Banker Algorithm with GUI

Submitted in partial fulfillment of the requirements of

**Mini Project (Operating System)**

for

Second Year of Computer Engineering

By

Omkar Patil 21102A0003

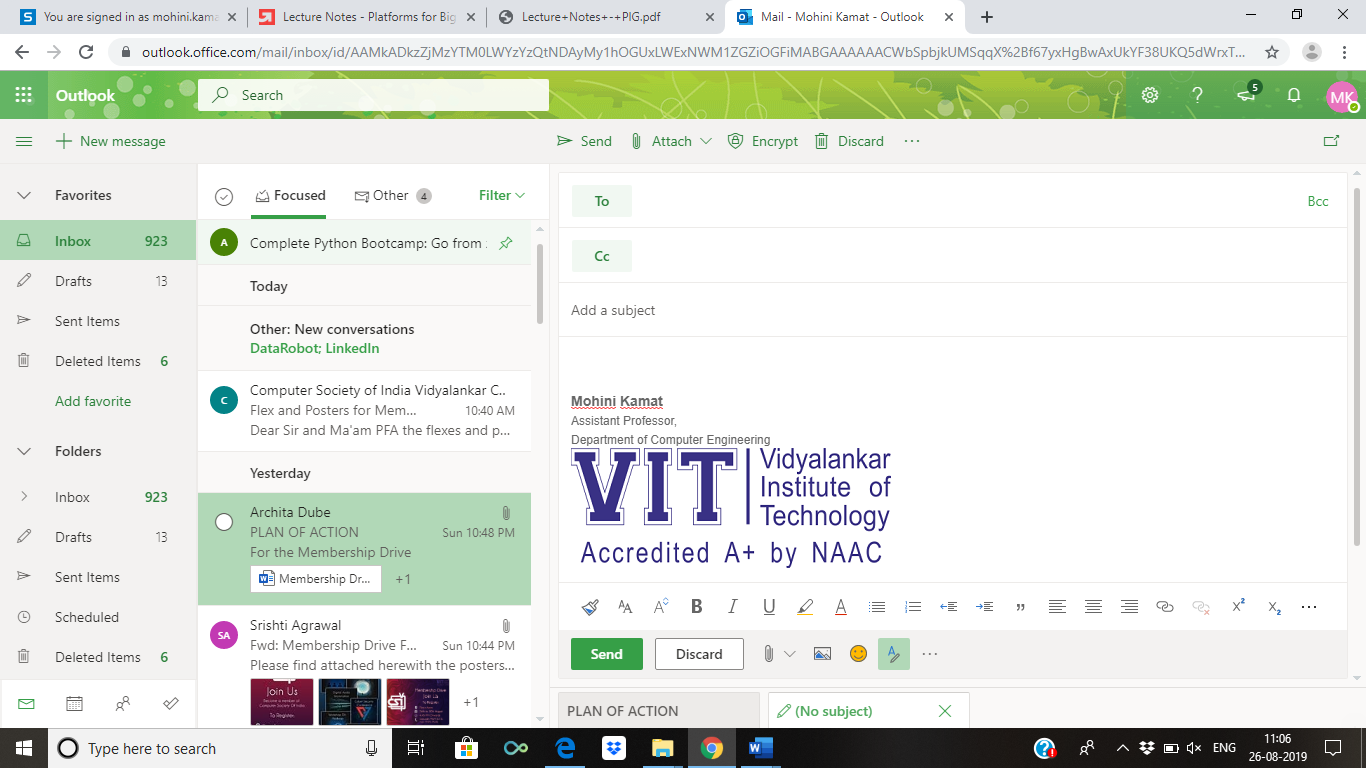
Deep Salunkhe 21102A0014

Pranav Redij21102A0005

Under the Guidance of

Prof. Pankaj Vanwari

Department of Computer Engineering



**(An Autonomous Institute Affiliated to University of Mumbai)**

Vidyalankar Institute of Technology

Wadala(E), Mumbai-400437

University of Mumbai

2022-23

**CERTIFICATE OF APPROVAL**

This is to certify that the project entitled

Banker Algorithm with GUI

is a bonafide work of

**Omkar Patil 21102A0003**

**Deep Salunkhe 21102A0014**

**Pranav Redij21102A0005**

submitted to the University of Mumbai in partial fulfillment of

**Mini Project (Operating System)**

for

Second Year of Computer Engineering

Guide Head of Department Principal

(Name)

