7. Aplicación para el e-Assessment en la asignatura Programación Orientada a Objetos

En los seis primeros capítulos de este documento se ha ido introduciendo todo lo necesario para entender globalmente esta herramienta e-Assessment que se presenta en el capítulo 7. Se han estudiado los tipos de bases de datos en el mercado y se ha profundizado en tres soluciones, una por cada tecnología, todo esto con vista a elegir la mejor base de datos posible para la herramienta en cuestión. Para la elección se realizó un test de rendimiento que presentó MongoDB como una base de datos con unos resultados altamente satisfactorios, bastante mejores que el del resto de soluciones estudiadas. Con todo lo anterior asimilado, la herramienta que a continuación se presenta puede entenderse como un todo, que aúna evaluación, código y almacenamiento de datos.

Como se ha comentado, el objetivo principal de este Proyecto Final de Carrera consiste en la realización de una herramienta informática que sirve de ayuda para la evaluación del sistema de prácticas de la asignatura Programación Orientada a Objetos, impartida por el Departamento de Ingeniería Telemática para el Grado de Ingeniería de Sistemas de Telecomunicación.

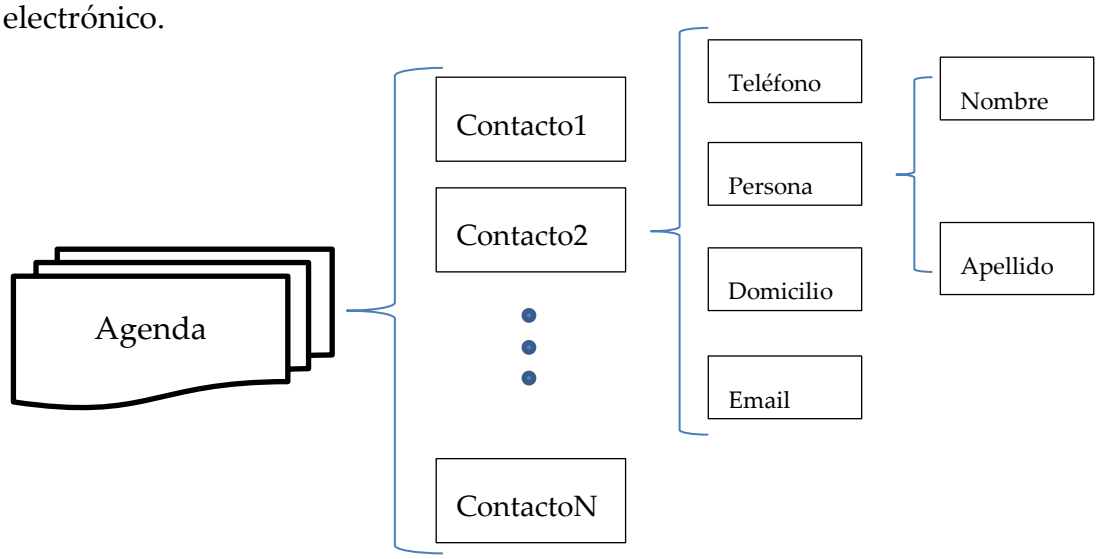
7.1 Introducción

Como parte del proceso de prácticas de la asignatura Programación Orientada a Objetos, los alumnos de la misma deben realizar una práctica final, de manera individual, en la que deben demostrar los conocimientos adquiridos durante el desarrollo del curso académico.

Cuando se habla de e-Assessment se hace referencia a una forma de evaluación remota que permite comprobar el nivel de conocimientos de una persona en un campo determinado. Hace uso de herramientas informáticas, y normalmente se tratan de sistemas en línea. Estas herramientas no requieren de interacción con las personas, por lo que trabajando de forma autónoma realizando una serie de procesos reiterativos en el tiempo.

Este tipo de herramientas permiten en cierta forma un ahorro de costes, y una mayor comodidad para el usuario, ya que se obvia la figura del evaluador, que puede dedicar su tiempo de trabajo a otros menesteres, y por otra parte se permite realizar la evaluación en cualquier momento de tiempo, lo que supone una importante ventaja para el alumno. Asimismo, estas herramientas se encuentran más allá de cualquier concepto de subjetividad en la evaluación, lo que ayuda a evitar las habituales suspicacias en las correcciones.

La evaluación a realizar se centra en la práctica final de la asignatura anteriormente mencionada; el trabajo del alumno consiste en el desarrollo de un código en lenguaje de programación JAVA para modelar el funcionamiento de una agenda, formada por un conjunto de contactos, cada uno de los cuales consta de un número de teléfono, una persona, un domicilio y un correo electrónico.



Sobre la agenda a desarrollar se podrán hacer una serie de operaciones, para lo cual se provee a los alumnos de unas interfaces que deberán desarrollar para poder superar con éxito la prueba.

7.2 Funcionamiento de la Herramienta e-Assessment

Bajo los supuestos explicados en la introducción, la herramienta desarrollada en este Proyecto Final de Carrera se encarga de tomar el código escrito por el alumno, almacenarlo en la base de datos MongoDB, y según sea necesario, compilar el código y/o ejecutar el mismo en una máquina servidora, en un proceso transparente al alumno, que únicamente espera la respuesta de la aplicación para conocer si el resultado ha sido válido, o si por el contrario necesita modificaciones. Según el resultado de la operación, la evaluación correspondiente será subida a la base de datos para que los profesores de la asignatura puedan saber la nota obtenida por los alumnos.

Para que todo lo expuesto anteriormente sea viable, se decide realizar una aplicación cliente y otra servidora, para que dialoguen entre ellas de forma autónoma. La aplicación cliente estará localizada junto al código de cada usuario, por lo que habrá una aplicación cliente por usuario. La aplicación servidora se encuentra ubicado en el Departamento de Ingeniería Telemática, donde puede ser accedido por los alumnos de la asignatura a través de la aplicación cliente. Se muestra una síntesis del funcionamiento de ambas a continuación:

- Cliente: Toma todos los archivos necesarios del usuario, junto con su login, para subirlos a la aplicación servidora. Una vez que realizado lo anterior, se le presentan dos opciones al alumno para que decida qué hacer: 1. Compilar el código y 2. Ejecutar las pruebas establecidas para su código.

- Servidor: Escucha permanentemente ante potenciales usuarios y en el momento en que se establece una conexión, lo primero que se hace es tomar los ficheros subidos por el cliente y subirlos a la base de datos MongoDB, en una base creada para el usuario. Posteriormente y tras recibir la opción elegida por el usuario, compilará el código, o pasará a su ejecución. Para cualquiera de las dos pruebas, se devolverá un mensaje de OK o de fallo al cliente; en caso de fallo, se devuelve el fichero de salida de errores para que el alumno pueda ver dónde está el error y proceder a su corrección.

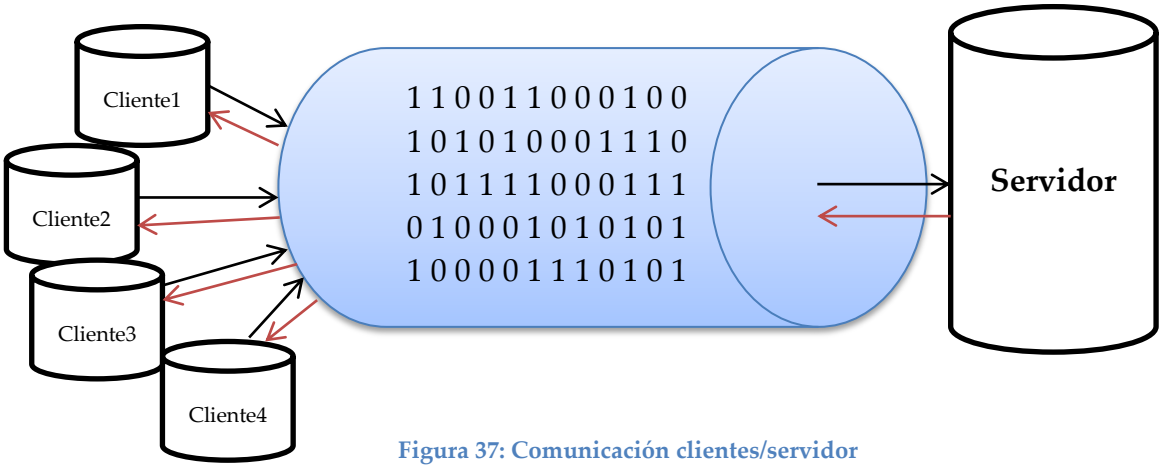


Figura 37: Comunicación clientes/servidor

Para llevar a cabo lo anterior se desarrolla una interfaz gráfica muy sencilla que permite al usuario interactuar con la máquina servidora y poder de esta forma elegir entre las opciones disponibles.

