Universidad Central de Venezuela
Facultad de Ciencias
Escuela de computación

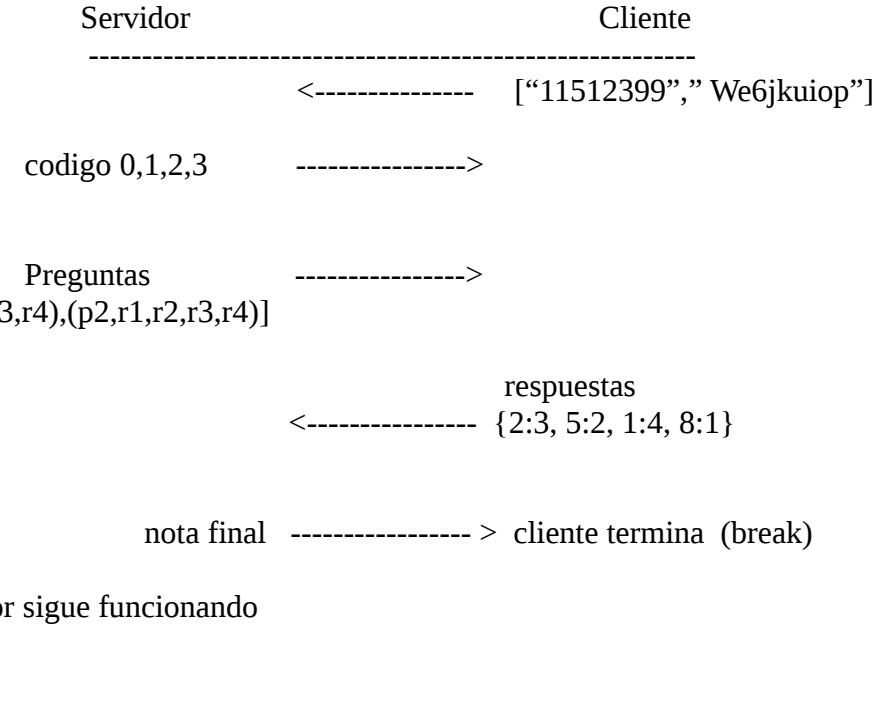
Caracas 20 Septiembre 2019
Modificación 28 Octubre 2019

Descripción de Actividades: Realización de examenes 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.



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 porque no encuetra 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.

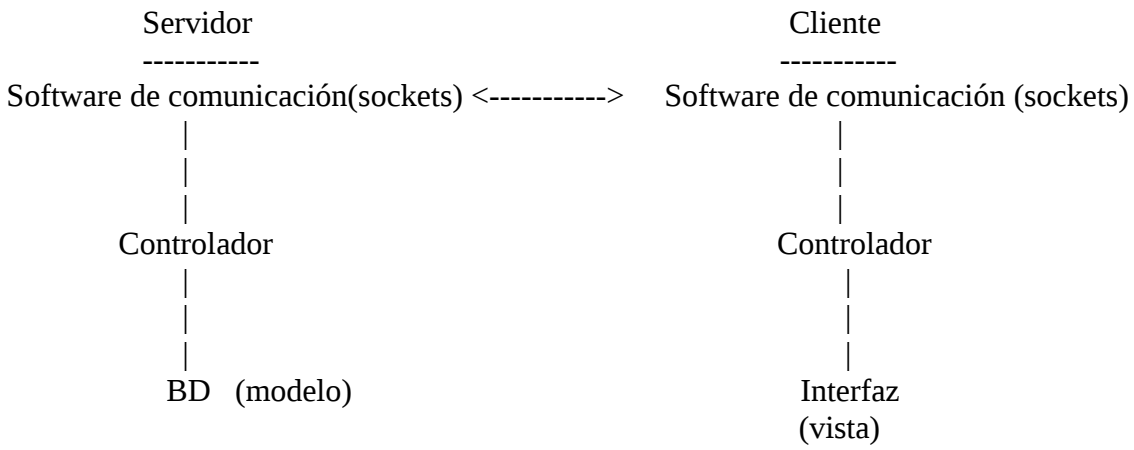
Si el codigo es 0, significa que las credenciales son correctas y el aspirante no ha presentado, entonces el cliente se prepara a recibir las preguntas que son enviadas por el servidor.

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 (envio, 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 sqlitebrowser, 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 librerias 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 librerias 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, llamese select, selectors, asyncio, twisted. Existe una tercera aproximación, que es usar los dos enfoques simultaneamente, 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 simultaneas 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 funcion enviar_mensaje y recibir_mensaje, y el envio de objetos como diccionarios o listas para lo cual se usan las funciones enviar_objeto y recibir_objeto, estas dos ultimas funciones usan a pickle para serializar el envio y recepción de objetos.

