**NOMBRES**:

LUIS FELIPE VELASCO TAO – JUAN DAVID GONZALES DIMATE

**Desarrollar los siguientes ejercicios de Sistemas distribuidos utilizando la estructura Cliente Servidor.**

1. Un individuo desea invertir su capital en un banco y desea saber cuánto dinero ganará después de un mes si el banco paga a razón de 2% mensual. Desarrollar una aplicación cliente-servidor, donde el banco es el servidor y la máquina cliente realizará el proceso para visualizar en pantalla la ganancia del individuo.
2. Una tienda ofrece un descuento del 15% sobre el total de la compra y un usuario desea saber cuánto deberá pagar finalmente por la compra de sus productos. Desarrollar una aplicación cliente-servidor, donde la tienda es el servidor que debe realizar el proceso de descuento sobre la compra y retornar al usuario para que visualice en pantalla lo que debe pagar el usuario por la compra.
3. En una empresa un empleado desea un préstamo de $1'000.000, la empresa realiza los préstamos de acuerdo con el sueldo del empleado, si el empleado obtiene un sueldo mayor a $3'500.000 la empresa le acepta el préstamo. El préstamo se debe pagar durante 6 cuotas con un interés del 1.2%. Visualizar en la interfaz gráfica del cliente, el total del préstamo y la cuota mensual.
4. Incluir un Cliente2 al sistema distribuido con arquitectura cliente servidor de forma tal que pueda realizar solicitudes.

**SOLUCION**

1. Para el ejercicio del banco y el préstamo definimos los siguientes scripts desarrollados en Python y el uso de sus librerías de Sockets e Hilos.

**Servidor**

import tkinter

import socket, threading

win = tkinter.Tk() # Crea la ventana principal

win.title('Servidor del banco') #nombre de la ventana

win.geometry("500x500") #Dimesiones de la ventana

users = {}#Diccionario de usuario, también puede conectarse a la base de datos

ip = "192.168.0.14" #IP del servidort

tablero = "" #Variable para almacenar los datos para el display

flag = False #variable para el control de la conectividad

port = 5000 #Puerto del socket

text = tkinter.Text(win, height=25, width=60)

labeltext = tkinter.Label(win, text='LOG DEL SERVIDOR', font=('Helvetica', 18, 'bold')).grid(row=0, column=0)

text.grid(row=1, column=0)

def run(ck, ca):

global flag

try:

userName = ck.recv(1024)# Aceptar como unidad la información enviada por el cliente en 1k. La información recibida aquí es de tipo byte

users[userName.decode("utf-8")] = ck#Decodificar y almacenar información del usuario

printStr = userName.decode("utf-8") + " conexión\n"#Muestre si la conexión se realizó correctamente en el cuadro de visualización de la conexión

text.insert(tkinter.END, printStr)

while True:

rData = ck.recv(1024)# Aceptar como unidad la información enviada por el cliente en 1k. La información recibida aquí es de tipo byte

dataStr = rData.decode("utf-8")

infolist = dataStr.split(":")#Dividir la cadena para obtener el nombre de usuario que se enviará y la información enviada por el cliente

d = int (infolist[1])

r = ""

ganancia = int(d\*0.02)

total = int(d+ganancia)

r = "- Dinero a invertir: $"+str(d)+"\n Ganansia (2%): $" + str(ganancia)+"\n Total: $"+str(total)+"\n"

tablero = "- Consulta realizada por "+str(infolist[0])+" - Valor $"+str(d)

users[infolist[0]].send((r).encode("utf")) #Envio de cadena al cliente extraído de la infolist y contenido en la lista de users

text.insert(tkinter.END, tablero+"\n")

# El cliente que quiere enviar información envía información al cliente objetivo

flag = False

except:

if flag is not True:

text.insert(tkinter.END, "- El servidor esta apagado\n")# Mostrado en la ventana de información

flag = True

def start():

server = socket.socket(socket.AF\_INET, socket.SOCK\_STREAM)#socked ipv4 o ipv6 y acuerdos relacionados

server.bind((ip, port))# ¡IP vinculante y número de puerto! ! ! 1: Tenga en cuenta que el número de puerto de entrada es de tipo str y aquí debe pasarse en tipo int