Es importante remarcar que el proceso debe ser totalmente transparente al alumno, ya que en caso contrario se añadiría una dificultad de comprensión adicional que dificultaría el trabajo de los alumnos, por lo que se les proporciona una normativa de uso de la herramienta clara y específica que pretende evitar lo anterior.

La herramienta se desarrolla en la distribución Linux Ubuntu Server, pero es útil para cualquier otra distribución. El lenguaje de programación utilizado para el desarrollo del código es C.

Se proporciona en la siguiente página una línea de tiempo con los procesos que se llevan a cabo en este proyecto entre ambas aplicaciones:

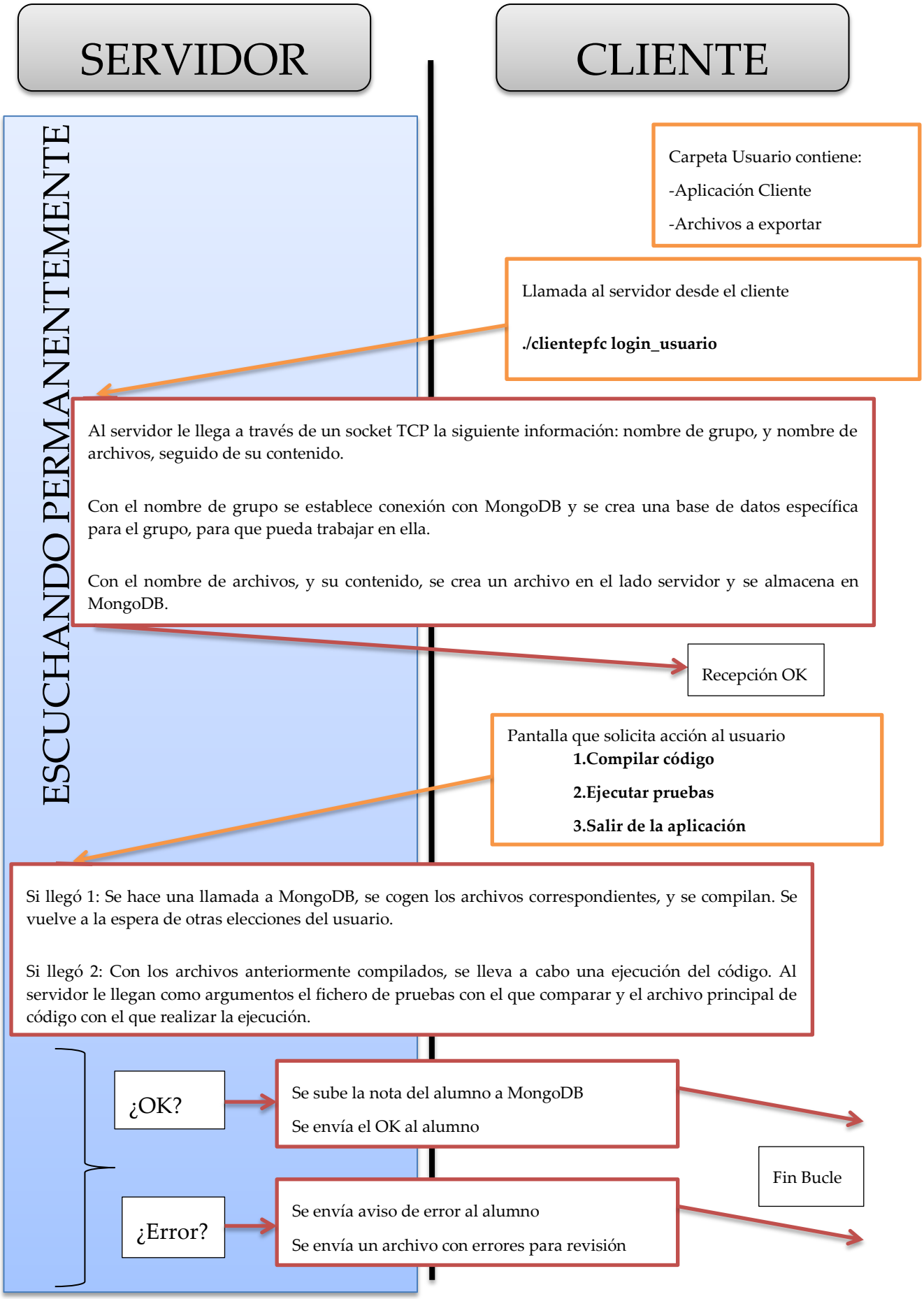


Figura 38: Línea temporal de la aplicación

Una vez explicado el funcionamiento básico de la herramienta se va a explicar en detalle cómo se ha construido, y qué hacen las aplicaciones cliente y servidor, con nombres ClientePFC y ServidorPFC respectivamente.

7.3 Aplicación ClientePFC

La aplicación ClientePFC es la que encargada de la interacción con el servidor desde la parte del alumno, debiendo éste instalarlo para poder ejecutar la herramienta.

7.3.1 Organización de los directorios

La organización del directorio vendrá dada por la siguiente estructura:

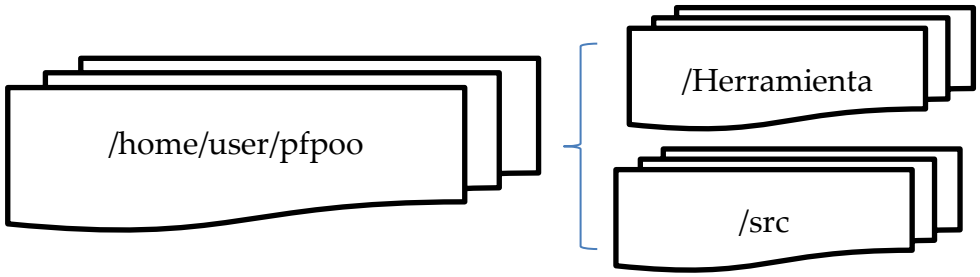


Figura 39: Estructura básica carpeta aplicación cliente /home/user/pfpoo

La aplicación y el código estarán alojados en la carpeta principal pfpoo, y contendrá las carpetas Herramienta y src, común a todos los usuarios. Además de estas dos carpetas, pfpoo también alojará el fichero makefilexxx donde xxx es el login del alumno.

