Second Network Programming Homework

إعداد الطلاب: خضر منير عودة 2510 رشا علان علي 2729

2024

إشراف الدكتور المهندس: مهند عيسى
السنة الخامسة هندسة الاتصالات والالكترونيات

Question 1: Bank ATM Application with TCP Server/Client and Multi-threading:

برنامج السيرفر:

```
import socket, threading

HOST = '0.0.0.0'
PORT = 3434

accounts = {
    "KHDER": 1050,
    "RASHA": 5050
}

def handle_client(conn, addr):
    print("Connected by {}".format(addr))
    while True:
        data = conn.recv(1024).decode()
        if not data:
            break

        account_number, operation, amount = data.split()
        if account_number not in accounts:
            conn.sendall("Invalid account Name".encode())
            continue

        try:
            amount = float(amount)
        except ValueError:
            conn.sendall("Invalid amount".encode())
            continue

        if operation == "check_balance":
            balance = accounts[account_number]
            conn.sendall("Your balance is: {}".format(balance).encode())
        elif operation == "deposit":
            accounts[account_number] += amount
            conn.sendall("Deposit successful. New balance:
{}".format(accounts[account_number]).encode())
        elif operation == "withdraw":
            if accounts[account_number] < amount:
                conn.sendall("Insufficient funds".encode())
            else:
                accounts[account_number] -= amount
```

```
                conn.sendall("Withdrawal successful. New balance:
{}".format(accounts[account_number]).encode())
        else:
            conn.sendall("Invalid operation".encode())

    conn.close()
    print("Client {} disconnected".format(addr))

with socket.socket(socket.AF_INET, socket.SOCK_STREAM) as s:
    s.bind((HOST, PORT))
    s.listen()
    print("Server listening on {}:{}".format(HOST, PORT))
    while True:
        conn, addr = s.accept()
        thread = threading.Thread(target=handle_client, args=(conn, addr))
        thread.start()
```

تشغيل البرنامج:

```
Server listening on 0.0.0.0:3434
```

برنامج الزبون الأول:

```
import socket

HOST = '127.0.0.1'
PORT = 3434

while True:
    account_number = input("Enter your account Name: ")
    operation = input("Enter operation (check_balance, deposit, withdraw):
")
    if operation in ("deposit", "withdraw"):
        amount = float(input("Enter amount: "))
    data = f"{account_number} {operation} {amount}".encode()

    with socket.socket(socket.AF_INET, socket.SOCK_STREAM) as s:
        s.connect((HOST, PORT))
        s.sendall(data)
        response = s.recv(1024).decode()
        print(response)
```

حتى نجعل السيرفر يخدم عدد كبير من المستخدمين بنفس الوقت يجب الاستفادة من المودل threading، تم تعيين IP السيرفر على 0.0.0.0 من أجل نخديم أي عنوان بالشبكة، ورقم المنفذ على 3434. خزنت الحسابات في dictionary له الاسم accounts، اعتمدت على رقم الحساب في التخزين بحيث جعلت رقم الحساب هو المفتاح والمبلغ المالي هو القيمة المقابلة.

بتعرف التابع handle_client(conn, addr) أتعامل مع اتصالات العملاء بحيث مررت له سوكيت العميل وهو البارمتر conn وعنوان العميل addr. استقبل معلومات العميل باستخدام:

```
data = conn.recv(1024).decode()
```

وثم عن طريق تعريف المتحولات رقم الحساب ونوع العملية المرادة وإجمالي القيمة المضافة أو المسحوبة أستطيع فصل هذه البيانات باستخدام data.split().

