

Universidad Central de Venezuela
Facultad de Ciencias
Escuela de computación

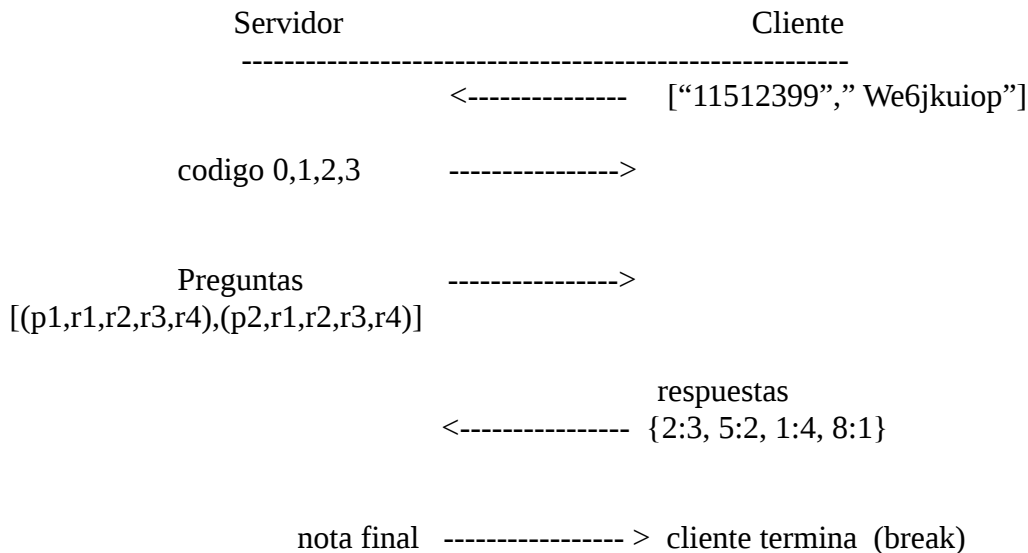
Caracas 20 Septiembre 2019

Descripción de Actividades: Realización de exámenes con sockets
Sistemas Distribuídos

Klaus Bertok
CI 11512399

Flujo de mensajes entre cliente y servidor

El siguiente es el flujo de mensajes que deberían de seguir entre el cliente y el servidor para la realización de un examen.



Servidor sigue funcionando

En la primera linea el cliente envía una lista con dos elementos, el login y el password que es la cadena que se genera aleatoriamente en el momento del registro del aspirante.

De ahora en adelante el servidor tiene identificado al cliente.

Ahora el servidor envía un código, si es 1 es porque no encuentra la cedula, entonces el cliente emite el mensaje correspondiente y se sale.

Si el codigo recibido es 2, es porque la cedula y el password estan OK pero el aspirante ya presento la prueba y si el codigo es 3, cedula ok pero password erroneo.

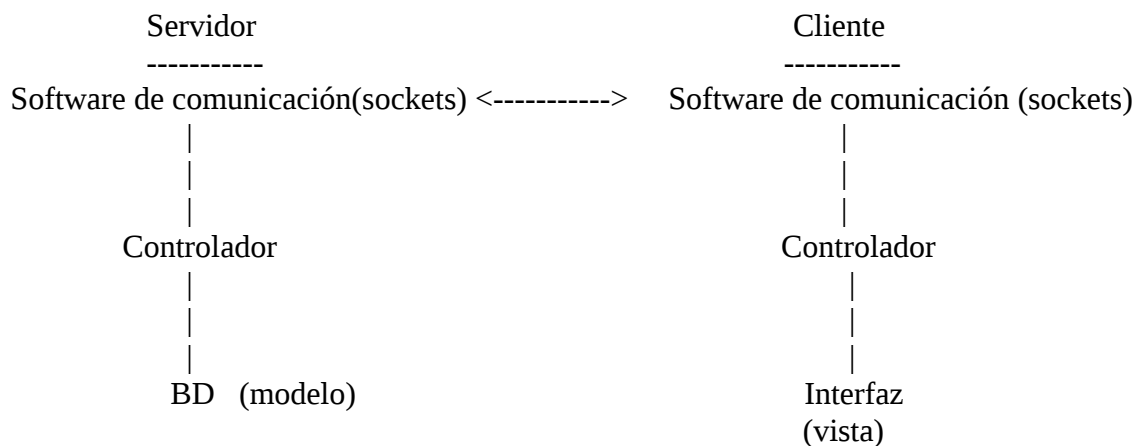
Si el codigo es 0, significa que las credenciales son correctas y el aspirante no ha presentado, entonces el cliente se prepara a recibir las preguntas que son enviadas por el servidor.