La carpeta src será la base donde estará todo el código del alumno:

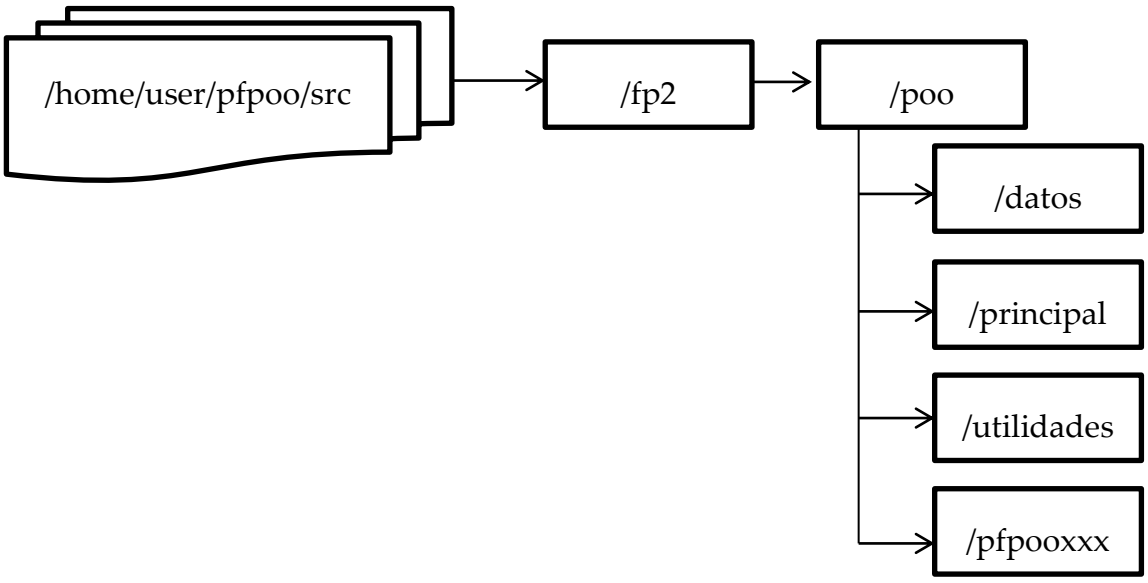


Figura 40: Organización interna de la carpeta src : /home/user/pfpoo/src

- Datos: Contiene los ficheros de entrada para pasar las pruebas
- Utilidades: Contienen las interfaces sobre las que construir el código del alumno y la carpeta que contiene las excepciones que debe lanzar la aplicación en su caso.
- Principal: Contiene la clase Principal.class necesaria para la ejecución de las pruebas
- Pfpooxxx: Carpeta individual de cada usuario xxx en la que alojará su código para realizar la funcionalidad de una agenda. Alojará los siguientes archivos:
  - Agenda.java
  - Contacto.java
  - Persona.java
  - Domicilio.java
  - Telefono.java
  - CorreoElectronico.java

La carpeta Herramienta aloja la aplicación cliente desarrollado en el transcurso de este Proyecto Final de Carrera. Contiene los siguientes ficheros:

- Clientepfc.c
- Funciones_cliente.c
- Funciones_cliente.h
- Constantes.h
- Makefile

Para poder hacer uso de la aplicación ClientePFC, el alumno debe llamar al comando make dentro de la carpeta Herramienta, para así construir la aplicación, para finalmente llamar a la aplicación seguida por su nombre de usuario para iniciar la comunicación con el servidor:

```
>>./clientepfc mi_login
```

Este último comando dará inicio a todo el proceso cliente que se verá en los siguientes apartados.

7.3.2 Funcionamiento de ClientePFC

Una vez se ha llamado a la aplicación cliente mediante el comando anteriormente explicado, se inician una serie de procesos que tienen como propósito principal subir los archivos de código del usuario al servidor, y allí compilarlo y ejecutar una serie de pruebas que evaluarán al alumno de cara a la nota final de la asignatura Programación Orientada a Objetos del Departamento de Ingeniería Telemática.

La interfaz gráfica que tiene el alumno es sencilla para evitar problemas mayores derivados de una difícil comprensión; a continuación se muestra una imagen:

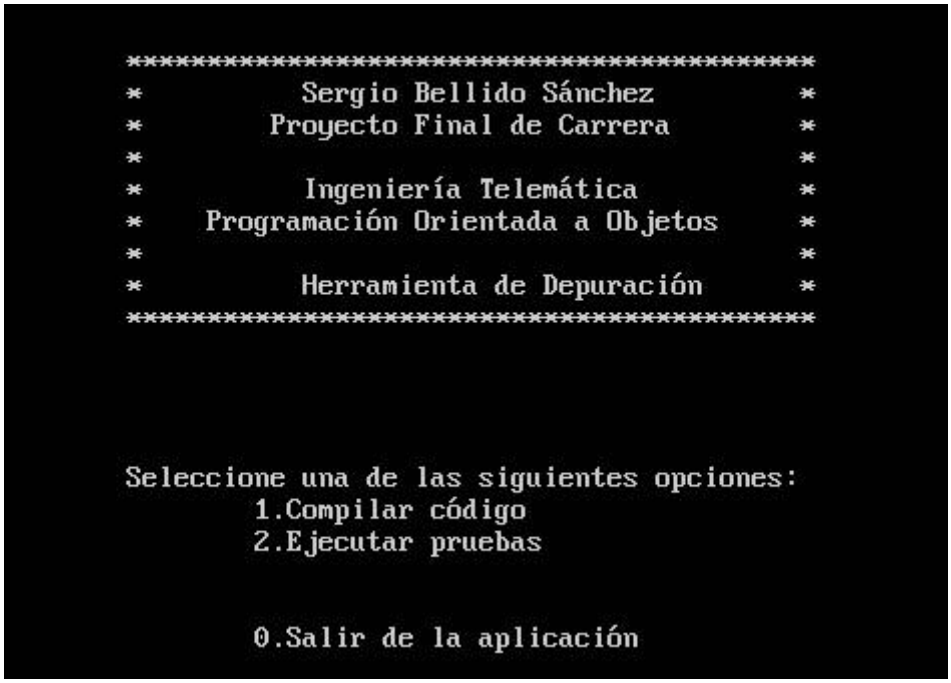


Figura 41: Pantalla de inicio para el usuario

Antes de que aparezca esta pantalla se hace una breve comprobación para ver que se ha insertado el nombre de usuario, y si no fuera así, se sale de la aplicación. Recordar que la llamada debe ser como sigue:

```
>>./clientepfc mi_login
```

Una vez se ha iniciado ClientePFC el siguiente paso es una rutina que comprueba que estén todos los archivos necesarios para que el código elaborado por el alumno sea válido. La aplicación buscará en las carpetas del usuario, dónde deben ubicarse los ficheros, y en caso de ser todo correcto seguirá ejecutándose; en caso de error, se parará la aplicación y no llegará a comunicarse con el servidor.

```
void comprobacion(char *concatenado, char **archivos)
```

Una vez realizada la comprobación y siendo válido el resultado, se procede al primer punto complejo de la aplicación, que es la conexión con el servidor. Para ello se va a destacar el siguiente código dentro de la función *crea_socket*():

```
int dir = inet_aton("192.168.1.12", &(destino.sin_addr));
```

El código de arriba va a definir a qué máquina vamos a conectarnos durante el resto del proceso.

```
if((puerto = getservbyname("pfc","tcp")) == NULL)
{
	printf("error funcion tomar puerto");
	exit(1);
}
```

Se crea un servicio, pfc, asociado al puerto 8888, y que utiliza protocolo orientado a conexión TCP, que va a asegurar que las tramas lleguen en orden al servidor. Para que los usuarios puedan hacer uso de este servicio deben modificar el archivo /etc/services y añadir la siguiente línea de código

```
#pfc   8888/tcp
```

Con lo anterior, (sin #), ya será posible trabajar a través del puerto seleccionado.

```
if(connect(descrito,(struct sockaddr *)&destino,sizeof(struct sockaddr)))
	error("connect");
```

El código anterior es el que realiza la conexión hacia el destino con las características especificadas, algunas de las cuales ya hemos visto, y alguna más que puede verse en el anexo 3 Código Aplicación ClientePFC.

Tras establecer la conexión, el primer paso es mandar el nombre de usuario al servidor, para que éste pueda crear una base de datos específica para el usuario, y además crear las carpetas necesarias por usuario para que puedan trabajar con el código de forma local.

Una vez establecida la conexión y creado el usuario en el servidor, los ficheros de código JAVA del usuario serán subidos al servidor con una rutina que se repite para cada uno de los archivos. Primeramente el cliente coge el nombre del archivo y lo sube al servidor, una vez hecho esto, se coge el contenido del mencionado fichero, se lee de principio a fin, se almacena en un buffer y se manda al servidor para que lo trate.

