```
# Lab 4 - web api, multithreading, multiprocessing, os, review of json, gui
# Names: Trien Bang Huynh and Marcel Gunadi
# lab4process.py - Multiprocessing

import urllib.request, json
import os
import tkinter as tk
import tkinter.messagebox as tkmb
from collections import defaultdict
import time
import tkinter.filedialog
import multiprocessing as mp
from dotenv import load_dotenv



# You can ignore and comment out two lines of below code if you don't store your
API in .env file.
# ---------------------------------
load_dotenv() #                      |
myAPIkey = os.getenv("API_KEY")#    |
# ---------------------------------


# API key is a secret but you can hardcode your API key here to fetch data
# HEADERS = {"X-Api-Key": "INSERT-YOUR-API-KEY-HERE"}
HEADERS = {"X-Api-Key":f"{myAPIkey}"}

class MainWin(tk.Tk):
    """
    A main tk window for the application, allowing users to select states and fetch
national park data.
    """
    def __init__(self):
        super().__init__()
        self.title("US NPS")
        self._stateSelection = []
        self._dataAPI = []

        with open("states_hash.json", 'r') as fh:
            self._states_dic = json.load(fh)

        tk.Label(self, text="National Park Finder", fg="green",font=('Times',
17)).grid(row=0, column=0, columnspan=3, pady=10, padx=10)

        self.L1 = tk.Label(self, text="Select up to 5 states",
fg="black",font=('Times', 15))
        self.L1.grid(row=1, column=0, columnspan=3, pady=10)

        # Listbox and Scrollbar
        self.listbox = tk.Listbox(self, width = 50, height=10,
selectmode="multiple")
        self.scrollbar = tk.Scrollbar(self, orient=tk.VERTICAL,
command=self.listbox.yview)
        self.listbox.configure(yscrollcommand=self.scrollbar.set)


        # populate the listbox with US's states
        for state in self._states_dic.values() :
```

```
            self.listbox.insert(tk.END, state)


        self.listbox.grid(row=2, column=0,ipadx=5, padx=20, pady=20, sticky="nsew")
        self.scrollbar.grid(row=2, column=1, sticky="ns")


         # Select button
        self.B = tk.Button(self, text="Submit choice", font=('Times', 15), command=
self.onClicked)
        self.B.grid(row=3, column=0, columnspan=2, padx=20, pady=20)

        # status label
        self.L2 = tk.Label(self, text="",font=('Times', 15))
        self.L2 .grid(row=4, column=0, columnspan=3, pady=10, padx=10)


    def onClicked(self):
        """
        A callback function for submit button to check if the selection is valid,
        and proceed to fetch park data for the selected states.
        """
        # Clear the previous state selection
        self._stateSelection = []

        # Check the number of selected states
        if len(self.listbox.curselection()) == 0  or
len(self.listbox.curselection()) > 5:
            tkmb.showerror("Error", "Please select between 1 and 5 states.")
            self.listbox.selection_clear(0, tk.END)
            return
        else:
            for index in self.listbox.curselection():
                codeState, nameState = list(self._states_dic.items())[index]
                self._stateSelection.append((codeState,nameState))
            self.parksFinder()


    def parksFinder(self):
        """
        A method which fetches park data for selected states using multiple
threads,
        show the fetched data in the listbox and the fetching status in a label.
        """
        self.L1['text'] = "Select parks to save parks info to file"
        self.B['text'], self.B['command'] = "Save", self.saveFile
        self.listbox.delete(0, tk.END)

        args = []
        for choice in self._stateSelection:
            # an ex of choice = "AL": "Alabama"
            stateCode, stateName = choice

            # Configure API request
            endpoint = f'https://developer.nps.gov/api/v1/parks?
stateCode={stateCode}'

            args.append((endpoint, stateName))
```

```
        # start recording fetching time
        start =  time.time()

        self.pool = mp.Pool(processes= len(self._stateSelection))
        self._dataAPI = self.pool.map(requestAPI, args)


        print(f"Time of fetching data from API for all processes: {time.time() -
start:.2f}s")


        for dataObj in self._dataAPI:
           for stateName, dicOfParksInfo in dataObj.items():
                for park in dicOfParksInfo['data']:
                    tempStr = stateName + ": " + park['name']
                    self.listbox.insert(tk.END, tempStr)

        self.L2['text'] = f"Fetching data for {len(self._stateSelection)} state(s)"

    def saveFile(self):
        """
        A method which saves selected park information to a JSON file in the chosen
directory.