# 2: el parámetro de bind () tiene la forma de una tupla

server.listen(10)# Configure el monitoreo y establezca el número máximo de conexiones

text.insert(tkinter.END, "- El servidor se inició correctamente\n")# Mostrado en la ventana de información

while True:# El bucle infinito se usa aquí porque el servidor simulado debe estar ejecutándose todo el tiempo

ck, ca = server.accept()# Aceptar la información del cliente conectado

# Donde ca es una tupla compuesta por ip y número de puerto, ck información sobre el cliente

t = threading.Thread(target=run, args=(ck, ca))#Abre un hilo cada vez que un cliente está conectado

# Los parámetros de la función entrante en la función Thread también están en forma de tuplas

t.start()#Hilo abierto

def startSever():

s = threading.Thread(target=start)#Habilitar un hilo para iniciar el servidor

s.start()#Hilo abierto

#Inicio del servidor y de la interfaz grafica

startSever()

win.mainloop()

La parte indicada en verde hace referencia a la tarea que desempeña el servidor, en este caso, retornarle al usuario la ganancia que obtendrá con su inersión.

**Cliente**

import tkinter

from tkinter import messagebox

import socket

import threading

from random import randint

import os

win = tkinter.Tk()

idt = str(randint(0,50)) # Identificador de cada uno de los clientes

win.title("Cliente Banco N° "+idt) # titulo de la ventana con el Id del cliente

win.geometry("500x500") # dimensiones dela ventana

ip = "192.168.0.14" #IP del servidor

port = 5000 # Puerto del servidor

cliente\_id = "Cliente "+idt #Nombre del cliente

estadoVar = "" # Cadena con el estado del servidor

icon = "icons/bank.png" # ubicacíón del icono

flag = False #control de impresiónes en el campo de texto

fin = True # control del cierre del cliente

ck = None# Se utiliza para almacenar información del cliente

def getInfo():

#Se usan variables globales que permitan ser usados por otros metodos

global flag

global ck

global fin

while fin:

try:

data = ck.recv(1024)#Se utiliza para recibir la información enviada por el servicio

text.insert(tkinter.END, data.decode("utf-8"))#Display en el cuadro de mensaje

flag = False

except:

ck = None

if flag is not True:

estadoVar = "Sin conexion"

text.insert(tkinter.END,"- "+estadoVar+"\n")

flag = True

def connectServer():

global ck

try:

client = socket.socket(socket.AF\_INET, socket.SOCK\_STREAM)#socked ipv4 o ipv6 y acuerdos relacionados

client.connect((ip, port))# ¡Conecta ip y número de puerto! ! ! 1: Tenga en cuenta que el número de puerto de entrada es de tipo str y aquí debe pasarse en tipo int

# 2: el parámetro de bind () tiene la forma de una tupla

client.send(cliente\_id.encode("utf-8"))

ck = client

t = threading.Thread(target=getInfo)

estadoVar = "Conectado"

text.insert(tkinter.END,"- "+estadoVar+"\n")

t.start()

except:

estadoVar ="Sin conexion"

text.insert(tkinter.END,"- "+estadoVar+"\n")

def reconectar():

global flag

if flag is True:

connectServer()

def pedirPrestamo():

st = salario.get()

sendStr = cliente\_id+":"+st

ck.send(sendStr.encode("utf-8"))

salario.set("")

#Componentes de la interfaz

win.iconphoto(False, tkinter.PhotoImage(file=icon))

salario = tkinter.Variable()

labeltext= tkinter.Label(win, text="INVERSIONES", font=('Helvetica', 18, 'bold')).grid(padx=5,pady=5,row=0, column=0, columnspan=2)

labelUse = tkinter.Label(win, text="ID: ").grid(padx=5,pady=5,row=1, column=0)

entryUser = tkinter.Label(win, text=cliente\_id).grid(padx=5,pady=5,row=1, column=1,sticky="ew")