El código reducido de la rutina sería el que muestra a continuación, que es el correspondiente a la comunicación básica de conexión mutua y envío y recepción de datos, (puede ampliarse el código completo en el anexo correspondiente):

```
if((cuantos = send(descrip_local,cadenas[h], strlen(cadenas[h]), 0)) <0){}
if((cuantos = recv(descrip_local, respuesta, 1, 0)) < 1)
	error("\nError al recibir información de creación de fichero");

origen = open(concateno1, O_RDONLY);
if(origen < 0){}
do
{
	if((leidos = read(origen, buffer_fichero, 100)) < 0)
	{}
	if(leidos > 0)
	{
		cuantos = send(descrip_local, buffer_fichero, leidos, 0);
		if(cuantos < 0)
			error("\nError al enviar contenido fichero");
	}
}
while(leidos > 0);
```

Lo que ha ocurrido justo arriba es que se ha enviado primeramente el nombre del archivo, que se encuentra en *cadenas[h]*, posteriormente se ha recibido una confirmación, más tarde se ha abierto mediante *open()* el archivo especificado en la cadena concateno, se ha metido todo el contenido en *buffer_fichero*, y se han enviado tantos bytes como se han leído en el fichero.

Una vez ha finalizado toda la parte de subida de archivos, la aplicación ClientePFC queda a la espera de una respuesta por parte de la aplicación ServidorPFC. Una vez que ésta haya llegado, la comprobación inmediata es ver si es un OK u otra respuesta; en caso de OK continua la aplicación, en otro caso, se termina la ejecución.

La segunda parte de la aplicación viene a la hora de elegir la opción de trabajo, que ya vimos en la imagen de comienzo de este apartado. Tiene tres opciones:

- Compilar el código
- Ejecutar las pruebas
- Salir del programa

7.3.2.1 Compilar el código

Si la opción elegida ha sido la primera, correspondiente a la ejecución del código fuente del alumno, la aplicación cliente envía un "1" a ServidorPFC, y éste ejecutará una rutina remota en la que compilará el código JAVA proporcionado por el alumno a través de la aplicación ClientePFC.

Se muestra un pantallazo del aviso al alumno sobre la opción elegida:

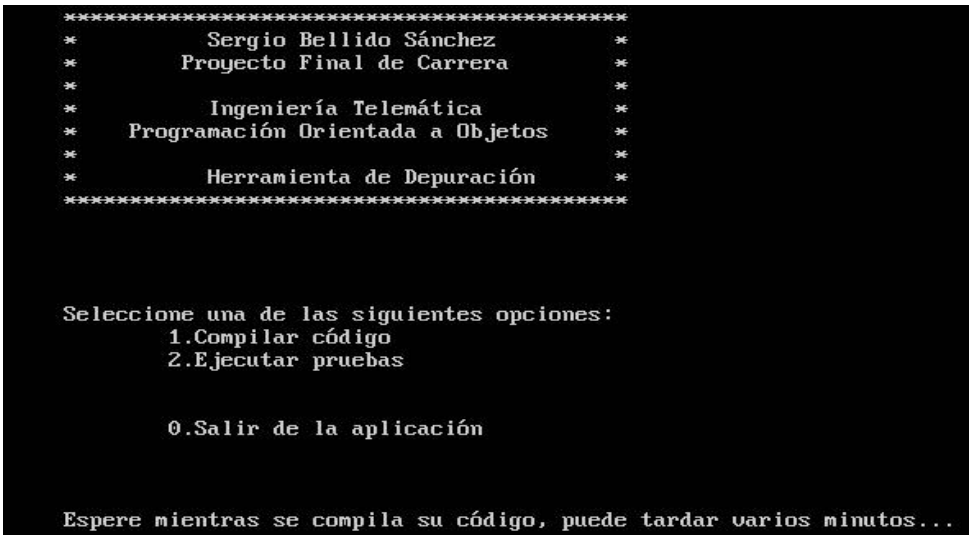


Figura 42: Interfaz de espera mientras se compila el código

Como respuesta tenemos dos opciones, OK o mal. Si empezamos por la primera el usuario recibirá una respuesta confirmándole el éxito de la operación, mostrando el siguiente mensaje:

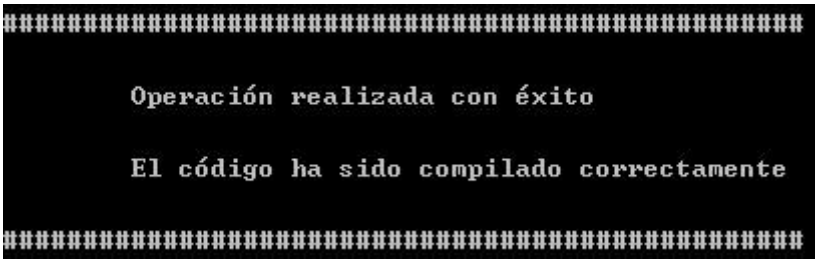


Figura 43: Compilación exitosa

Si la respuesta de la compilación ha sido negativa, significa que a la hora de compilar el código se han detectado problemas en el mismo, y que por tanto no ha podido finalizar la operación. Cuando ocurre lo anterior, la aplicación ClientePFC espera la llegada del archivo de errores, y cierra la aplicación. El alumno recibe por pantalla un aviso de que su código no ha podido ser validado debido a fallos en el mismo.

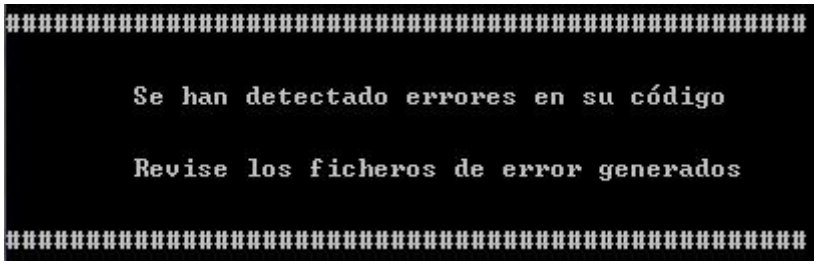


Figura 44: Código no ejecutado debido a errores del mismo

Este archivo de errores estará en la carpeta Herramienta y permitirá al alumno observar dónde ha estado el fallo, o los fallos en su caso, para poder solventarlo y entonces volver a utilizar la aplicación.

7.3.2.2 Ejecutar las pruebas

Si la opción elegida ha sido la segunda, lo primero que debe hacer el alumno es elegir el fichero de entrada que quiere utilizar para pasar las pruebas, cuyos nombres son **agendaxy**, donde "x" e "y" son números, y que se alojan en la carpeta *Datos* de la ruta */home/user/pfpoo/src/fp2/poo/Datos*

Figura 45: Invitación a meter el nombre del fichero de entrada a pruebas

Una vez se ha comprobado que el archivo existe, se mandan al servidor el nombre del archivo y su contenido.

El siguiente paso es tomar el código para la ejecución de las pruebas, cuyo nombre es **Principalxy.java**, siendo "x" e "y" los mismos números utilizados para la elección del fichero de entrada.

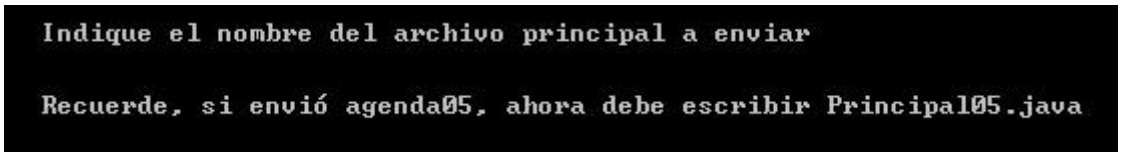


Figura 46: Invitación a meter el nombre del archivo JAVA para ejecutar pruebas

Una vez se ha tomado el nombre, también se comprueba su existencia y en caso afirmativo se envía su contenido al servidor para que pueda ejecutar la prueba correspondiente.

Como en el apartado de compilación, aquí las respuestas pueden ser Ok o mal. En el primer caso se informa al alumno de su éxito mostrando la misma pantalla de enhorabuena que para el éxito en la compilación del código. En caso de error, de nuevo se muestra una pantalla de aviso en el que se le indica al alumno que debe revisar su directorio para ver qué errores ha cometido en su código.

