Karachi Campus

Discovering Knowledge

# OS Lab (CSL-320)

# ATM MACHINE

## Submitted By:


Raza Ali Khan (02-235221-029)

Anusha Mirza (02-235221-007)

Rida Alam (02-235221-038)


## Submitted To:

Fatima Zafar

## Date

January 10th 2024

## TABLE OF CONTENTS

## ❖ Abstract

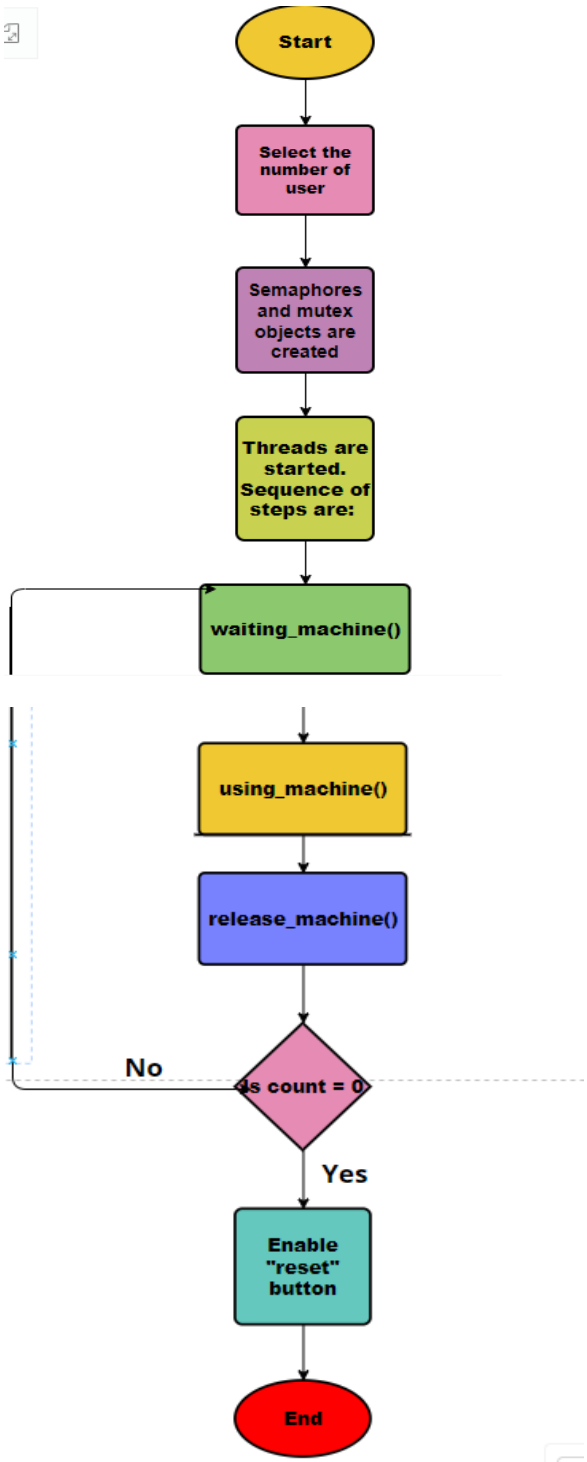
The ATM Simulator is a multi-threaded application built using Python's Tkinter library. It simulates users accessing an ATM machine concurrently. The application allows users to select the number of users and shows their interactions with the ATM. Each user is represented by an image, and their actions such as waiting for the machine, using the machine, and releasing the machine are displayed in a graphical interface.

## ❖ Introduction

The ATM Simulator project utilizes Python's Tkinter for creating a graphical user interface and threading to accommodate multiple users accessing an ATM simultaneously. The primary goal of this project is to demonstrate the concurrent behavior of users interacting with an ATM system. Users can select the number of concurrent users they wish to simulate, and the application visually represents each user's actions, such as waiting for the machine, using the machine, and releasing the machine.

The application presents a visual representation of users by a unique image, and shows their interactions with the ATM machine. Semaphore objects manage access to the machine, ensuring mutual exclusion and synchronized behavior among the users.