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

Pickle tambien 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, despues 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 opcion 2 de listar todos los estudiantes, observemos uno que tenga los campos Presento, Fecha, Nota vacios, 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 cedula 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 presentacion de la prueba, la interfaz es orientada a caracteres, aunque ya se comentó que la intencion 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 ejecutandose 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 ultima 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 esten concentrados en archivos de configuracion en servidor_configuracion y cliente_configuracion, en el primero estan las variables de Ip permitidas, tamaños de los passwords a la hora de registrarse si los password contendran numeros y caracteres especiales, la base de datos, numero de preguntas totales y numero 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 envia todos los campos al cliente excepto el Campo Respuesta, que es el campo que el servidor chequeara 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 usara en el momento de presentar la prueba.
Al realizar la prueba el programa automaticamente 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 cantitad de preguntas acertadas y y es el numero total de preguntas.

Programas y pruebas individuales
Cada una de las funciones tienen la construccion: if __name__=="__main__":
lo que hay despues de alli solo lo ejecuta el mismo programa y sirve par 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 interface grafica (tkinter) que puede trabajar con, clases, e incorporar sockets, threads y acceso a datos, en la carpeta del cliente, *cliente_grafico_prototipo*, se encuentra c*liente_grafico_examen.py* que realiza un examen y obtiene su puntuacion simulando la base de datos de preguntas con un archivo plano: cliente_preguntas,

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

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.

Segunda parte del proyecto:

Comunicación y consistencia entre dos servidores:

En esta segunda fase del problema de la automatización de la presentación de examenes se incorpora un segundo servidor, que debe mantenerse comunicado con el otro servidor (ambos nodos).

La primera fase del problema, esta descrita dentro de la carpeta: **un_servidor** en el correspondiente informe.

En esta segunda parte, un servidor o nodo, debe saber si un aspirante se inscribió en el otro nodo e impedir que presente examen en un nodo en el que no se haya inscrito, si previamente ya se inscribió o presentó en el otro nodo. Con esto se impide que pueda presentar la prueba dos veces en los dos nodos.

En la primera fase del desarrollo de una solución para este problema, se implementó un cliente y un servidor y dicha comunicación era síncrona, sin embargo en esta segunda fase la comunicación entre los dos servidores se implementa de manera asíncrona, la razón de ello, es que se desconoce el orden y la cantidad de aspirantes que se inscribirán en un nodo o en otro, en un momento dado. Por lo que necesariamente dicha comunicación debe ser asíncrona (usando select).

La manera de implementar la solución propuesta fue desglosando el problema de la comunicación en problemas mas pequeños pero que estuvieran alineados a la solución del problema global que es lograr una comunicación para poder preservar una consistencia de la información que manejan los dos servidores en todo momento.

- Comunicación entre los servidores usando select.

- Utilización de datos comunes (memoria compartida) usando colas y escribiendo en ellas de manera simétrica y consistente, cada servidor (nodo).
- implementacion en cada servidor de la interpretacion de los datos que genera y que recibe (algoritmo) para preservar la consistencia.

Al resolver cada problema por separado y finalmente integrar todo, se puede llegar a una solución del problema global, programar toda la solución de una sola vez, es complicado y confuso.

El código esta compuesto por las siguientes carpetas:

psd la carpeta principal.
un_servidor: primera parte del proyecto con un cliente y un servidor
dos_servidores: segunda parte del proyecto, que integra a la primera, esta compuesta por dos clientes, dos servidores y dos bases de datos, en ella está integrada, la primera fase, la comunicación de los dos servidores, las colas y la lógica de comunicación (algoritmo) de comunicación entre los servidores.
test_dos_servidores: es la carpeta que contiene un prototipo, previa a la solución integrada (dos_servidores) con listas en vez de bd para implementar el subproblema de colas y comunicación de los servidores, esta carpeta solo se usó para realizar pruebas preliminares antes de implementar la solucion integrada en la carpeta **dos_servidores**.

Posibles mejoras:
Ya que se intentó aprovechar el codigó de la primera fase, tenemos que la comunicación entre cada servidor y sus clientes es síncrona y la comunicación entre los servidores es asíncrona, se puede implementar una solución con clientes que sean atendidos por threads y usen comunicación asíncrona también.

A la hora de presentar examen opción 2, se pide la cédula, y luego vuelve a pedir la cédula y el string aleatorio para validar credenciales, después permite presentar el examen en caso de que no se haya presentado o indicar que ya se presentó si ya se hizo, lo normal es que pida la cédula una sola vez, esto se hizo para no tener que reestructurar el código de la primera parte y poder aprovecharlo.