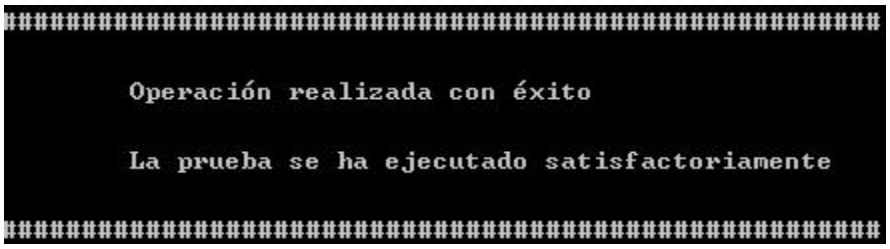


Figura 47: Éxito en la ejecución de la prueba

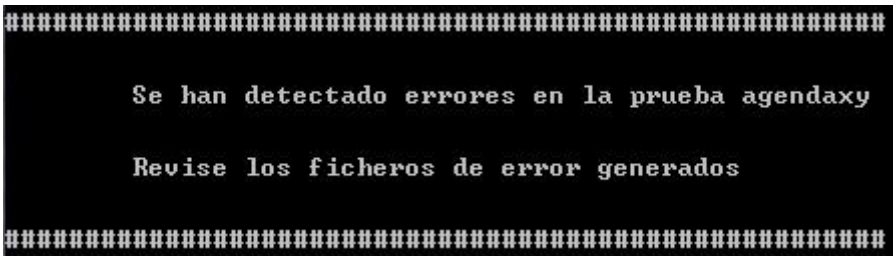


Figura 48: Indicación de que hay fallos en su código y no ha sido posible superar la prueba

En síntesis, lo explicado en el apartado 8.2.2 es el funcionamiento cíclico de la aplicación ClientePFC. El alumno podrá utilizarlo tantas veces como sea necesario hasta completar con éxito las pruebas a las que se somete su código, que es el objetivo prioritario de esta herramienta.

Destacar que en la Herramienta de e-Assessment ServidorPFC se insertan las notas de los usuarios en la base de datos MongoDB en caso de éxito, por lo que el alumno puede estar seguro de que se evaluación está actualizada y disponible en todo momento para los profesores de la asignatura.

En este apartado se han visto todos los aspectos interesantes que forman la aplicación desarrollada para el lado cliente, del alumno, para que pueda existir diálogo con la entidad remota donde se evaluará el desempeño del alumno. Todo el código está disponible en el anexo 3: Código de la aplicación ClientePFC

El siguiente apartado se va a centrar en la aplicación servidora, que llevará el peso de la evaluación del usuario: ServidorPFC.

7.4 Aplicación ServidorPFC

La aplicación ServidorPFC es la encargada de recibir el código de los alumnos a través de unos flujos que permitirán embeber el código .java desarrollado por el usuario, dentro de esta aplicación escrita en lenguaje C.

Su propósito es validar el código, compilarlo, y en su caso ejecutar una serie de pruebas definidas por los profesores de la asignatura Programación Orientada a Objetos, del departamento de Ingeniería Telemática, con la intención final de realizar la evaluación de los alumnos.

Como herramienta de e-Assessment, debe asistir a la evaluación de los alumnos, facilitando la tarea de los profesores con la automatización de los procesos, de forma que los encargados de la asignatura sólo deban encargarse de problemas puntuales de los alumnos, y no de la verificación de las pruebas. Además, la herramienta proveerá a los profesores de la nota correspondiente al alumno, que podrá ser consultada desde la base de datos MongoDB.

7.4.1 Organización de los directorios

Para poder desarrollar esta aplicación en la parte servidora se han debido instalar primeramente la base de datos MongoDB, y su driver C para que pueda existir una interacción entre la herramienta y la base de datos. En el anexo 1: Manual de Instalación podrá consultar más detalle.

De esta forma, tendremos la siguiente organización de directorios, que tendrán como raíz /home/ubuntuserver/

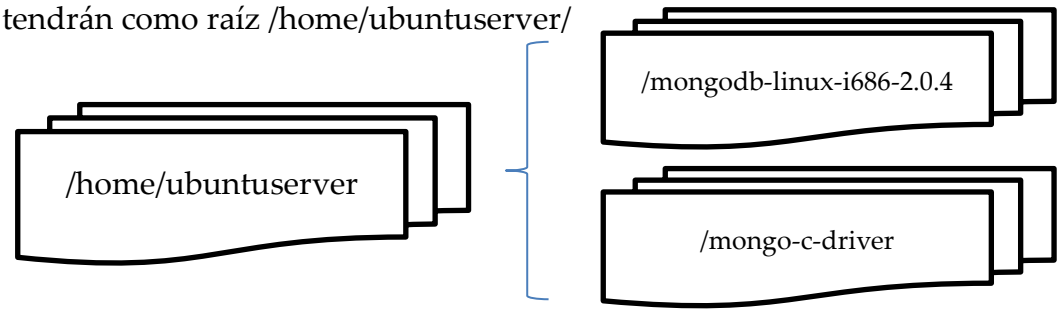


Figura 49: Directorios incluidos dentro de la raíz /home/ubuntuserver/

Dentro del directorio /mongodb-linux-i686-2.0.4 tendremos los ejecutables de la base de datos MongoDB.

En el directorio /mongo-c-driver estará alojado el driver de lenguaje C de MongoDB y el código de la aplicación ServidorPFC. Se muestra a continuación una distribución del mismo:

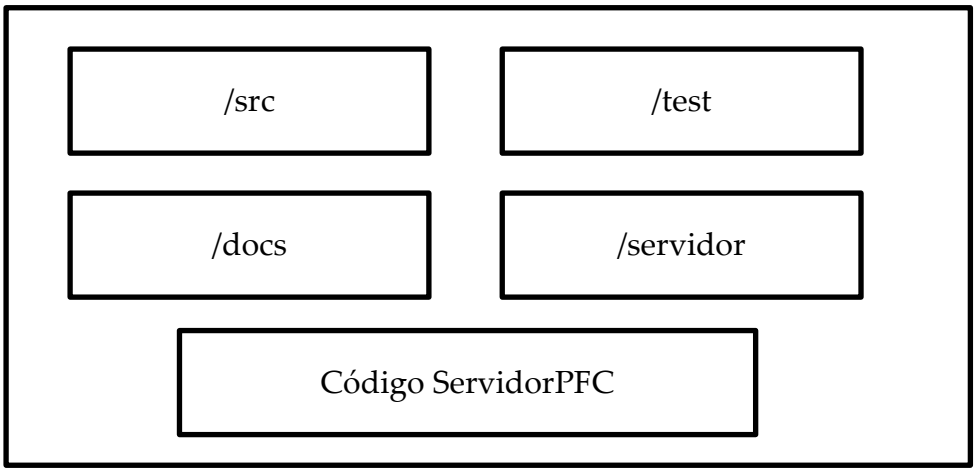


Figura 50: Organización del directorio /mongo-c-driver

Las carpetas /src /test y /docs son propias del driver C para MongoDB. La carpeta /servidor alojará el código y ejecuciones enviadas por los alumnos. El código de la aplicación también se ubicará en esta carpeta, entrando posteriormente en detalles del mismo.

