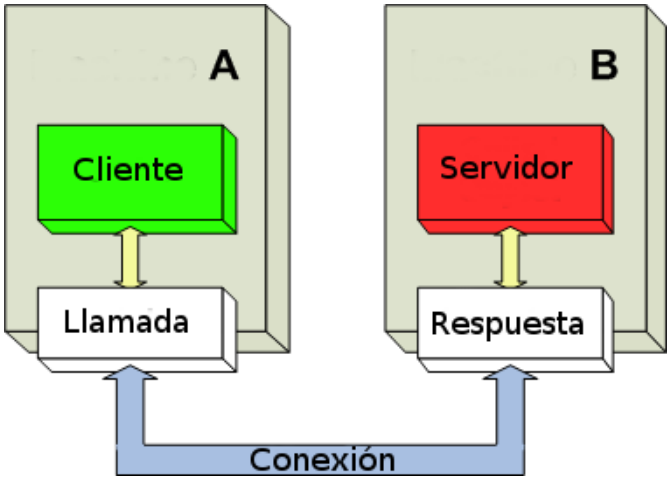
Reporte Práctica 2 [Redes 2017]

Nombre(s): Angie Daniela Velásquez Garzón

Bernal Cedillo Enrique Antonio

- **Procedimientos Remotos**

  Sistema de procesamiento distribuido donde los procedimientos pueden ser invocados por procesos de máquinas ajenas a la máquina dueña del proceso que alberga el procedimiento.
  Quien realiza la llamada al procedimiento (subrutina) es llamado cliente, y a quien ejecuta el procedimiento, servidor.
  Permite una comunicación transparente entre procesos, donde las invocaciones a los procedimientos se hacen como si fueran locales, ignorando los detalles explícitos relacionados a la interacción remota.



- **¿Qué es una IP y para qué sirve?**

  El protocolo de IP es uno de los más importantes en internet, especifica el formato de los paquetes que son enviados y recibidos a través de los routers y equipos finales a través de la red.
  Es un protocolo de comunicación de datos digitales que trabaja en la Capa de Red (de acuerdo al modelo OSI), de manera en que el único dato necesario para la entrega de paquetes hacia el destino es la dirección IP que se encuentra en la cabecera (header) de los paquetes.
  Sirve para la comunicación bidireccional por la cual se transfieren datos mediante un protocolo no orientado a conexión (Cada paquete se trata como unidades de información independiente, sin una comunicación continua entre equipos finales) transmitiendo paquetes conmutados.

- **¿Qué es un puerto?**

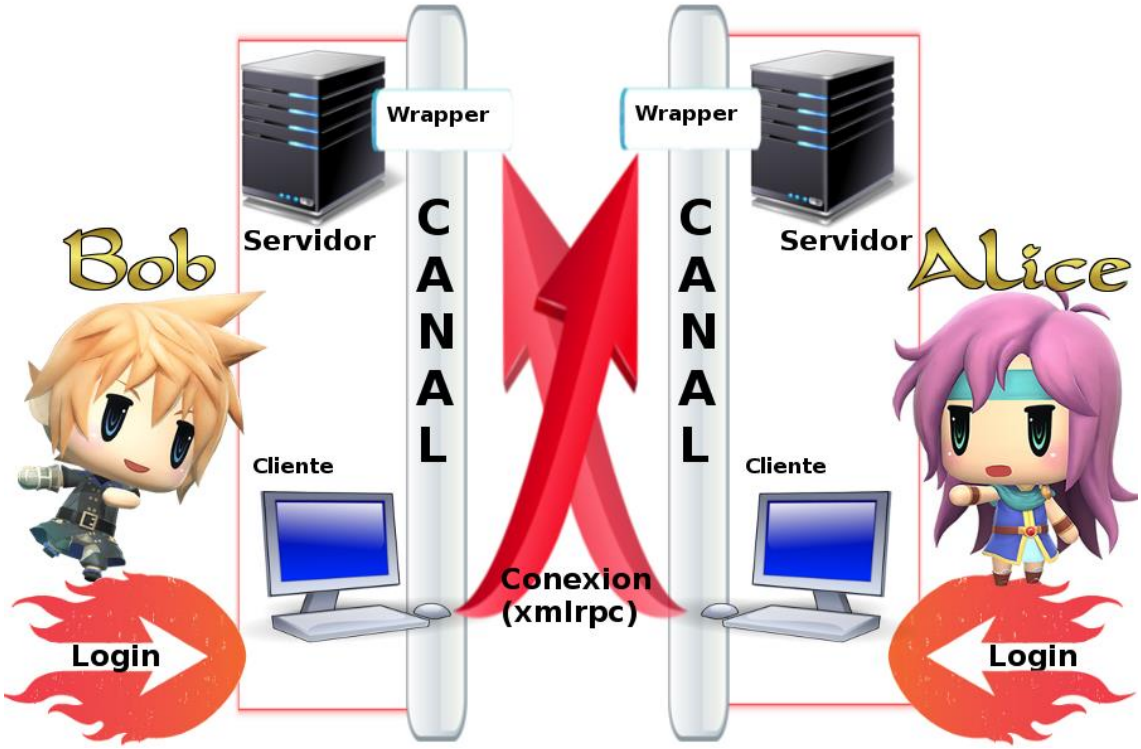
  Es la interfaz a partir de la cual se pueden enviar y recibir datos (Puede ser hardware/software).
  En el contexto de redes, es una construcción lógica que puede identificar un proceso o servicio de red. Siempre está asociado a una dirección IP y a un protocolo que se ubica en la capa de transporte (modelo OSI), administrando así el envio o re-ensamblaje de la información enviada/recibida a través de una red.
  Los números de puerto se indican por medio de 16 bits, por lo que existen 65,536 clasificados de la siguiente manera:
    - [0,1024) “Bien conocidos”, para uso exclusivo de protocolos conocidos.
    - [1024,49151] Pueden ser usados por cualquier aplicación.
    - [49152,65535] Se asignan de forma dinámica a las aplicaciones.