Salario = tkinter.Label(win, text="Valor a invertir: $").grid(padx=5,pady=5,row=2, column=0, sticky="ew")

labeltext= tkinter.Label(win, text="Respuesta",font=('Helvetica', 18, 'bold')).grid(padx=5,pady=5,row=4, column=0, columnspan=2)

button1 = tkinter.Button(win, text="Enviar", command=pedirPrestamo, bg="#62DF9B").grid(padx=5,pady=5,row=3, column=0,columnspan=2, sticky="ew")

entrySalario = tkinter.Entry(win, textvariable=salario).grid(padx=5,pady=5,row=2, column=1,sticky="ew")

text = tkinter.Text(win,pady=5, height=10, width=60)

text.grid(padx=5,pady=5,row=5, column=0,columnspan=2)

connectServer()

button2 = tkinter.Button(win, text="Reconectar", command=reconectar, bg="#DF6264").grid(padx=5,pady=5,row=7, column=0,columnspan=2, sticky="ew")

#Metodo para controlar el cierre del cliente y su hilo

def on\_closing():

global fin

if messagebox.askokcancel("Cerrar", "Quiere cerrar el cliente?"):

win.destroy()

ck.close()

fin = False

os.\_exit(0)

#Aplicación del protocolo

win.protocol("WM\_DELETE\_WINDOW", on\_closing)

win.mainloop()

Para la creación de varios clientes, solo se requiere ejecutar el mismo script y por medio de los IDS con valores aleatorios se puede garantizar que los clientes que se creen puedan recibir las respuestas a sus peticiones óptimamente.

A continuación, se muestra el funcionamiento del servidor trabajando con dos clientes, teniendo en el servidor un campo en donde se registran los clientes conectados y las peticiones realizadas por estos.