Si se quiere usar los programas desde distintas máquinas, es necesario colocar las Ips de los servidores y de los clientes y usar un sistema de nombrado o bien colocando las Ips en los /etc/hosts de cada máquina o bien realizando esto ultimo en el DNS interno o bien usando DNSmasq para que las máquinas se puedan localizar.

Como probar los programas:

Se deben abrir cuatro terminales:
Arriba estarán los dos servidores y abajo los clientes.

Terminal 2, ejecutar este despues de 1

./psd/dos_servidores/>python s**ervidor4321c1234.py**

Terminal 1, ejecutar este primero

./psd/dos_servidores/> python **servidor1234.py**

Terminal 3

Terminal 4

./psd/dos_servidores/>python **cliente4321.py** ./psd/dos_servidores/> python **cliente1234.py**

El terminal 1 debe ejecutarse primero ya que el servidor del terminal 2 tiene el cliente que se conecta al servidor del terminal 1, los terminales 3 y 4 (los clientes) se puede ejecutar en cualquier orden o solo el terminal 3 y despues el terminal 4, o viceversa, la única restricción es leventar el terminal 1 primero y despues el terminal 2
para que el cliente del terminal 2 se puede conectar al servidor del terminal 1 y poder establecer la comunicación.

Porque esos nombres en los archivos, servidor4321c1234.py, indica que es el servidor que escucha por el puerto 4321 y tiene un cliente que establece un canal de comunicación donde se encuentra el servidor que escucha por el puerto 1234.

servidor1234.py indica que es el servidor que escucha por el puerto 1234.

Aunque se usó cliente4321.py y cliente1234.py la única diferencia es el puerto al cual se conectan, ya que el código es idéntico para el caso de los clientes.

Cómo funciona la comunicación entre los servidores:

en la carpeta ./psd/dos_servidores_test se hizo un prototipo que en vez de realizar actualizaciones de campos en la base de datos (carpeta dos_servidores) usa listas que almacenan las cédulas y permiten comprobar casos de prueba, todos ellos están implementados en la carpeta (dos_servidores)

Cada cliente que se conecta a un servidor, puede hacer dos cosas, registrarse o presentar el examen en caso de que no lo haya presentado, entonces:
Si la opción es registro, (se usan nombres simbólicos de los Nodos para hacer más amigable la programación, nodo 1 es Caracas (servidor4321c1234.py) y nodo 2 es Maracay (servidor1234.py)
Lo primero que hace un cliente al conectarse es actualizar las colas de entrada, cada elemento de la cola de entrada es un elemento del otro servidor, es decir es alguien que se inscribió en el otro servidor y le notifica a este este hecho al otro servidor.

Si la cédula esta en el otro nodo, le indicará que debe presentar alla.
Si no, colocará esta cédula en la cola de salida para que el otro nodo sepa que se inscribió en este nodo, sino ocurre nada de eso es porque ya está inscrito en este nodo y no colocará nada en la cola de salida.

Con la opcion 2 de presentar examen es algo similar, si no está registrado en el otro nodo, y si está registrado en este nodo podrá comenzar a presentar el examen, de aquí en adelante se ejecuta el código de la primera parte del proyecto, se le preguntará nuevamente la cédula y el string aleatorio y procederá a realizar la prueba, se le enviará el resultado y se actualizará la BD, tambien se escribe en el directorio donde se ejecuta el servidor un archivo cuyo nombre es la cedula del aspirante y cuyo contenido es la cédula y un diccionario con las preguntas y respuestas para dejar una constancia mas detallada de que presento el examen.

El papel de las estructuras de datos y la comunicación de los servidores:

Cada servidor accede a una cola de entrada y a una cola de salida, cada vez que un aspirante se inscribe en un nodo, se le de la cola de entrada y se actualiza la BD y se escribe en la cola de salida para notificar al otro servidor.

Para comunicar a los dos servidores, el cliente1234, leera de su q_saliente y enviara un send para que el servidor1234 esciba en su q_entrante.

El servidor 1234 leera de su q_saliente y enviara send a cliente1234 para que este escriba en su q_entrante.

De esta manera los dos servidores se mantendrán comunicados en todo momento.

Algunos casos de prueba:
cliente se inscribe en nodo, intenta inscribirse en el otro nodo.
cliente se inscribe en nodo, presenta en nodo, e intenta presentar nuevamente en el mismo nodo o inscribirse o presentar en el otro nodo.
Varios clientes se inscriben en un nodo y cualquiera de ellos intenta inscribirse en el otro nodo o presentar en el otro nodo.
Varios clientes se inscriben alternativamente en ambos nodos y cualquiera de ellos intenta inscribirse o presentar en el otro nodo donde no se habia inscrito o habia presentado.
Se abren dos a mas clientes conectados a un servidor o a los dos servidores y se comprueba la concurrencia y la consistencia.