Una vez respondido el examen por parte del cliente, las respuestas son guardadas e un diccionario y enviadas al servidor, este recibe el diccionario y consulta con la base de datos para ver la cantidad de respuestas acertadas, como ya tiene la cedula y las respuestas, actualiza la base de datos, indicandole que el usuario ya presento y el resultado del examen, tambien inserta la fecha y la hora, adicionalmente genera un archivo binario con la cedula, el diccionario de preguntas y respuestas y la fecha y la hora y lo guarda en el file system para dejar constancia de la presentacion del examen y finalmente envia un mensaje con la nota del examen que es impreso por el cliente para finalmente salir de la aplicación.

Arquitectura de la aplicación

En cuanto a la arquitectura de la aplicación, se han intentado separar las funciones de comunicación, de las de acceso a datos y de las de presentación de la información.

Posible arquitectura de la aplicación:



En este caso, el software de comunicación, actúa como controlador llamando a las funciones de acceso a datos (modelo) las de comunicación (envío, recepción y conversión de datos) y las de presentación (resolución del examen)

Adicionalmente hay un programa independiente , adminBD que es el que permite realizar el registro, el listado completo o por cédula y el borrado de estudiantes.

Cabe destacar que esta funcionalidad descrita, se puede hacer con sqllitebrowser, pero para los efectos de probar la aplicación es suficiente con adminBD.

La intención original, ha sido la de implementar, toda la funcionalidad requerida, haciendo uso de librerías nativas o que vienen con la instalación por defecto de python, llamense, tkinter, threading, select, sqlite, etc... y no tener que instalar programas de terceros ni librerías que nos evitarían programar los sockets directamente, no se instaló mysql ni postgres, se uso sqlite, cuyas librerías y acceso ya viene con python .

Instalar un conector que permita programar estilo web, evitando los problemas de comunicación a bajo nivel ha podido ser una buena opción, pero la premisa fundamental es realizar la programación con sockets puros. Y es entonces cuando uno se da cuenta de lo importante que son los protocolos de aplicación y el middleware de los frameworks de programación web o librerías de comunicación asíncrona como twisted y de lo duro que es manejarlo todo a nivel de sockets, a pesar de que el concepto se ve sencillo.

El servidor de sockets

De acuerdo a lo investigado, existen principalmente dos alternativas, usar threads, que fue el enfoque implementado, o usar programación síncrona, llámese select, selectors, asyncio, twisted. Existe una tercera aproximación, que es usar los dos enfoques simultáneamente, threads y select ya que no son excluyentes.

El servidor, acepta conexiones y lanza un thread para atender cada conexión, aunque el programa solo fue probado con dos clientes realizando dos conexiones simultáneas y realizando la prueba correspondiente.

El cliente de sockets

El cliente de sockets, es un cliente muy básico, que crea el socket, se conecta y comienza la comunicación con el servidor, actualmente la comunicación solo está habilitada para conexiones locales, 127.0.0.1 puerto 8080.

Es importante destacar que existen dos tipos de mensajes, los mensajes de tipo string, para lo cual se usa la función enviar_mensaje y recibir_mensaje, y el envío de objetos como diccionarios o listas para lo cual se usan las funciones enviar_objeto y recibir_objeto, estas dos últimas funciones usan a pickle para serializar el envío y recepción de objetos.

El envío y recepción de mensajes y objetos está ampliamente comentada tanto en el cliente, como en el servidor.

Pickle también es usado, pero para serializar una lista y guardarla en un archivo, este es el paso en el que se guarda la información del aspirante, cédula, fecha y diccionario con preguntas y respuestas con el nombre de la cédula del aspirante en el punto final antes de enviar las respuestas al cliente.