تشغيل برنامج الزبون:

```
Enter your account Name: KHDER
Enter operation (check_balance, deposit, withdraw): deposit
Enter amount: 5000
Deposit successful. New balance: 6050.0
Enter your account Name: RASHA
Enter operation (check_balance, deposit, withdraw): check_balance
Your balance is: 5050
Enter your account Name:
```

Question 2: TO DO List App:

```
import tkinter as tk

class TodoListApp:
    def __init__(self, root):
        self.root = root
        self.tasks = []

        title_label = tk.Label(root, text='TODO List App',
font=('Helvetica', 24))
        title_label.grid(row=0, column=0, columnspan=2, padx=10, pady=10)

        self.task_entry = tk.Entry(root)
        self.task_entry.grid(row=1, column=0, padx=10, pady=5)

        add_button = tk.Button(root, text='Add Task',
command=self.add_task)
        add_button.grid(row=1, column=1, padx=10, pady=5)

        tasks_label = tk.Label(root, text='Tasks:', font=('Helvetica',
18))
        tasks_label.grid(row=2, column=0, columnspan=2, padx=10, pady=5)

        self.task_listbox = tk.Listbox(root, selectmode=tk.SINGLE)
        self.task_listbox.grid(row=3, column=0, columnspan=2, padx=10,
pady=5)

        mark_button = tk.Button(root, text='Mark Task as Completed',
command=self.mark_task_completed)
        mark_button.grid(row=4, column=0, padx=10, pady=5)

        remove_button = tk.Button(root, text='Remove Task',
command=self.remove_task)
        remove_button.grid(row=4, column=1, padx=10, pady=5)

    def add_task(self):
        task = self.task_entry.get()
        if task:
            self.tasks.append(task)
            self.task_listbox.insert(tk.END, task)
            self.task_entry.delete(0, tk.END)

    def mark_task_completed(self):
        selected_index = self.task_listbox.curselection()
```

```
        if selected_index:
            task_index = selected_index[0]
            task = self.task_listbox.get(task_index)
            self.tasks[task_index] = f'[COMPLETED] {task}'
            self.task_listbox.delete(task_index)
            self.task_listbox.insert(task_index, self.tasks[task_index])

    def remove_task(self):
        selected_index = self.task_listbox.curselection()
        if selected_index:
            task_index = selected_index[0]
            self.tasks.pop(task_index)
            self.task_listbox.delete(task_index)

if __name__ == '__main__':
    root = tk.Tk()
    root.title('TODO List')
    app = TodoListApp(root)
    root.mainloop()
```

شرح الكود:

import tkinter as tk: يتم استيراد المودل tkinter ويتم تعيين اسمها المستعار tk. هذا يتيح استخدام tk بدلاً من الاسم الكامل للوحدة في جميع أسطر الكود.

هذا السطر يعرّف فئة (Class) تُسمى TodoListApp الفئات هي بمثابة قوالب لإنشاء كائنات (Objects) تحتفظ بالبيانات (صفات) و(طرق). في هذه الحالة، يمثل كائن TodoListApp تطبيق قائمة المهام بأكمله.

هذه هي طريقة __init__، والمعروفة أيضًا بالمنشئ (Constructor)، للصنف TodoListApp يتم استدعاؤها تلقائيًا عند إنشاء مثيل جديد للفئة.

يشير البارمتر self إلى الكائن الحالي نفسه.

يخزن self.root = root نافذة التطبيق الرئيسية (تم إنشاؤها باستخدام tk.Tk()) في سمة root للكائن.

self.tasks = [] يبادر بإنشاء قائمة فارغة تسمى tasksلتخزين عناصر قائمة المهام.

عناصر واجهة المستخدم (UI):

- title_label: تسمية تعرض "تطبيق قائمة المهام" مع تنسيق.
- task_entry: حقل إدخال للمستخدمين لكتابة مهام جديدة.
- add_button: زر يؤدي إلى تشغيل طريقة add_taskعند النقر فوقه.
- tasks_label: تسمية تعرض "المهام.":
- task_listbox: قائمة عرض لإظهار عناصر قائمة المهام.
- mark_button: زر يؤدي إلى تشغيل طريقة mark_task_completed عند النقر فوقه.
- remove_button: زر يؤدي إلى تشغيل طريقة remove_task عند النقر فوقه.