**FUNCIONAMIENTO DE SERVIDOR Y DOS CLIENTES – EJERCICIO BANCO**

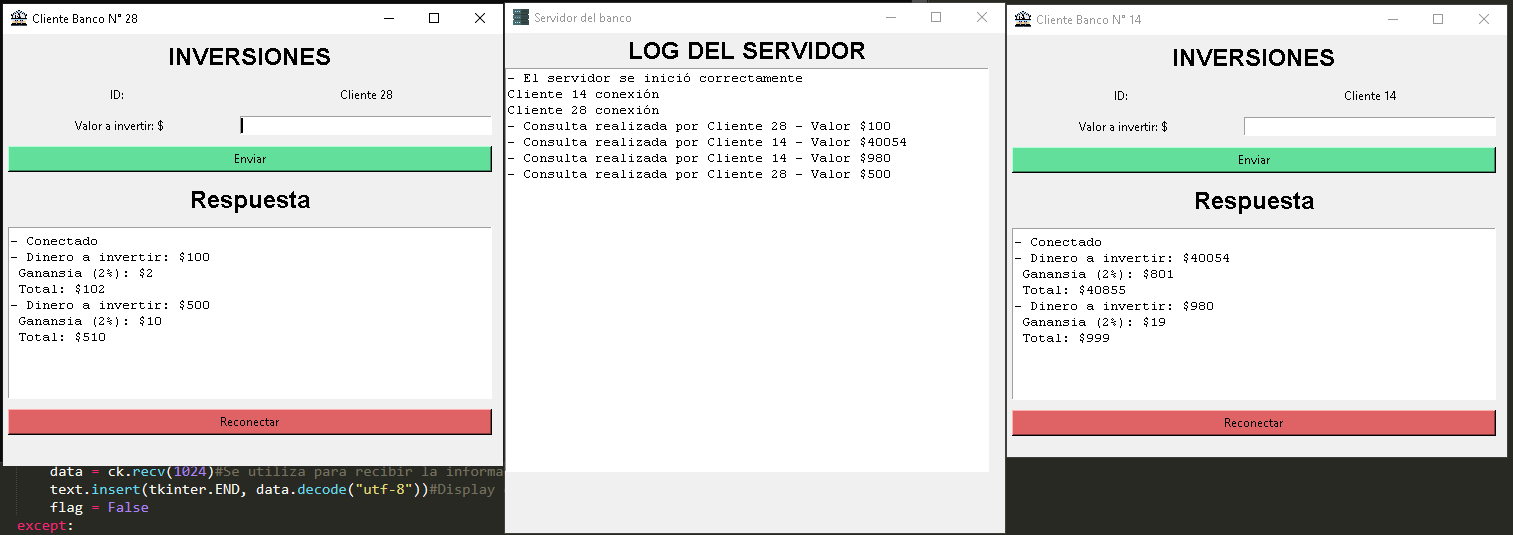


Ilustración 1 En el centro tenemos al servidor el cual muestra un Log con las peticiones y conexiones de los clientes

1. Para el ejercicio del supermercado definimos los siguientes scripts desarrollados en Python y el uso de sus librerías de Sockets e Hilos.

**Servidor**

import tkinter

import socket, threading

win = tkinter.Tk() # Crea la ventana principal

win.title('Servidor del supermercado')

win.geometry("500x500")

users = {}#Diccionario de usuario, también puede conectarse a la base de datos

ip = "192.168.0.14"

tablero = ""

icon = "icons/server.png" # ubicacíón del icono

flag = False

port = 5000

text = tkinter.Text(win, height=25, width=60)

labeltext = tkinter.Label(win, text='LOG DEL SERVIDOR',font=('Helvetica', 18, 'bold')).grid(row=0, column=0)

text.grid(row=1, column=0)

win.iconphoto(False, tkinter.PhotoImage(file=icon))

def run(ck, ca):

global flag

try:

userName = ck.recv(1024)# Aceptar como unidad la información enviada por el cliente en 1k. La información recibida aquí es de tipo byte

users[userName.decode("utf-8")] = ck#Decodificar y almacenar información del usuariofont=('Helvetica', 18, 'bold')

printStr = userName.decode("utf-8") + " conexión\n"#Muestre si la conexión se realizó correctamente en el cuadro de visualización de la conexión

text.insert(tkinter.END, printStr)

while True:

rData = ck.recv(1024)# Aceptar como unidad la información enviada por el cliente en 1k. La información recibida aquí es de tipo byte

dataStr = rData.decode("utf-8")

infolist = dataStr.split(":")#Dividir la cadena para obtener el nombre de usuario que se enviará y la información enviada por el cliente

d = int (infolist[1])

r = ""

ganancia = int(d\*0.15)

total = int(d-ganancia)

r = "- Total sin descuento: $"+str(d)+"\n Descuento (15%): $" + str(ganancia)+"\n TOTAL A PAGAR: $"+str(total)+"\n"

tablero = "- Consulta realizada por "+str(infolist[0])+" - Valor compra $"+str(d)

users[infolist[0]].send((r).encode("utf"))

text.insert(tkinter.END, tablero+"\n")

# El cliente que quiere enviar información envía información al cliente objetivo

flag = False

except:

if flag is not True:

text.insert(tkinter.END, "- El servidor esta apagado\n")# Mostrado en la ventana de información

flag = True

def start():

server = socket.socket(socket.AF\_INET, socket.SOCK\_STREAM)#socked ipv4 o ipv6 y acuerdos relacionados

server.bind((ip, port))# ¡IP vinculante y número de puerto! ! ! 1: Tenga en cuenta que el número de puerto de entrada es de tipo str y aquí debe pasarse en tipo int

# 2: el parámetro de bind () tiene la forma de una tupla

server.listen(10)# Configure el monitoreo y establezca el número máximo de conexiones

text.insert(tkinter.END, "- El servidor se inició correctamente\n")# Mostrado en la ventana de información

while True:# El bucle infinito se usa aquí porque el servidor simulado debe estar ejecutándose todo el tiempo

ck, ca = server.accept()# Aceptar la información del cliente conectado

# Donde ca es una tupla compuesta por ip y número de puerto, ck información sobre el cliente

t = threading.Thread(target=run, args=(ck, ca))#Abre un hilo cada vez que un cliente está conectado

# Los parámetros de la función entrante en la función Thread también están en forma de tuplas

t.start()#Hilo abierto

def startSever():

s = threading.Thread(target=start)#Habilitar un hilo para iniciar el servidor

s.start()#Hilo abierto

# Lo siguiente trata sobre el funcionamiento de la interfaz

startSever()

win.mainloop()

La parte en verde define la actividad del servidor al realizar el calculo del ahorro del cliente al realizar una compra con cierto valor.

