***Proyecto***

***“Simulación de un***

***Sistema domótico en Python”***

Integrantes: → Tomas Ibaceta Rol: 201721039-7

→ Patricio Carrasco Rol: 201721064-8

→ Gustavo Silva Rol: 201721012-5

Ramo: → [ELO330] Programación de Sistemas

Fecha: → 14 – 12 – 2021

Contexto: → Se ha observado la necesidad de registrar datos en la nube es una necesidad común, debido al acceso remoto que logra. Dentro de este tópico es que alumnos del ELO330 de la UTFSM han desarrollado previamente una solución simplificada a este problema (Tarea 2 – ELO330 – 2021S2), es debido a ello que; se han planteado replicar el sistema medición en un nuevo lenguaje, este corresponde a Python. Este cambio del lenguaje es debido a que este ha estado en auge en el ultimo tiempo, debido a su nivel de abstracción de alto nivel que ofrece a sus usuarios y programadores.

De ello se establece la siguiente problemática, ¿Como puedo crear una arquitectura cliente/servidor en lenguaje de Python que sea concurrente?

De esta manera se plantea la siguiente estructura de programa que se comunicara mediante TCP/UDP con programas del mismo lenguaje (python); todos evidenciable en el siguiente diagrama.

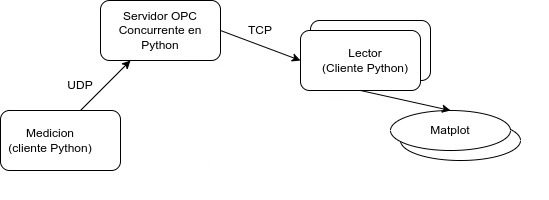


Figura 1: Diagrama de arquitectura de solución

En la figura 1 se puede observar de la grafica que existe un servidor OPC, este es el encargado de emular el servidor en la nube, administrando conexiones TCP con los distintos clientes y UDP con el único sensor que estamos observando. Es importante recalcar que hay distintas formas de realizar un servidor concurrente, uno de ellos corresponde a simularlo a través de la estructura ***select.***

Por otra parte, de la misma grafica se observa el programa Medicion.py, este es el encargado de crear una rampa de periodo 1 amplitud 1, enviando los valores de esa señal cada ***delta\_t*** [s] a opc.py mediante una conexión UDP.

Finalmente observamos un cliente, que corresponde al programa lector escrito en Python este se comunica mediante TCP con opc para leer el valor de un único sensor; logrando graficar a través de utilizar una librería llamada ***matplotlib*** junto con un buffer circular.

**Estrategia y Solución para resolver Problema:**

**Medidor.py:**

La función de este programa es crear un socket *UDP* y mandar datos periódicamente de una función determinada, para este caso son dientes de sierra de 50% de ciclo de trabajo.

La estrategia que se toma es pasar el nombre de *hostname* , *puerto* a usar en *aragorn* y el valor de *delta t* por argumentos al correr el ejecutable del programa; guardando cada una de las variables en el formato que corresponde. Una vez teniendo las variables se procede a crear el socket para UDP :

*sock = socket.socket(socket.AF\_INET, socket.SOCK\_DGRAM)*

y luego guardando las direcciones del server en la estructura *sockaddr\_in* como son el puerto en el que trabaja y su dirección de red (variables que se pasan al ejecutar el programa) .

Luego en un while(1) se generan constantemente valores de números flotantes de la función de diente de sierra y estos son mandados a través de la función *sendto* con dirección al server guardado en la estructura sockaddr\_in :

*sendData = str.encode(str(num))*

*sent = sock.sendto(sendData, server\_address)*

**lector.py:**

De igual forma que *medidor.py* la información del servidor *opc.c* se pasa por argumentos cuando se ejecuta el programa *lector.py ( hostname , numero de puerto aragorn y delta )* . Se almacena el hostname en una tupla junto con el puerto a usar, luego se crea el buffer para almacenar los 20 datos que corresponde con un arreglo de flotantes.