## ❖ Flowchart


Start
Select the number of user
Semaphores and mutex objects are created
Threads are started. Sequence of steps are:
waiting_machine()
using_machine()
release_machine()
Is count = 0
No
Yes
Enable "reset" button
End

- **Source code**

```
import tkinter as tk
from tkinter import ttk
from threading import Thread, Semaphore
from PIL import Image, ImageTk
import time

class Constants:
    man_ = ["man_black", "man_red", "man_yellow", "man_blue", "man_cyan"]

class App(tk.Tk):
    def _init_(self):
        super()._init_()



        self.number_of_users_text = ttk.Label(self, text="Type N° Users:", font=("Comic Sans MS", 18, "bold"))
        self.number_of_users_text.grid(row=0, column=0, padx=10, pady=10)

        self.number_of_user_combobox = ttk.Combobox(self, values=["1", "2", "3", "4", "5"])
        self.number_of_user_combobox.grid(row=0, column=1, padx=10, pady=10)
        self.number_of_user_combobox.set("4")

        self.counter_reset = int(self.number_of_user_combobox.get())
```

```
        self.btn_start = ttk.Button(self, text="Start", command=self.start_threads)
        self.btn_start.grid(row=0, column=2, padx=10, pady=10)

        self.image_atm = tk.Label(self, text="ATM Image Placeholder", font=("Comic Sans MS", 12))
        self.image_atm.grid(row=1, column=0, padx=10, pady=10)

        atm_image_path = "/home/pak/Downloads/ratm.jpg"  # Replace with the actual path
        atm_image = Image.open(atm_image_path)
        atm_image = atm_image.resize((110, 250), Image.ANTIALIAS if hasattr(Image, 'ANTIALIAS') else 3)
        self.atm_photo = ImageTk.PhotoImage(atm_image)
        self.image_atm.config(image=self.atm_photo)

        self.btn_reset = ttk.Button(self, text="Reset", command=self.reset_threads)
        self.btn_reset.grid(row=0, column=3, padx=10, pady=10)
        self.btn_reset["state"] = "disabled"

        self.waiting_text = tk.Label(self, text="File Release", font=("Comic Sans MS", 20))
        self.waiting_text.grid(row=2, column=0, padx=10, pady=10)

        self.man_images = []

        for i in range(len(Constants.man_)):
```

```
        man_image_path = f"/home/pak/Pictures/{Constants.man_[i]}.png"  # Replace with the actual paths
        man_image = Image.open(man_image_path)
        man_image = man_image.resize((60, 70), Image.ANTIALIAS if hasattr(Image, 'ANTIALIAS') else 3)
        man_photo = ImageTk.PhotoImage(man_image)

        man_image_label = tk.Label(self, image=man_photo)
        man_image_label.image = man_photo
        man_image_label.grid(row=5, column=i, padx=10, pady=10)

        self.man_images.append(man_image_label)

    self.processes_text = tk.Label(self, text="Processes :", font=("Comic Sans MS", 20))
    self.processes_text.grid(row=2, column=1, padx=10, pady=10)

    self.processes_text_area = tk.Text(self, width=40, height=10, font=("Comic Sans MS", 12), state=tk.DISABLED)
    self.processes_text_area.grid(row=3, column=1, padx=10, pady=10)

    self.release_text = tk.Label(self, text="File Waiting", font=("Comic Sans MS", 20))
    self.release_text.grid(row=2, column=2, padx=10, pady=10)

  def start_threads(self):
    self.btn_start["state"] = "disabled"
```

```
        self.number_of_user_combobox["state"] = "disabled"
        machine = Semaphore(1)
        mutex = Semaphore(1)
        self.counter_reset = int(self.number_of_user_combobox.get())
        for i in range(int(self.number_of_user_combobox.get())):
            Person(machine, Constants.man_[i], mutex, self)

    def reset_threads(self):
        for i in range(int(self.number_of_user_combobox.get())):
            self.man_images[i].place(x=0, y=0)
            self.man_images[i].config(state=tk.NORMAL)

        self.processes_text_area.config(state=tk.NORMAL)
        self.processes_text_area.delete("1.0", tk.END)
        self.processes_text_area.config(state=tk.DISABLED)

        self.counter_reset = int(self.number_of_user_combobox.get())
        self.number_of_user_combobox["state"] = "normal"
        self.btn_reset["state"] = "disabled"
        self.btn_start["state"] = "normal"

class Person(Thread):
    def _init_(self, machine, name, mutex, app):
        super()._init_(group=None)
        self.machine = machine
```

