Práctica 2

Vilchis Domínguez Miguel Alonso

1. Contexto:

Alice y Bob trabajan en la compañía X, por políticas de la empresa se bloqueó el acceso a internet, y se trabaja con una red interna. Alice y Bob son muy buenos amigos y quieren platicar todo el tiempo por lo que te pidieron que hicieras una aplicación que funcione como chat, para que se mantengan en contacto.La red en la que se encuentran su IP es fija, tu aplicación puede hacer uso de esa información.

2. Objetivo de la práctica

Que los alumnos logren comprender y aplicar el concepto de llamadas a procedimientos remotos a través de una arquitectura cliente-servidor. Por medio del desarrollo de una aplicación que tiene como fin fungir como un cliente de un chat y ofrecerá diversos servicios (para esta práctica, recepción de texto) por medio de los cuales modificará el estado del sistema (despliegue del texto en la interfaz). Y a su vez, permitirá hacer usos de los servicios que otro servidor ofrezca para transmitir texto.

3. Detalles generales

Es la primer práctica del proyecto que se irá construyendo durante el semestre, por ahora los objetivos son: Mandar texto entre clientes con ayuda de la biblioteca xmlrpc y con conexiones a ubicaciones fijas, debe funcionar tanto para equipos dentro de la misma red, como para dos instancias del programa ejecutandose en la misma computadora, en donde cada instancia cuenta con un puerto diferente.

4. Especificaciones:

- El alcance de ésta práctica se limita a intercambiar mensajes de texto entre dos computadoras que se encuentran en la misma red.
- La manera en la que se creará la interconexión entre los usuarios será usando la biblioteca xmlrpc.
- Es imporante notar que xmlrpc define dos roles, servidor y cliente, en este rol el cliente hace llamadas a procedimientos remotos y el servidor ejecuta lo referente a ese procedimiento remoto y regresa una respuesta.
  Deben de mantener en mente ese diseño, cuando programen su aplicación.
- El puerto para las conexiones será el 5000 si se está trabajando con computadoras distintas.

- La estructura del proyecto se encuentra dividida de la siguiente manera:

```
|
|– –Constants
                |– –Constants.py
                |– –AuxiliarFunctions.py
|– –GUI
                |– –LoginWindow.py
                |– –ChatWindow.py
|– –Channel
                |– –ApiServer.py
                |– –ApiClient.py
                |– –Channel.py
|– –GraphicalUserInterface.py
```

  - *Carpeta Constants:* Dicha carpeta contiene el módulo Constants.py en donde se deben colocar todas las constantes que se utilicen a lo largo del proyecto, con el fin de evitar número mágicos y fomentar la legibilidad del código.
    AuxiliarFunctions: En este archivo encontraran funciones auxiliares cuyo propósito es facilitar el desarrollo del sistema.
  - *Carpeta GUI:* En esta carpeta se desarrollaran el código referente a las interfaces gráficas a través de las cuales el usuario interactuará con el sistema. Deben desarrollara por completo. Al ejecutar el archivo main de esta práctica se desplegará una ventana, la cual, si se está trabajando de manera local, pedirá el puerto del cliente y el puerto del contacto. Seguido de esto mostrará una ventana de chat para permitir la comunicación. En el caso de trabajar con instancias en distintas computadoras solo se preguntara por el campo ip del contacto, y seguido de esto se mostrara la ventana del chat.

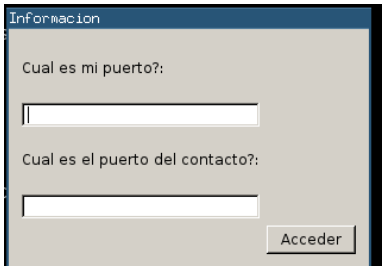

Figura 1: Imagen que muestra la etapa de Login para uso local

Figura 2: Imagen que muestra la ventana del chat despues del login

  - *Carpeta Channel:* En este sitio se implementará todo lo referente a los procedimientos remotos con los que el sistema funcionará. La definición en particular de los archivos es el siguiente:

- *Channel*: Este archivo contiene la clase Channel, el objetivo de esta clase es organizar los threads que darán soporte a los diferentes servicios que se irán agregando en el chat, el primero de ellos es el thread para dar servicio al servidor de xmlrpc que del cual hara uso el contacto con el que nos estámos comunicando. Es importante notar que en esta clase solo se declaran e inician los threads. La definición del servidor y cliente de xmlrpc como clase se realizará en los siguientes archivos.
- *ApiServer* : En este archivo se define la clase MyApiServer, la cual representa el servidor de procedimientos remotos de este cliente. A través de la definición de los métodos que se encuentren en la clase FunctionWrapper, otro cliente de chat podrá mandarle datos a este cliente. Hint: Se debe crear un thread cuyo target sea una instancia de la clase MyApiServer en la clase Channel.
- *ApiClient*: En este archivo se define la clase MyApiClient, el cual define la instancia de xmlrpc que tendrá contacto con el Servidor xmlrpc del cliente con el que estamos hablando.

- *GraphicalUserInterface.py*: Este archivo tiene como propósito funcionar como main, es decir el usuario ejecutar $\$pyhtonGraphicalUserInterface.py$ se debe de desplegar su ventana de login, con la instancia que soporta la comunicación entre dos computadoras, es decir, solicitando el la ip del contacto. Si el usuario ejecuta $\$pyhtonGraphicalUserInterface.py \quad -l$ entonces se debe de preguntar por el puerto del cliente y el puerto del contacto para poder tener dos instancias dentro de la misma computadora trabajando ambas en localhost.

- Se les proporcionará una carpeta con la estructura ya descrita, para esta práctica .

5. Preguntas para el reporte:

- Define el concepto de procedimientos remotos.
- ¿Qué es una IP y para qué sirve?
- A grandes rasgos define lo que es un puerto.
- ¿Se puede tener dos clientes en una misma computadora? Es decir, abrir una terminal y correr una instancia para Alice, y abrir otra terminal para correr una instancia para Bob y que estos se puedan comunicar.
- Además de las preguntas, se debe agregar un pequeño reporte de las particularidades de su código, como son:
  - Cual es el flujo del problema y para que sirve cada archivo dentro de la carpeta GUI y Channel
  - Principales problemas que se encontraron y cómo los solucionaron.
  - Problemas que no fueron solucionados.

6. Entrega

Fecha de entrega: 22 de Agosto del 2016

La entrega es por parejas, la práctica con la estructura mencionada debe estar almacenada en un depósito de github llamado Redes2017, en la rama Practica2, el readme debe contener el nombre de los 2 integrantes del equipo.

Deben mandar el link del depósito con el asunto *[Redes-17]Practica2* al siguiente correo:

mvilchis@ciencias.unam.mx

Nota: Basta con que un integrante del equipo suba su código al depósito.
Nota: A las 11:59 del díal de la entrega se descargará el contenido de los depósitos, para evaluación, asegurense que se encuentre su versión final para esa hora.