El directorio /servidor cuenta con otra carpeta /pfpoo, y dentro de esta ya encontramos las distintas posibilidades que se presentan:

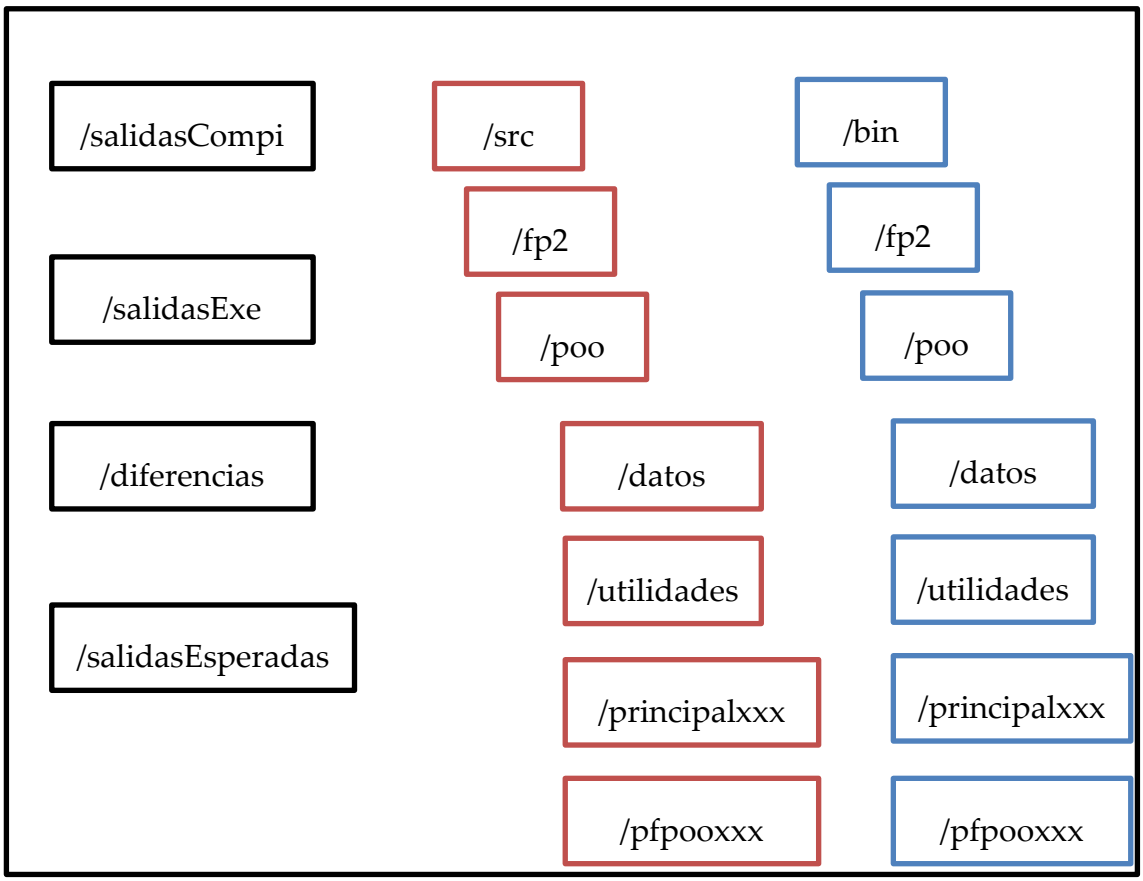


Figura 51: Organización del directorio /servidor/pfpoo

- /salidasCompi: Directorio que almacena los ficheros de errores de la compilación de cada código de los alumnos.
- /salidasExe: Directorio que almacena los ficheros de errores de cada prueba ejecutada para cada alumno.
- /diferencias: Directorio que almacena los ficheros donde se almacenarán las diferencias existentes entre las salidas de los ficheros ejecutados y las salidas esperadas para los mismos
- /salidasEsperadas: Directorio que almacena las salidas que deberían tener los códigos ejecutados por los alumnos.

La carpeta /src va a almacenar el código de los alumnos una vez estos elijan la opción "compilar" desde la aplicación cliente. Las carpetas /principalxxx y /pfpooxxx se crearán para cada uno de los usuarios.

La carpeta /bin va a almacenar los ficheros .class generados tras la compilación del código, y que será utilizado para la ejecución de las pruebas previstas por los profesores de la asignatura. Al igual que el directorio /src, /bin tiene unas carpetas propias por cada usuario que hace uso de la herramienta: /principalxxx y /pfpooxxx. Las otras carpetas son comunes a todos.

- /datos: almacenará los ficheros de entradas para pruebas previstas en la asignatura: agenda00, agenda01, … agendaxy
- /utilidades: almacenará las interfaces sobre las que se apoyará el código de los alumnos para la elaboración del código. También contiene la carpeta /Excepciones para la gestión de excepciones JAVA que debe lanzar el código desarrollado por el alumno
- /principalxxx: Contiene el código .java que tendrá el método main() que llevará a la ejecución de las pruebas de la asignatura. Serán Principal00.java, Principal01..java, … Principalxy.java
- /pfpooxxx: Contiene el código desarrollado por los alumnos, que se habrán apoyado en las interfaces disponibles en la carpeta /utilidades, y que deben tener los siguientes nombres:
  - Agenda.java
  - Personas.java
  - Contacto.java
  - Telefono.java
  - Domicilio.java
  - CorreoElectronico.java

Por último, señalar que el código sobre el que se ha escrito esta aplicación ServidorPFC consta de los siguientes ficheros:

- servidorPFC.c
- funciones_mongo.c
- funciones_mongo.h
- funciones_servidor.c
- funciones_servidor.h
- respuestas_cliente.c
- respuestas_cliente.h
- makefile

En este apartado se ha visto la organización de los ficheros y directorios que forman parte de la aplicación ServidorPFC. El número de subcarpetas en los directorios /src y /bin obedece a la propia organización de las prácticas, que se lleva a cabo de modo ordenado y normalizado para que la evaluación pueda ser correcta a todos los alumnos. Las carpetas creadas para la generación de ficheros de error se justifican desde el punto de vista educativo, ya que proporcionar al alumno los errores que contiene su código puede ayudarle a superar de forma autónoma las pruebas a las que se somete en la evaluación.

7.4.2 Funcionamiento de ServidorPFC

Para poder hacer uso efectivo de la herramienta de e-Assessment objeto de este Proyecto Final de Carrera hay que utilizar el terminal de Linux, y escribir el siguiente comando:

```
>>make
```

Tras lo cual se construirá la aplicación, y para arrancarla:

```
>>./servidorpfc
```

Una vez realizados estos pasos preliminares, la herramienta va a estar escuchando de manera permanente, hasta que de forma manual paremos el proceso.

Esta herramienta no precisa, y no usa, interfaz gráfica alguna para interactuar, más allá de los errores internos del sistema que puedan aparecer por pantalla como aviso de que algo no ha funcionado como estaba previsto.

Lo primero que hay que señalar es que para poder trabajar con MongoDB a través de su driver hay que crear una serie de variables que se detallan a continuación:

```
mongo conn[1];
gridfs gfs[1];
gridfile gfile[1];
bson *p;
```

El primero declara una estructura de tipo mongo que servirá para establecer la conexión con la base de datos. El segundo, gridfs, sirve para insertar ficheros completos en MongoDB. Gridfile ayuda en la lectura y selección de los ficheros insertados por gridfs, y por último, el objeto bson sirve para insertar documentos de distintos tipos en MongoDB, desde cadenas de caracteres, enteros u otros objetos propios de MongoDB.

Tras la declaración de las variables necesarias para que todo el sistema funcione, se llama a la función crea_socket(), que crea un flujo para aceptar conexiones y devuelve un entero que caracteriza a ese flujo. Algunas de las funciones utilizadas en ClientePFC ya se mostraron, y otras nuevas deben ser usadas desde el lado servidor para crear los contextos:

```
descriptor = socket(AF_INET, SOCK_STREAM, 0)
inet_aton("192.168.1.12", &(local.sin_addr))
puerto=getservbyname("pfc","tcp")
bind(descriptor, (struct sockaddr *) &local, sizeof(local))
listen(descriptor, encola)
```

Las funciones nuevas en este caso son bind(), que asocia un descriptor a un puerto, y listen(), que hará que el servidor escuche y espere conexiones de hasta un número de equipos limitado por la variable “encola”.

Una vez se ha creado el contexto, comenzamos la rutina de escucha de conexiones, que estará continuamente a la espera de que un equipo quiera conectarse al servidor:

```
descriptor_nuevo=accept(descriptor,(struct sockaddr *)&servidor, &len)
```

Lo anterior devuelve un entero, que es un descriptor de la conexión establecida con el equipo remoto. Una vez que la conexión se ha establecido se inicia el diálogo con la máquina remota, que tendrá instalada la aplicación ClientePFC.