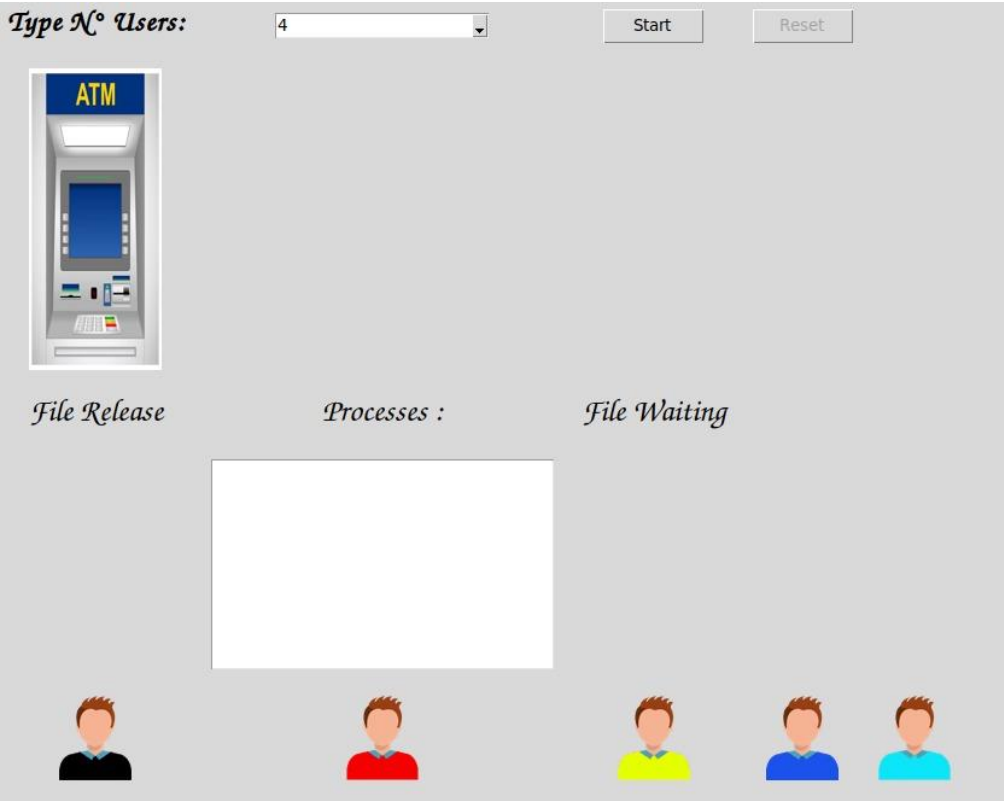
```
        self.name = name
        self.mutex = mutex
        self.app = app
        self.start()

    def run(self):
        try:
            self.waiting_machine()

            self.machine.acquire()
            self.using_machine()

            self.machine.release()
            self.release_machine()

            if self.app.counter_reset == 0:
                self.app.btn_reset["state"] = "normal"

        except Exception as ex:
            print(ex)

    def waiting_machine(self):
        try:
            self.mutex.acquire()
```