Finalmente, para comunicarse con el servidor opc dentro de un while infinito hasta que se usa la señal *SIGINT*; se crea el socket con el número del Puerto del servidor y se crean objetos para la comunicación entre servidor y cliente gracias a las funciones que posee el socket en python, como lo son los métodos ***recv*** que permite leer datos que le manda el servidor y el otro método es ***send*** que permite mandar datos al servidor. Luego, se vuelve a realizar la conexión del socket con el método ***connect***; para así mandar el carácter ´n´ para advertir al servidor que envíe datos a este cliente y este mismo almacena de a 20 datos en un buffer circular el cual dará paso al gráfico.

Para plotear ocupa la siguiente estructura:

fig = plt.figure()

…

fig.clear()

ax = plt.subplot()

ax.plot(np.asarray(xValues), np.asarray(yValues), linewidth=2.0)

plt.pause(0.02)

plt.show()

Lector ocupan un buffer circular para ordenar los datos que se caracteriza por ser ***FIFO***, es decir; cada vez que llega una nueva medición se agrega al final del buffer de 20 datos, pero para ello deben eliminar el primer dato del buffer a través del método ***pop*** y agregar al final del buffer con el método ***append***

**opc.py:**

Para el servidor solo se necesita como argumento el número de puerto en el que se desea trabajar, gracias a las características que posee un socket se puede trabajar con UDP y TCP en el mismo socket sin ningún tipo de problema, por lo que se crean los 2 socket :

socket\_TCP = socket.socket(socket.AF\_INET, socket.SOCK\_STREAM)

socket\_UDP = socket.socket(socket.AF\_INET, socket.SOCK\_DGRAM)

y haciendo el bind correspondiente a cada uno, con la dirección del servidor almacenada en la estructura de tipo sockaddr\_in. En el loop infinito realizado con while(isfinish == False) (con condiciones de excepción para finalizar con ctrl +C) se utiliza la función select() con un time out de 1 segundo para emular un servidor concurrente, de esta manera solo revisara las acciones de TCP o UDP si es que hubo movimiento en alguno de ellos. Si es que esto último sucede primero se analiza si el movimiento fue en una conexión TCP y si tiene que mandar los datos a un cliente en específico; y luego analiza si el movimiento fue en alguna conexión UDP para almacenar datos que llegan desde el medidor (que en algún momento puede ser enviado por la conexión TCP del socket al cliente que lo solicitó).

Es importante recalcar que el select funciona muy similar a como funciona en c, en este caso ya retorna 3 listas:

*read\_lists, write\_list, exception\_list = select.select(socket\_lists\_rd, socket\_lists\_rd, socket\_lists\_rd, timeout)*

donde en cada lista, ya se encuentran los descriptores de socket en que solamente hubo movimientos.

**Situaciones problemáticas del programa:**

Una de las situaciones problemáticas que ocurren el programa es que el medidor siempre genera una función de periodo 1 segundo, ya que nunca se indica que este cambie por lo que si se elige un delta muy grande lo más probable es que la gráfica realizada por el cliente lector no se distinga como corresponde además de que esta lee cada un segundo lo cual ocasiona que no se forme la señal como uno desee.

**Conclusiones:**

Es importante recalcar que este código se basa en una solución ya propuesta con anterioridad en una evaluación del curso ya mencionado. Esto hace que sea importante hacer notar una característica de este proyecto: “este proyecto se basa en una traducción de código”; lo cual no desmerece su desarrollo, esto dice que un lenguaje de alto nivel como python, puede reducir mucho lo que es código, pero el nivel de abstracción produce que se puedan perder detalles y eficiencia si es que no se entiende lo que se esta realizando por debajo; lo cual se consigue entendiendo como realizar el mismo código en lenguajes como C.

Además, este proyecto tiene muchas posibles mejoras, una de ellas es considerar y evaluar el comportamiento con versiones de estos programas en lenguajes diferentes a python. Por otra parte se puede reestructurar la solución para que se realice con multi-hebra.