|  |  |
| --- | --- |
|  | **DEPARTMENT OF COMPUTER ENGINEERING** |

**Mini Project Report**

|  |  |
| --- | --- |
| Semester | S.E. Semester IV – Computer Engineering |
| Subject | Operating System |
| Subject Professor In-charge | Prof. Pankaj Vanwari |
| Assisting Teachers | Prof. Pankaj Vanwari |

|  |  |
| --- | --- |
| Roll Numbers | Name of Students |
| 21102A0014 | Deep Salunkhe |
| 21102A0003 | Omkar Patil |
| 21102A0005 | Pranav Redij |

**Name of the Project: Banker’s Algorithm GUI**

**Description:**

We have used the Tkinter library to create a graphical user interface for the Banker's Algorithm, which is a resource allocation and deadlock avoidance algorithm commonly used in operating systems. The user is prompted to enter the number of processes and resources, and then the maximum available resources, allocated resources, and maximum resource needs for each process are inputted through Entry widgets.

After the user inputs all the required information, the Compute function is executed, which calculates whether the system is in a safe state or not. It checks if the allocated resources for each process are less than or equal to the maximum resource needs for that process and also whether the total allocated resources are less than or equal to the maximum available resources. If these conditions are met, the system is in a safe state and can continue to allocate resources to the processes. Otherwise, the system is in an unsafe state and a message is displayed to the user.

The output of the Compute function is displayed in a Label widget on the GUI, which shows the total allocated and available resources and whether the system is in a safe or unsafe state

**Project Code:**

from tkinter import \*

*# Create the main window*

window = Tk()

window.title(" Banker's Algorithm ")

*# Set the window size to the maximum screen size*

*# width = window.winfo\_screenwidth()*

*# height = window.winfo\_screenheight()*

window.geometry(f"500x500")

window.configure(bg='lightblue')

Label( text='Enter the number of Processes').pack()

prc = Entry()

prc.pack()

Label( text='Enter the number of Resources').pack()

rcs = Entry()

rcs.pack()

def Imput():

    NoOfProcesses = int(prc.get())

    NoOfResources = int(rcs.get())

    Label( text=f'Max Available Resources for {NoOfResources} processes', bg='lightblue', fg='#f00').pack()

    ar = Entry()

    ar.pack()

    AlWd = []

    for \_ in range(NoOfProcesses):

        Label( text=f'Allocated Resource for Process {\_+1}', bg='lightblue', fg='#f00').pack()

        temp = Entry()

        temp.pack()

        AlWd.append(temp)

    MaxWd = []

    for \_ in range(NoOfProcesses):

        Label( text=f'Max Resource for Process {\_+1}', bg='lightblue', fg='#f00').pack()

        temp = Entry()

        temp.pack()

        MaxWd.append(temp)

    def Compute():

        processes = int(prc.get())

        resources = int(rcs.get())

        max\_resources = [int(x) for x in ar.get().split(' ')]

        currently\_allocated = [[int(x) for x in \_.get().split(' ')] for \_ in AlWd]

        max\_need = [[int(x) for x in \_.get().split(' ')] for \_ in MaxWd]

        allocated = [0] \* resources

        for i in range(processes):

            for j in range(resources):

                allocated[j] += currently\_allocated[i][j]

        output = f"\nTotal allocated resources : {allocated}"

        available = [max\_resources[i] - allocated[i] for i in range(resources)]

*# print(f"total available resources : {available}\n")*

        output += f"\nTotal available resources : {available}\n"

        output += "\nMax needs of resources: \n"

        for o in max\_need:

            output += str(o)

            output += '\n'

        running = [True] \* processes

        count = processes

        while count != 0:

            safe = False

            for i in range(processes):

                if running[i]:

                    executing = True

                    for j in range(resources):

                        if max\_need[i][j] - currently\_allocated[i][j] > available[j]:

                            executing = False

                            break

                    if executing:

                        output += f"\nprocess {i + 1} is executing"

*# print(f"process {i + 1} is executing")*

                        running[i] = False

                        count -= 1

                        safe = True

                        for j in range(resources):

                            available[j] += currently\_allocated[i][j]

                        break

            if not safe:

                output += "\nProcesses are in an unsafe state."

*# print("the processes are in an unsafe state.")*

                break

            output += f"\nProcess is in a safe state.\nAvailable resources : {available}\n"

        X = Label( text=output)

        X.pack()

    Compute = Button(text='Calculate', command = Compute)

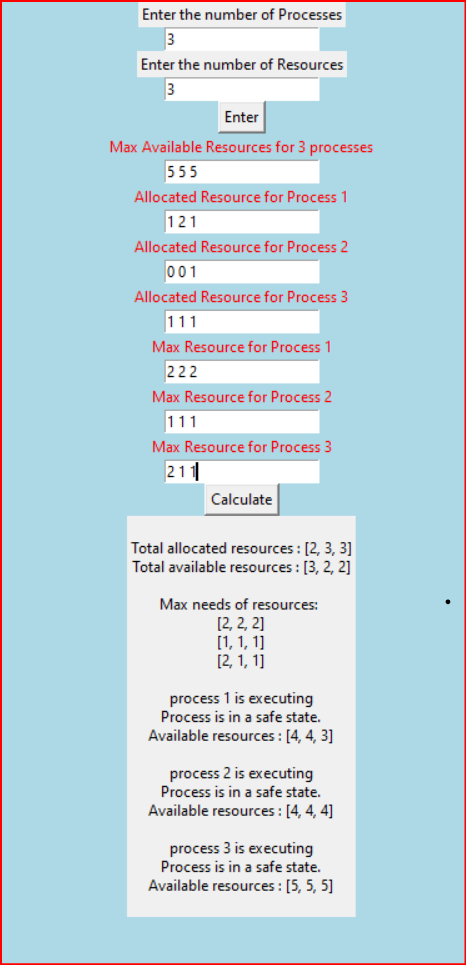
    Compute.pack()

Imput = Button( text='Enter', command = Imput)

Imput.pack()

*# Run the event loop*

window.mainloop()

**Result/ Output:**