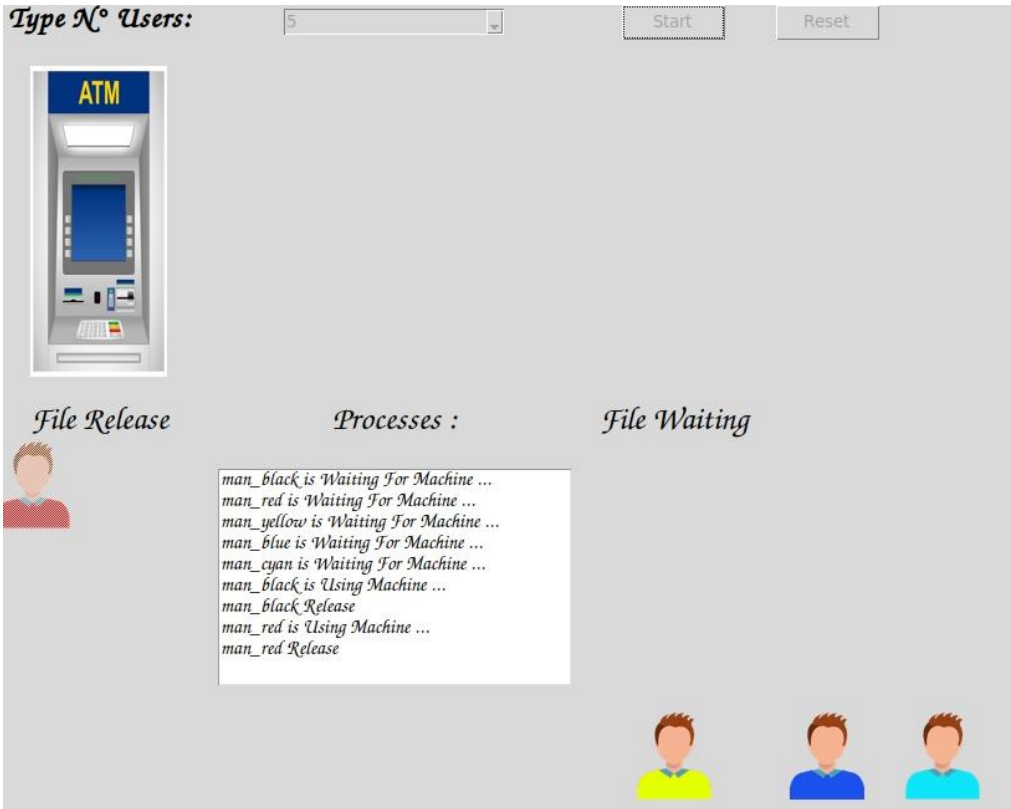
```
            for i in range(len(Constants.man_)):
                if self.name == Constants.man_[i]:
                    self.fade_transition(i)
                    break

            self.app.processes_text_area.config(state=tk.NORMAL)
            self.app.processes_text_area.insert(tk.END, f"{self.name} is Waiting For Machine ...\n")
            self.app.processes_text_area.config(state=tk.DISABLED)
            self.mutex.release()

        except Exception as ex:
            print(ex)

    def using_machine(self):
        try:
            time.sleep(3)

            for i in range(len(Constants.man_)):
                if self.name == Constants.man_[i]:
                    self.translate_transition(210 + (4 - i) * 90, 125, i)

            self.mutex.acquire()
            self.app.processes_text_area.config(state=tk.NORMAL)
            self.app.processes_text_area.insert(tk.END, f"{self.name} is Using Machine ...\n")
```