Como probar la aplicación

Si estamos en la misma máquina, primero lanzar servidor10.py, después lanzar cliente10.py. El cliente nos pedirá una cédula y un password, para ello podemos tener un tercer terminal y lanzar AdminBd.py y ejecutar la opción 2 de listar todos los estudiantes, observemos uno que tenga los campos Presento, Fecha, Nota vacíos, utilizaremos la cédula y el password que estén allí, enseguida el servidor nos enviará las preguntas aleatorias, contestaremos, nos enviará la nota y podremos comprobar que los datos del usuario han sido actualizados en la base de datos y que un archivo con la cédula del aspirante ha sido generado, si intentamos ingresar nuevamente con las credenciales del aspirante nos dirá que ya presentó y nos sacará del cliente puesto que ya presentó.

En cuanto a la fase de presentación de la prueba, la interfaz es orientada a caracteres, aunque ya se comentó que la intención es usar tkinter y proporcionar una interfaz gráfica, el programa tiene un temporizador regresivo que aparece junto al prompt, si contestamos la última pregunta antes que termine el temporizador o pulsamos s para salir, el cliente enviará el diccionario con preguntas y respuestas pero el temporizador seguirá ejecutándose hasta el final.

Si el temporizador termina antes de contestar todas las preguntas, el prompt no vuelve a salir porque quien lo despliega es el temporizador y este ya terminó, entonces o contestamos esta última pregunta presionando 1,2,3 o 4 o pulsamos s para salir, y en ese caso el cliente enviará el diccionario con las preguntas y respuestas hasta las que el temporizador nos haya permitido antes de terminar.

Al parecer el programa principal no puede terminar a un thread ya arrancado, una posibilidad de mejora podría ser usar procesos en vez de threads y ver si con un proceso en el que se encuentra el temporizador regresivo se puede enviar un kill desde el programa principal, devolviendo el control al programa principal.

En cuanto al manejo de la seguridad

Las restricciones de acceso de maquinas por IP, se manejan con una lista de python que se encuentra en el archivo servidor_configuracion, IPSAUTORIZADAS

La linea de código que hace esto en servidor10.py es:

```
if direccionCliente[0] not in IPSAUTORIZADAS: #and fechas_horario_permitido():
```

Este chequeo se realiza despues que el servidor acepta una conexión, entonces se analiza si el componente de la direccion no se encuentra en la lista de ips aceptadas en ese caso el servidor cierra la conexión con el cliente, el cliente pedirá la cedula y el password y cuando intente enviarlos notará que la conexión por el lado del servidor se ha cerrado y saldrá de la aplicación.

Lo otro que se comprueba es que la conexión se haga en las fechas y horas habilitadas para ello, para este fin se usa la funcion, fechas_horario_permitido(), que tiene configurada una fecha inicial, una fecha final, un horario inicial y un horario final, si en el momento en el que se realiza la aceptacion de la conexión está dentro de esos parámetros, se seguirá con el flujo del programa, sino se cerrará la conexión como en el caso de las ip no autorizadas, por el momento la linea esta comentada para configurar la funcion con los párametros requeridos, la funcion ha sido probada ya.

De haber continuado con la implementacion de la interfaz grafica y los problemas presentados:

Algunos aspectos a tomar en cuenta, no es conveniente implemetar la GUI (cliente) de manera procedimental, ya que su código no se ejecuta linealmente, es la naturaleza de la programacion basada en eventos, lo mismo sucede con el software de comunicación, es de naturaleza basada en eventos, se recibe una conexión, llega un mensaje, se cierra la conexión, son todos eventos. Se pulsa un boton, se llena un campo, son eventos, lo conveniente es usar clases.

Y ello sucede porque hay que usar objetos globales de alguna clase para almacenar los datos, este seria el modelo, podria ser una variable global o un objeto global o una base de datos. En cualquier caso debe existir algun tipo de globalidad.

El problema encontrado usando interfaz gráfica, es como conectar el codigo de comunicación de sockets de bajo nivel con el codigo que accede a la base de datos, o que active una ventana que haga algo dentro de la interfaz

Claramente tenemos tres tipos de código, el código de comunicación, el código, de acceso a base de datos y el código que nos presenta la interfaz de usuario.