**Cliente**

import tkinter

import socket

import threading

from random import randint

win = tkinter.Tk()

idt = str(randint(0,50))

win.title("Cliente Supermercado N° "+idt)

win.geometry("500x500")

ip = "192.168.0.14"

port = 5000

cliente\_id = "Cliente "+idt

icon = "icons/market.png" # ubicacíón del icono

estadoVar = ""

flag = False

ck = None# Se utiliza para almacenar información del cliente

def getInfo():

global flag

global ck

while True:

try:

data = ck.recv(1024)#Se utiliza para recibir la información enviada por el servicio

text.insert(tkinter.END, data.decode("utf-8"))#Display en el cuadro de mensaje

flag = False

except:

ck = None

if flag is not True:

estadoVar = "Sin conexion"

text.insert(tkinter.END,"- "+estadoVar+"\n")

flag = True

def connectServer():

global ck

try:

client = socket.socket(socket.AF\_INET, socket.SOCK\_STREAM)#socked ipv4 o ipv6 y acuerdos relacionados

client.connect((ip, port))# ¡Conecta ip y número de puerto! ! ! 1: Tenga en cuenta que el número de puerto de entrada es de tipo str y aquí debe pasarse en tipo int

# 2: el parámetro de bind () tiene la forma de una tupla

client.send(cliente\_id.encode("utf-8"))

ck = client

t = threading.Thread(target=getInfo)

estadoVar = "Conectado"

text.insert(tkinter.END,"- "+estadoVar+"\n")

t.start()

except:

estadoVar ="Sin conexion"

text.insert(tkinter.END,"- "+estadoVar+"\n")

def reconectar():

global flag

if flag is True:

connectServer()

def pedirPrestamo():

st = salario.get()

sendStr = cliente\_id+":"+st

ck.send(sendStr.encode("utf-8"))

salario.set("")

#Abajo está la interfaz

win.iconphoto(False, tkinter.PhotoImage(file=icon))

salario = tkinter.Variable()

labeltext= tkinter.Label(win, text="DESCUENTO 15%", font=('Helvetica', 18, 'bold')).grid(padx=5,pady=5,row=0, column=0, columnspan=2)

labelUse = tkinter.Label(win, text="ID: ").grid(padx=5,pady=5,row=1, column=0)

entryUser = tkinter.Label(win, text=cliente\_id).grid(padx=5,pady=5,row=1, column=1,sticky="ew")

Salario = tkinter.Label(win, text="Valor compra: $").grid(padx=5,pady=5,row=2, column=0, sticky="ew")

labeltext= tkinter.Label(win, text="Respuesta",font=('Helvetica', 18, 'bold')).grid(padx=5,pady=5,row=4, column=0, columnspan=2)

button1 = tkinter.Button(win, text="Enviar", command=pedirPrestamo, bg="#62DF9B").grid(padx=5,pady=5,row=3, column=0,columnspan=2, sticky="ew")

entrySalario = tkinter.Entry(win, textvariable=salario).grid(padx=5,pady=5,row=2, column=1,sticky="ew")

text = tkinter.Text(win,pady=5, height=10, width=60)

text.grid(padx=5,pady=5,row=5, column=0,columnspan=2)

connectServer()

button2 = tkinter.Button(win, text="Reconectar", command=reconectar, bg="#DF6264").grid(padx=5,pady=5,row=7, column=0,columnspan=2, sticky="ew")

win.mainloop()

A continuación, se verá el funcionamiento del ejercicio con teniendo al servidor activo y dos clientes realizando peticiones a este.