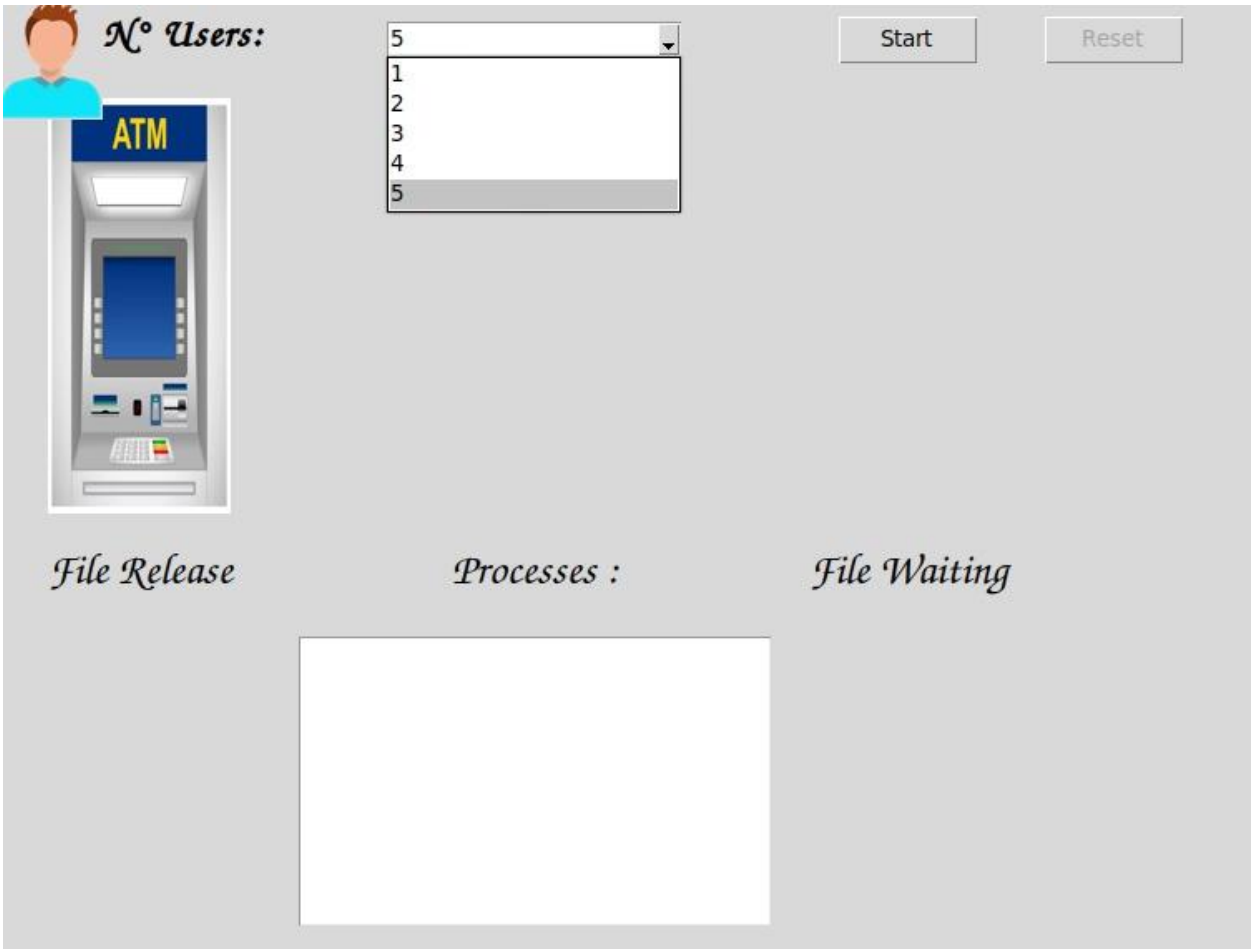
```
        self.app.processes_text_area.config(state=tk.DISABLED)
        self.mutex.release()
        time.sleep(3)

    except Exception as ex:
        print(ex)

def release_machine(self):
    try:
        self.mutex.acquire()

        for i in range(len(Constants.man_)):
            if self.name == Constants.man_[i]:
                self.translate_transition(0, 360, i)
                break

        self.app.processes_text_area.config(state=tk.NORMAL)
        self.app.processes_text_area.insert(tk.END, f"{self.name} Release\n")
        self.app.processes_text_area.config(state=tk.DISABLED)
        self.mutex.release()

        self.app.counter_reset -= 1
        time.sleep(2)

    except Exception as ex:
```

```
            print(ex)

    def translate_transition(self, x, y, i):
        self.app.man_images[i].place(x=x, y=y)
        self.app.man_images[i].config(state=tk.DISABLED)

    def fade_transition(self, i):
        self.app.man_images[i].config(state=tk.NORMAL)

if _name_ == "_main_":

    app = App()
    app.title("ATM Simulator")
    app.geometry("800x530")
    app.mainloop()
```

## ❖ Output screen shots




N° Users:
5
1
2
3
4
5
Start
Reset
ATM
File Release
Processes :
File Waiting

## ❖ Conclusion

The ATM Transaction project successfully demonstrates concurrent access to an ATM machine using Python's Tkinter library and threading functionalities. Through a graphical user interface multiple individuals interacting with the ATM concurrently can be seen. Semaphore objects regulate access to the ATM, ensuring that only one user can use the machine at a time while others wait for their turn.

The project's graphical representation effectively showcases the actions of users, such as waiting, using, and releasing the machine, allowing for a visual understanding of concurrent interactions. This application serves as an illustrative tool to comprehend the concept of concurrent access and synchronization in a simulated ATM environment.

## ❖ References


- [12] H. Zegai, "ATMSimulator: A Simple ATM Simulator," GitHub. [Online]. Available: https://github.com/HouariZegai/ATMSimulator

- [11] Victor Turrisi , "Semaphores on Python," Stack Overflow, Jul 20 '15. [Online]. Available: https://stackoverflow.com/questions/31508574/semaphores-on-python