Los archivos de configuracion

Se ha intentado en lo posible que los parámetros estén concentrados en archivos de configuración en `servidor_configuracion` y `cliente_configuracion`, en el primero están las variables de IP permitidas, tamaños de los passwords a la hora de registrarse si los passwords contendrán números y caracteres especiales, la base de datos, número de preguntas totales y número de preguntas a escoger, para el caso del temporizador para realizar el examen se configura en `cliente_configuracion`, HORAS, MINUTOS y SEGUNDOS

La base de datos

Hay dos tablas, estudiante y pregunta, sqlite no tiene datos de tipo fecha ni booleanos, por lo que ambos deben ser simulados.

La tabla pregunta tiene los campos: Numero, Pregunta, Respuesta1, Respuesta2, Respuesta3, Respuesta4 y Respuesta. Donde Numero es la clave primaria, el servidor envía todos los campos al cliente excepto el campo Respuesta, que es el campo que el servidor chequeará a la hora de recibir los resultados, actualizar los resultados y notificar al cliente.

La tabla estudiante tiene los campos: Nombre, Apellido, Cedula, Password, Presento, Resultado y Fecha, al registrarse (`adminBD.py`), se llenan los campos Nombre, Cedula (clave primaria) no pueden haber dos campos iguales con la misma cedula. Apellido y se genera una clave aleatoria que se guarda en el campo password, esa clave se usará en el momento de presentar la prueba.

Al realizar la prueba el programa automáticamente llena los campos Presento le coloca un Si, puesto que sqlite no tiene datos booleanos hay que simularlo, y se coloca la Fecha como un string, puesto que sqlite tampoco tiene datos de tipo fecha y la nota se coloca x/y donde x es la cantidad de preguntas acertadas y y es el número total de preguntas.

Programas y pruebas individuales

Cada una de las funciones tienen la construcción: `if __name__=="__main__":`

lo que hay después de allí solo lo ejecuta el mismo programa y sirve para realizar pruebas unitarias, cada uno tiene datos de prueba, que permiten probar individualmente cada funcionalidad sin tener que ejecutar el programa completo.

Los puntos de mejora son muchos:

En el contexto de la utilización de sockets como herramienta de comunicación entre procesos y siguiendo con la intención de usar una interfaz gráfica (tkinter) que puede trabajar con, clases, e incorporar sockets, threads y acceso a datos, en la carpeta del cliente, *cliente_grafico_prototipo*, se encuentra *cliente_grafico_examen.py* que realiza un examen y obtiene su puntuación simulando la base de datos de preguntas con un archivo plano: `cliente_preguntas`,

La interfaz actual todavía presenta muchas deficiencias, debido a que es necesario dominar correctamente tkinter para adecuar y controlar correctamente los layouts y el comportamiento en general a través de su correcta programación.

Agregar el icono y el logo correspondientes.

Impedir el cierre de la ventana X

Impedir redimensionar la ventana principal (ya fue configurado).

Ya están programados dos threads uno para la hora actual y otra para el contador regresivo. Para que la interfaz no se ponga lenta pues ambos son repetitivos.
Configurar el look and feel de la interfaz (colores, letras, tamaños) haciendo uso del archivo configuracion_cliente.py

Otro punto de mejora es el manejo de errores y excepciones, centralizando los mensajes en un archivo servidor_excepciones y cliente_excepciones respectivamente.

Otro punto de mejora, pudiera ser diseñar una base de datos, que permita manejar distintas pruebas y tipos de distintas facultades, usar postgres que permite muchas conexiones simultaneas y no sqlite que es para ser accedido por el dispositivo en el que se encuentra embebido.

Otro punto de mejora, es el tratamiento del envio y recepcion de mensajes a nivel de comunicación, llamense los send y los recv, para que usen encabezados fijos y armado del mensaje, chequeando que el que envia, haya enviado todo el mensaje completamente y el que lo recibe lo haga tambien por completo, haciendo el tratamiento adecuado a nivel de excepciones.
Cuando la comunicación es probada a nivel de localhost, todo funciona bien, porque la transmision se hace internamente y no hay perdida de mensajes ni de paquetes, pero la realidad es distinta, en la red siempre se presentaran errores.

Otro punto de mejora seria la utilización de ssl sockets en vez de sockets.
Y con toda certeza y seguridad hay muchos puntos de mejora más.