        """

        if len(self.listbox.curselection()) == 0:
            tkmb.showerror("Error", "Please select at least 1 park")
            return

        # each key (statename) will have list of selected parks
        selectedParksWithStates = defaultdict(list)

        for i in self.listbox.curselection():
            stateAndPark = self.listbox.get(i)
            stateName = stateAndPark.split(":")[0].strip()
            parkName = stateAndPark.split(":")[1].strip()
            selectedParksWithStates[stateName].append(parkName)

        # open current directory
        directory = tk.filedialog.askdirectory(initialdir='.')

        filesSave = []
        # if user choose a directory
        if directory:
            for stateName, listOfParks in selectedParksWithStates.items():
                listToSave = []
                for park in listOfParks:
                    parkDictForFile = {}
                    tempDict = {}
                    # loop through each parks in selected states to find the parks
chosen from the listbox
                    for dataObj in self._dataAPI:
                        for state, dicOfParksInfo in dataObj.items():
                            if stateName != state:
                                continue
                            for parkDict in dicOfParksInfo['data']:
                                if parkDict['name'] == park:
                                    tempDict['full name'] = parkDict['fullName']
                                    tempDict['description'] =
```

```
parkDict['description']
                                    tempAcList = []
                                    for act in parkDict['activities']:
                                        tempAcList.append(act['name'])
                                    tempDict['activities'] = ", ".join(tempAcList)
                                    tempDict['url']  = parkDict['url']
                                    break


                        parkDictForFile[park] = tempDict
                        listToSave.append(parkDictForFile)

                    # saving file in selected directory
                    filename = stateName + ".json"
                    pathToSaveFile = directory + "/" + filename
                    filesSave.append(filename)


                    with open(pathToSaveFile, 'w') as fh:
                        json.dump(listToSave, fh, indent=3)


                # create a confirm messagebox
                filesSaveStr = ", ".join(filesSave)

                confirmed = tkmb.askyesno("Confirmation", f"Save files:
{filesSaveStr}")
                if confirmed:
                    self.closeWin()

            else:
                self.listbox.selection_clear(0, tk.END)


        def closeWin(self):
            """
            A method which will close the application window.
            """
            self.destroy()
            self.quit()

    def requestAPI(args):
        '''
        A global function which fetches data from the given API endpoint and is
    accessed by multi-processes.
        '''
        endpoint, stateName  = args[0], args[1]

        # Make API request and get response
        req = urllib.request.Request(endpoint,headers=HEADERS)
        response = urllib.request.urlopen(req)

        # Parse JSON data from response
        data = json.loads(response.read().decode())

        return {stateName:data}


    def main():
```

```
    app = MainWin()
    app.mainloop()

if __name__ == '__main__' :
    main()



'''
Step 5: Analysis
Execution time order from slowest to fastest: Serial -> Multiprocessing ->
Multithreading

The serial approach is slowest because it executes tasks in a sequential manner.
Each task has to complete before the next one begins,
which results in idle time if the task involves waiting for a network response.

Multithreading is faster than serial because it allows for concurrent execution.
When one thread is blocked waiting for a network response, other threads can
continue execution.
This makes the most of the time spent waiting for I/O and is why multithreading is
usually the
fastest approach for I/O-bound tasks, even considering Python's Global Interpreter
Lock (GIL).

Multiprocessing is faster than serial but slower than multithreading in this
context.
Multiprocessing involves multiple Python processes each with their own interpreter
and memory space, which means there's a larger overhead compared to threading.
Additionally, multiprocessing can outperform threading for CPU-bound tasks, but for
I/O-bound tasks like in this lab (fetching data from APIs), the added overhead and
lack of shared memory make it slower than multithreading.
'''
```