  int size;
  int dni;
  char letra;

  sd = socket(AF_INET, SOCK_STREAM, 0);
  if (sd < 0) {
    printf("Error en la llamada socket\n");
    return 1;
  }

  server_addr.sin_family = AF_INET;
  server_addr.sin_addr.s_addr = inet_addr("138.100.100.1"); // IP del servidor
  server_addr.sin_port = htons(4200);

  int on=1;
  setsockopt(sd,SOL_SOCKET,SO_REUSEADDR,&on,sizeof(on));

  if (bind(sd, (struct sockaddr *) &server_addr, sizeof(server_addr)) < 0){
    printf("Error en la llamada bind\n");
    return 1;
  }
  listen(sd, 5);
  size = sizeof(client_addr);
  while (1){
    sc = accept(sd, (struct sockaddr *)&client_addr, &size);
    if (sc < 0){
      printf("Error en accpet\n");
      break;
    }
    else {
       if (fork() == 0) {
             /* recibe la petición, un número entero */
             if (read(sc, (char *)&dni, sizeof(int)) < 0){
                   printf("Error en read\n");
                   break;
             }

             letra=obtenerLetra(dni);  // Se invoca a la función que calcula la letra

             /* envía el resultado y cierra la conexión */
             if (write(sc, (char *)&letra, sizeof(int)) < 0){
                   printf("Error en write\n");
                   break;
             }
             close(sc);
             return 0;
       }
    }
  }
  close (sd);
  return 0;
}
```