Se inicia pues una rutina que esperará la llegada primeramente del nombre de usuario para crear una base de datos con su nombre, y las carpetas correspondientes /principalxxx y /pfpooxxx en los directorios /src y /bin.

```
usuario = crear_usuario(cadena1);
status = conectar_mongo(conn, usuario);
status = conectar_grid(conn, usuario, gfs);
```

A la función crear_usuario se le pasa como parámetro "cadena1", que contiene el nombre de usuario que ha sido recibido en el servidor a través de la conexión que ha establecido con el cliente. La función devuelve el propio nombre de usuario, que será utilizado en los dos siguiente métodos para la creación del contexto en la base de datos MongoDB.

El siguiente paso del diálogo cliente/servidor consiste en quedar a la espera de el nombre del primer fichero, para crearlo, y posteriormente recibir su contenido. Una vez que tenemos el fichero con su nombre y contenido en el servidor, exactamente igual que como se encontraba en la máquina del alumno, procedemos a insertarlo en MongoDB, y así tantas veces como ficheros tengamos que recibir.

```
void insertar_mongodb(char *nombre, gridfs *gfs)
{
	int valor;
	valor = gridfs_store_file(gfs, nombre, nombre, "text/html");
}
```

La función insertar_mongodb tiene como argumentos el nombre del fichero a insertar, y un objeto gridfs para trabajar con estos ficheros, y que previamente ha sido asociado a la base de datos propia del usuario en MongoDB. La función gridfs_store_file() insertará el fichero con nombre "nombre" en la base de datos, y le asignará el nombre "nombre", que en este caso se han mantenido iguales para no crear confusión.

Para ver con más detalles cómo se realizan las conexiones con la base de datos MongoDB, así como estudiar cómo se ha realizado el paso de ficheros entre cliente y servidor puede ver el anexo 4: Código Herramienta ServidorPFC.

Una vez finalizada esta primera parte del proceso, en la que ha existido intercambio de datos entre las aplicaciones ClientePFC y ServidorPFC, llega el momento de elegir entre la opción compilar el código, o ejecutar las pruebas, para lo que nos introducimos en una nueva función: *rutina_eleccion()*; Lo primero que hace esta rutina es mandar un OK, indicando que la subida de archivos ha sido correcta, y luego espera la respuesta con la opción elegida por el alumno:

```
send(descriptor_nuevo, okay, strlen(okay), 0)
recv(descriptor_nuevo, respuesta, 100, 0)
```

7.4.2.1 Compilar el Código

El cliente ha mandado una respuesta, y hemos comprobado que contiene el número 1, lo que indica que el alumno quiere compilar su código. El proceso básico de esta rutina es tomar los ficheros necesarios de la base de datos MongoDB, guardarlos en el directorio /home/ubuntuserver/mongo-c-driver/servidor/pfpoo/src/fp2/poo/pfpooxxx , siendo xxx el nombre de cada usuario. Una vez guardados los ficheros, se podrá trabajar con ellos.

Para tomar los datos desde MongoDB hay que diferenciar entre dos versiones de esta herramienta e-Assessment, para CPUs de 64 bits y 32 bits, ya que en versiones de 32 bits el driver no trabaja bien a la hora de seleccionar ficheros en la base de datos MongoDB.

- Versión 64 bits:

```
valor1 = gridfs_find_filename(gfs, linea, gfile);
stream = fopen(ficheros, "w+");
gridfile_write_file(gfile, stream);
gridfile_destroy(gfile);
fclose(stream);
```

Con *gridfs_find_filename()* tomamos el objeto *gridfs* asociado a la base de datos del usuario en MongoDB, le pasamos como argumento el nombre del archivo que estamos buscando, *"linea"*, y el fichero es devuelto a un objeto de tipo *gridfile*. Posteriormente abrimos un archivo con el mismo nombre del fichero alojado en MongoDB, con la función *gridfile_write_file()*, conseguimos volcar el contenido del archivo situado en MongoDB al fichero localizado en el sistema de ficheros locales.

- Versión 32 bits

```
stream = fopen("mongofiles -d usuario get fichero
                                    -l./ruta_hasta/pfpooxxx/fichero, "w");

close(stream);
```

```
sergio@ubuntu:~$ ./mongodb-linux-i686-2.0.4/bin/mongofiles -d fp2 list
connected to: 127.0.0.1
Principal.java  1020
Contacto.java   2158
Domicilio.java  1026
Telefono.java   2448
Persona.java    2239
CorreoElectronico.java  1555
Agenda.java     5951
```

Figura 52: Comprobación de que el usuario fp2 tiene los archivos en su base de datos

La función *popen*() crea un flujo interno de datos dentro de la aplicación que permite usar comandos propios del sistema Linux. En este caso, usa el comando mongofiles, proporcionado junto con la base de datos MongoDB y que permite interactuar directamente con los ficheros de la misma, pudiendo insertar, seleccionar o listar datos. En este caso lo usamos para seleccionar (*get*) el fichero con nombre "*fichero*", en la base de datos asignada al usuario "*usuario*", y que será guardado en el sistema local de ficheros, en la ruta indicada: */home/ubuntuserver/mongo-c-driver/servidor/pfpoo/src/fp2/pfpooxx*

Una vez se han almacenados los ficheros en el sistema de archivos local al servidor, se llama a la función *compilar_aplicacion(makefilexxx)*, a la que se le pasa como argumento el archivo makefile correspondiente al usuario y que servirá para construir la aplicación desarrollada por el alumno.

```
pipe = popen("cd ./servidor/pfpoo/; make –f makefilexxx
                              2>./salidasCompi/errormakefilexxx", "w");
```

Lo anterior crea un flujo que sitúa el programa en el directorio donde está el makefile necesario para construir el código del alumno, y lo construye llamando específicamente a makefilexxx con la opción –f. La salida de errores se guardará en el directorio /salidasCompi desde el que se hará la comprobación necesaria para saber si la operación ha sido exitosa o no.

Si el proceso de compilar el código ha sido correcto y sin errores, el archivo errormakefilexxx estará vacío, y en caso contrario, contendrá los errores de programación del alumno. Esto lo comprobamos con esta rápida rutina:

```
archivo = open(./servidor/pfpoo/salidasCompi/errormakefilexxx, O_RDONLY);
leidos = read(archivo, line, tamaño)
if(leidos == 0)
       i=1;
else
       i=0;
close(archivo);
```

En resumen, si el archivo abierto estaba vacío, devuelve i=1, y si contenía datos, entonces i=0. Esto será devuelto a la rutina principal para que actúe en función del valor recibido.

7.4.2.1.1 Caso de Éxito

Si el archivo de errores estaba vacío, entonces el servidor manda una respuesta de OK al cliente, indicando que todo ha finalizado correctamente, tras esto se indicará que la rutina debe llegar a su fin y que quede a la espera de nuevas conexiones, y finalmente subirá la nota del alumno a la base de datos Notas, en la que se alojará el nombre de los alumnos que han superado con éxito la prueba de compilación.

```
bson_init(p);
bson_append_new_oid(p, "_id");
bson_append_string(p, "nombre", alumno);
bson_append_string(p, "apartado", "compilacion");
bson_append_int(p, "nota", 10);
bson_finish(p);
```

```
sergio@ubuntu:~$ ./mongodb-linux-i686-2.0.4/bin/mongo notas
MongoDB shell version: 2.0.4
connecting to: notas
> db.getCollectionNames()
[ "evaluacion", "system.indexes" ]
> db.evaluacion.find()
{ "_id" : ObjectId("4fba61249cf4387e00000015"), "nombre" : "fp2", "apartado" : "
compilacion", "nota" : 10 }
{ "_id" : ObjectId("4fba61d89cf4387e00000024"), "nombre" : "xxx", "apartado" : "
compilacion", "nota" : 10 }
>
```

Figura 53: Notas de los usuarios fp2 y xxx tras superar la prueba de compilación

Con esto se da por finalizado el ciclo con el usuario, quedando a la espera de nuevas operaciones.