**FUNCIONAMIENTO DE SERVIDOR Y DOS CLIENTES – EJERCICIO SUPERMERCADO**

Interfaz de usuario gráfica, Texto, Aplicación, Word

Descripción generada automáticamente

Ilustración 2 el servidor se encuentra en la mitad, mostrando su funcionamiento mediante el almacenamiento de las peticiones realizadas por los client

1. En el desarrollo del ejercicio del préstamo realizado por la empresa se hizo uso de Python como lenguaje de programación y sus librerías de Sockets, hilos y Tkinter.

**Servidor**

import tkinter

import socket, threading

win = tkinter.Tk() # Crea la ventana principal

win.title('Servidor Prestamos')

win.geometry("500x500")

users = {}#Diccionario de usuario, también puede conectarse a la base de datos

ip = "192.168.0.14"

tablero = ""

icon = 'icons/server.png'

flag = False

port = 5000

text = tkinter.Text(win, height=25, width=60)

labeltext = tkinter.Label(win, text='LOG DEL SERVIDOR',font=('Helvetica', 18, 'bold')).grid(row=0, column=0)

text.grid(row=1, column=0)

win.iconphoto(False, tkinter.PhotoImage(file=icon))

def run(ck, ca):

global flag

try:

userName = ck.recv(1024)# Aceptar como unidad la información enviada por el cliente en 1k. La información recibida aquí es de tipo byte

users[userName.decode("utf-8")] = ck#Decodificar y almacenar información del usuario

printStr = "" + userName.decode("utf-8") + " conexión\n"#Muestre si la conexión se realizó correctamente en el cuadro de visualización de la conexión

text.insert(tkinter.END, printStr)

while True:

rData = ck.recv(1024)# Aceptar como unidad la información enviada por el cliente en 1k. La información recibida aquí es de tipo byte

dataStr = rData.decode("utf-8")

infolist = dataStr.split(":")#Dividir la cadena para obtener el nombre de usuario que se enviará y la información enviada por el cliente

s = int (infolist[1])

r = ""

if s > 3500000:

cuota = (1000000/6)\*1.2

r = "SALARIO= $"+str(s)+"\nCuota mensual (con 1.2% de interes): $" + str(int(cuota)) + "\nValor prestado: $1000000\nValor total a pagar: $" + str(int(cuota\*6))+"\n"

tablero = "Prestamo pedido por "+str(infolist[0])+" - salario $"+str(s)+" - Aceptado"

else:

r = "Usted gana un salario inferior o igual a $3500000\nPor lo que no se le puede realizar el prestamo\n"

tablero = "Prestamo pedido por "+str(infolist[0])+" - salario $"+str(s)+" - Negado"

users[infolist[0]].send((r).encode("utf"))

text.insert(tkinter.END, "\n"+tablero+"\n")

# El cliente que quiere enviar información envía información al cliente objetivo

flag = False

except:

if flag is not True:

text.insert(tkinter.END, "- El servidor esta apagado\n")# Mostrado en la ventana de información

flag = True

def start():

server = socket.socket(socket.AF\_INET, socket.SOCK\_STREAM)#socked ipv4 o ipv6 y acuerdos relacionados

server.bind((ip, port))# ¡IP vinculante y número de puerto! ! ! 1: Tenga en cuenta que el número de puerto de entrada es de tipo str y aquí debe pasarse en tipo int

# 2: el parámetro de bind () tiene la forma de una tupla

server.listen(10)# Configure el monitoreo y establezca el número máximo de conexiones

text.insert(tkinter.END, "- El servidor se inició correctamente\n")# Mostrado en la ventana de información

while True:# El bucle infinito se usa aquí porque el servidor simulado debe estar ejecutándose todo el tiempo

ck, ca = server.accept()# Aceptar la información del cliente conectado

# Donde ca es una tupla compuesta por ip y número de puerto, ck información sobre el cliente

t = threading.Thread(target=run, args=(ck, ca))#Abre un hilo cada vez que un cliente está conectado

# Los parámetros de la función entrante en la función Thread también están en forma de tuplas

t.start()#Hilo abierto

def startSever():

s = threading.Thread(target=start)#Habilitar un hilo para iniciar el servidor

s.start()#Hilo abierto

# Lo siguiente trata sobre el funcionamiento de la interfaz

startSever()

win.mainloop()

La parte indicada en verde hace referencia a la tarea que desempeña el servidor, en este caso, el determinar si un empleado puede o no acceder a un préstamo.