- **¿Se puede tener dos clientes en una misma computadora?**

  Sí, debido a que los puertos son identificados por medio del Número de Puerto, lo cual permite tener a más de uno en una misma computadora.
  Sin embargo, dos puertos con el mismo número no pueden estar activos al mismo tiempo.

Reporte código

- **Flujo del problema (programa)**



(Debido a que son 2 instancias idénticas, la explicación es la misma para ambas)

-) El programa comienza con la pantalla de Login, en la que se obtiene la información necesaria sobre puertos/login para realizar la conexión

-) Cuando se pasa a la pantalla de chat, el canal se encarga de iniciar los hilos con los procesos de cliente y servidor (Con MyApiServer y MyApiClient).

-) Si ambas instancias del programa introdujeron correctamente la información correspondiente, xmlrpc se encargará de realizar la conexión entre los puertos,

-) Gracias a esto es posible enviar peticiones de cliente, por medio del wrapper, que sean ejecutadas en el servidor del otro contacto (en este caso, imprimir cadenas en pantalla).

-) Al cerrarse el programa, los hilos del canal también son detenidos y termina la ejecución.

(Información específica por archivo en la siguiente pregunta)

- **Archivos carpeta GUI**

    - **ChatGUI.py**
      Clase en la que se realiza el manejo de la interfaz grafica para el chat de texto, es aquí donde iniciamos el canal necesario para la comunicación entre cliente/servidor (casi junto a la inicialización de elementos gráficos).
      En esta pantalla se puede ver la conversación completa con el contacto y enviar un mensaje nuevo hacia el contacto.

    - **LoginGUI.py**
      En este archivo se encuentra la clase LoginGUI en la que recibimos por medio de campos de texto los parámetros necesarios para establecer la conexión entre las 2 instancias del programa.
      Su comportamiento es diferente dependiendo de si la conexión será Local o Remota,

- **Archivos carpeta Channel**

    - **ApiClient.py**
      Es la clase requerida para representar un cliente, se crea el proxy necesario para comunicarse con el servidor del contacto cada vez que se envía un mensaje (con ayuda de xmlrpc).

    - **ApiServer.py**
      Sirve para crear el servidor que en conjunto con FunctionWrapper ejecutará los métodos de forma remota.

    - **Channel.py**
      Es el canal encargado de correr los 2 hilos necesarios para la comunicación con el otro contacto. (Un hilo para ApiServer y otro para ApiClient)

- **Principales problemas encontrados**

  La conexión en el chat debe establecerse de modo que para cada usuario primero se cargue el servidor, sin establecer la conexión del cliente puesto ambos usuarios no se conectan de forma simultánea, uno de los principales problemas que se encontró fue que si un usuario establece conexión pero no hay quien reciba dicho mensajes, la aplicación lanzaba un error y detenía su ejecución.
  Se optó por realizar una captura de excepciones donde el cliente por cada mensaje que intenta establecer la conexión al puerto o dirección ip indicado, en caso de no poder establecer la conexión significa que el otro usuario aún no se encuentra conectado y esto se le informa al usuario.

  Al finalizar un chat, el hilo del servicio, continuaba ejecutándose por un tiempo indefinido a menos de que se matara al proceso, desde fuera de la aplicación.

- **Problemas sin resolver**

  Warning que ocurre al envíar el primer mensaje a través del chat:

  “QObject::connect: Cannot queue arguments of type 'QTextCursor'

  (Make sure 'QTextCursor' is registered using qRegisterMetaType().)”

  Debe ser algo con los threads, o ChatGUI.py (interfaz grafica) con alguna cadena de texto. Sin embargo el chat funciona correctamente.
  Al parecer, esto se debe a modificar elementos de la interfaz en un hilo distinto al main.