7.4.2.1.2 Caso de Error

Si el archivo de errores contiene la descripción de errores del alumno, entonces se le pasa el contenido del fichero a la aplicación ClientePFC en la máquina remota del alumno; de esta forma, el usuario podrá conocer qué fallos tiene su código para así proceder a su corrección, con la posibilidad de utilizar de nuevo la herramienta de e-Assessment hasta que consiga validar su código.

```
 GNU nano 2.2.6          Archivo: salidacompile

./src/fp2/poo/pfpoofp2/Telefono.java:32: error: invalid method declaration; ret$
    public Telefono3(){
           ^
1 error
make: *** [bin/fp2/poo/pfpoofp2/Telefono.class] Error 1
```

Figura 54: Ejemplo de código de error recibido por el usuario

Al considerarse inválido el código, no se actualiza la nota del usuario en la base de datos Notas de MongoDB, considerándose la ausencia del nombre del usuario en la base de datos como prueba no superada por parte del alumno.

7.4.2.2 Ejecutar las pruebas

Se ha comprobado que la respuesta del cliente ha sido 2, y por tanto se inicia la rutina que va a tratar la ejecución de las pruebas que han diseñado los profesores de la asignatura para que el código ya compilado de los alumnos pueda demostrar su validez.

En este caso vamos a quedar a la espera de que el cliente remoto envíe el fichero que se utilizará para pasar las pruebas del alumno, agendaxy, una vez recibido, se envía respuesta OK y quedamos a la espera de recibir un segundo archivo, el archivo Principalxy.java correspondiente a la prueba mencionada. Una vez hayamos recibido los ficheros, y los hayamos reubicado en su directorio local, /datos y /principalxxx respectivamente, pasaremos a la rutina de ejecución,

```
ejecutar_pruebas(agendaxy, makefilexxx);
```

En esta rutina se van a generar dos ficheros en función de cómo haya sido la ejecución.

La primera ejecución se hará a través de la función popen(), y es como sigue:

```
fp = popen("cd ./servidor/pfpoo; make agenda00 –f makefilexxx”, “w");
```

Esta línea ejecutará la *agenda00*, para lo que hará uso del archivo *Principal00.java*, se generará el *.class* correspondiente en el directorio */home/ubuntuserver/mongo-c-driver/servidor/pfpoo/bin/fp2/poo/principalxxx*.Tras esto se creará también el archivo *makefilexxxagenda00*, dentro del directorio ubicado en */home/ubuntuserver/mongo-c-driver/servidor/pfpoo/salidasEXE*. Este archivo generado servirá para comparar la salida de la prueba con la salida esperada en la siguiente operación.

```
fp = popen("diff ./servidor/pfpoo/salidasExe/makefilexxxagenda00
                 ./servidor/pfpoo/salidasEsperadas/salidaagenda00
                 >diffmakefilexxxagenda00", "w");
```

Lo anterior utiliza el comando de Linux diff, que compara dos ficheros y devuelve nada en caso de que sean idénticos, o la diferencia en caso de que no sean exactamente iguales.

La forma de saber si el alumno ha pasado la prueba es entrar en el último archivo generado, diffmakefilexxxagenda00 y comprobar si está vacío, si es así, se habrá superado la prueba exitosamente, en caso contrario no se habrá superado.

7.4.2.2.1 Caso de éxito

Si el archivo de diferencia diffmakefilexxxagenda00 estaba vacío, entonces el servidor manda una respuesta de OK al cliente, indicando que todo ha finalizado correctamente, tras esto se indicará que la rutina debe llegar a su fin y que quede a la espera de nuevas conexiones, y finalmente subirá la nota del alumno a la base de datos Notas, en la que se alojará el nombre de los alumnos que han superado con éxito la prueba de compilación.

```
bson_init(p);
bson_append_new_oid(p, "_id");
bson_append_string(p, "nombre", xxx);
bson_append_string(p, "apartado", "ejecucion");
bson_append_string(p, "prueba", agenda00);
bson_append_int(p, "nota", 10);
bson_finish(p);
valor = mongo_insert(conne, "notas.evaluacion", p);
```

En este caso se ha añadido un elemento más al documento respecto al caso de compilación de código. Se añade "prueba", que indicará el nombre de la prueba pasada. Esto es posible gracias a la naturaleza desnormalizada de MongoDB, que no obliga a utilizar unos modelos rígidos de datos, si no que puede adaptarse según las necesidades.

```
sergio@ubuntu:~$ ./mongodb-linux-i686-2.0.4/bin/mongo notas
MongoDB shell version: 2.0.4
connecting to: notas
> db.evaluacion.find()
{ "_id" : ObjectId("4fba61249cf4387e00000015"), "nombre" : "fp2", "apartado" : "
compilacion", "nota" : 10 }
{ "_id" : ObjectId("4fba61d89cf4387e00000024"), "nombre" : "xxx", "apartado" : "
compilacion", "nota" : 10 }
{ "_id" : ObjectId("4fba62859cf4387e0000002c"), "nombre" : "xxx", "apartado" : "
ejecucion", "prueba" : "agenda00", "nota" : 10 }
> _
```

Figura 55: Ejemplo de subida de notas. Se muestran para el caso de Compilación y Ejecución

7.4.2.2.2 Caso de error

Si el archivo de diferencias, diffmakefilexxxagenda00 contenía datos, es porque había errores en el código. Sabiendo esto, se le envía el fichero anteriormente mencionado al alumno para que pueda revisar su contenido y modificar su código.

Si el error se ha debido a una mala programación, recibirá un archivo de errores similar al que se muestra:

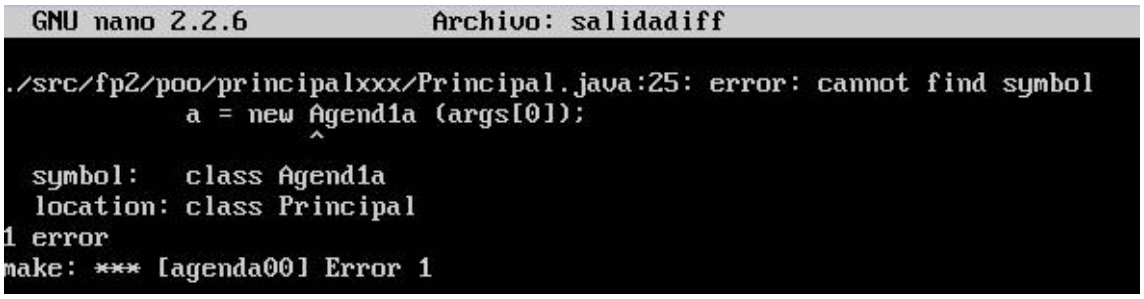


Figura 56: Error debido a fallos en la programación del código

Si por el contrario, si no existen fallos de código, pero sí ha fallado en la ejecución de las pruebas, (lo cual se debe a un mal uso de las interfaces proporcionadas), se le mostrará un fichero de errores en el que se detallan las diferencias entre la salida esperada, y la salida que generó su código.

Figura 57: Fallo debido a no haber insertado el número del contacto

Una vez llegados al final de la ejecución del ciclo, se marcará el final del mismo y se queda a la espera permanente de nuevas conexiones, iniciándose todo el proceso de nuevo y repitiendo los pasos una y otra vez.

Para más detalles sobre el código pueden visitar la sección anexo 4: Código Herramienta ServidorPFC.

Recapitulando lo visto en el capítulo 7, se ha desarrollado una herramienta informática para trabajar de forma remota que permite la evaluación autónoma de los alumnos, e-Assessment. Esta herramienta se ha desarrollado en lenguaje C, creando dos aplicaciones que dialogan entre sí, una desde el equipo del alumno, y la otra desde una máquina fija que escuchará permanentemente conexiones. En este diálogo se pasan ficheros a uno y otro lado, para la compilación de código, para su ejecución, o archivos de errores para comprobar dónde están los posibles fallos del alumno de cara a eventual corrección. El trabajo con estos ficheros y el almacenamiento de la evaluación individual se llevará a cabo con la ayuda de la base de datos MongoDB, cuyas marcas en el test de rendimiento del capítulo 6 justificaron su uso para esta aplicación.