**Cliente**

import tkinter

import socket

import threading

from random import randint

win = tkinter.Tk()

idt = str(randint(0,50))

win.title("Cliente Prestamos N°"+idt)

win.geometry("500x500")

ip = "192.168.0.14"

port = 5000

cliente\_id = "Empleado"+idt

icon = "icons/enterprise.png" # ubicacíón del icono

estadoVar = ""

flag = False

ck = None# Se utiliza para almacenar información del cliente

def getInfo():

global flag

global ck

while True:

try:

data = ck.recv(1024)#Se utiliza para recibir la información enviada por el servicio

text.insert(tkinter.END, "\n"+data.decode("utf-8"))#Display en el cuadro de mensaje

flag = False

except:

ck = None

if flag is not True:

estadoVar = "Sin conexion"

text.insert(tkinter.END,"- "+estadoVar+"\n")

flag = True

def connectServer():

global ck

try:

client = socket.socket(socket.AF\_INET, socket.SOCK\_STREAM)#socked ipv4 o ipv6 y acuerdos relacionados

client.connect((ip, port))# ¡Conecta ip y número de puerto! ! ! 1: Tenga en cuenta que el número de puerto de entrada es de tipo str y aquí debe pasarse en tipo int

# 2: el parámetro de bind () tiene la forma de una tupla

client.send(cliente\_id.encode("utf-8"))

ck = client

t = threading.Thread(target=getInfo)

estadoVar = "Conectado"

text.insert(tkinter.END,"- "+estadoVar+"\n")

t.start()

except:

estadoVar ="Sin conexion"

text.insert(tkinter.END,"- "+estadoVar+"\n")

def reconectar():

global flag

if flag is True:

connectServer()

def pedirPrestamo():

st = salario.get()

sendStr = cliente\_id+":"+st

ck.send(sendStr.encode("utf-8"))

salario.set("")

#Abajo está la interfaz

win.iconphoto(False, tkinter.PhotoImage(file=icon))

salario = tkinter.Variable()

labeltext= tkinter.Label(win, text="PRESTAMO", font=('Helvetica', 18, 'bold')).grid(padx=5,pady=5,row=0, column=0, columnspan=2)

labelUse = tkinter.Label(win, text="ID: ").grid(padx=5,pady=5,row=1, column=0)

entryUser = tkinter.Label(win, text=cliente\_id).grid(padx=5,pady=5,row=1, column=1,sticky="ew")

Salario = tkinter.Label(win, text="Salario: ").grid(padx=5,pady=5,row=2, column=0, sticky="ew")

labeltext= tkinter.Label(win, text="Respuesta",font=('Helvetica', 18, 'bold')).grid(padx=5,pady=5,row=4, column=0, columnspan=2)

button1 = tkinter.Button(win, text="Enviar", command=pedirPrestamo, bg="#62DF9B").grid(padx=5,pady=5,row=3, column=0,columnspan=2, sticky="ew")

entrySalario = tkinter.Entry(win, textvariable=salario).grid(padx=5,pady=5,row=2, column=1,sticky="ew")

text = tkinter.Text(win,pady=5, height=10, width=60)

text.grid(padx=5,pady=5,row=5, column=0,columnspan=2)

connectServer()

button2 = tkinter.Button(win, text="Reconectar", command=reconectar, bg="#DF6264").grid(padx=5,pady=5,row=7, column=0,columnspan=2, sticky="ew")

win.mainloop()

A continuación, se muestra cómo funciona cada uno de los scripts expuestos, ejecutando una instancia del servidor y dos clientes.

**FUNCIONAMIENTO DE SERVIDOR Y DOS CLIENTES – EJERCICIO PRESTAMO**

Interfaz de usuario gráfica, Texto, Aplicación

Descripción generada automáticamente

Ilustración 3 En el centro se encuentra la ventana del servidor el cual almacena las conexiones de los clientes y las peticiones que estos realicen