Mini Project Report Approval

This project report entitled Banker Algorithm with GUI

by

***Omkar Patil 21102A0003***

***Deep Salunkhe 21102A0014***

***Pranav Redij21102A0005***

is approved for Mini Project (Operating System) for Second Year of Computer Engineering.

|  |  |
| --- | --- |
| Internal Examiner | External Examiner |

Date:

Place:

Declaration

I declare that this written submission represents my ideas in my own words and where others' ideas or words have been included, I have adequately cited and referenced the original sources. I also declare that I have adhered to all principles of academic honesty and integrity and have not misrepresented or fabricated or falsified any idea/data/fact/source in my submission. I understand that any violation of the above will be cause for disciplinary action by the Institute and can also evoke penal action from the sources which have thus not been properly cited or from whom proper permission has not been taken when needed.

Name of student Roll No. Signature

1) Omkar Patil 21102A0003

2)Deep Salunkhe 21102A0014

3)Pranav Redij 21102A0005

Date:

Place:

Acknowledgements

This Project wouldn’t have been possible without the support, assistance, and guidance of a number of people whom we would like to express our gratitude to. First, we would like to convey our gratitude and regards to our mentor **Prof. Pankaj Vanwari**

for guiding us with his constructive and valuable feedback and for his time and efforts. It was a great privilege to work and study under his guidance.

We would like to extend our heartfelt thanks to our Head of Department, **Dr. Sachin Bojewar** for overseeing this initiative which will in turn provide every Vidyalankar student a distinctive competitive edge over others.

We appreciate everyone who spared time from their busy schedules and participated in the survey. Lastly, we are extremely grateful to all those who have contributed and shared their useful insights throughout the entire process and helped us acquire the right direction during this research project.

Abstract

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

**Table of Contents**

|  |  |  |  |
| --- | --- | --- | --- |
| **Sr No** | **Description** | | **Page No** |
| 1 | Introduction | | **8** |
| 2  3 | Problem Definition  Implementation | | **9**  **9** |
| 4 | Conclusion | | **13** |
|  | |

**Introduction:**

The Banker's Algorithm is a resource allocation and deadlock avoidance algorithm that was first described by Edsger Dijkstra in 1965. The algorithm is used in operating systems to manage resources such as CPU, memory, and I/O devices.

The algorithm is based on the idea of considering the available resources in the system and the current allocation of resources to processes. Each process must declare in advance the maximum number of resources it needs and the algorithm ensures that the total allocation of resources to processes does not exceed the available resources in the system.

The algorithm works by simulating the allocation of resources to each process and checking if the system remains in a safe state after the allocation. If the system remains in a safe state, the allocation is considered to be valid and is allowed to proceed. If the allocation would cause the system to enter an unsafe state, it is rejected.

The Banker's Algorithm is important because it helps prevent deadlocks in resource allocation. Deadlock occurs when two or more processes are waiting for each other to release resources, causing all processes to come to a halt. By ensuring that a request for resources is granted only if it leaves the system in a safe state, the algorithm helps prevent deadlocks and ensures that resources are used efficiently.

The implementation of the Banker's Algorithm using Python and the Tkinter library provides a user-friendly interface for resource allocation management. The Tkinter library is a standard Python library for creating graphical user interfaces and provides a set of tools for creating windows, buttons, labels, and other graphical elements. Using Python and Tkinter, it is possible to create a simple yet effective implementation of the Banker's Algorithm that can be used to manage resources in an operating system.

**Problem Definition:**

The Banker's Algorithm is used to prevent deadlocks in resource allocation. Deadlock occurs when two or more processes are waiting for each other to release resources, causing all processes to come to a halt. The Banker's Algorithm ensures that a request for resources is granted only if it leaves the system in a safe state.

**Implementation:**

**Implementation:**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()

**Output:**A picture containing timeline

Description automatically generated

.

**Conclusion:**

In conclusion, the Banker's Algorithm is an important algorithm in operating systems that helps prevent deadlocks in resource allocation. The implementation of the algorithm using Python and the Tkinter library provides a user-friendly interface for resource allocation management. The project provides a basic implementation of the algorithm and can be extended to include additional features.