**DEPARTMENT OF COMPUTER & SOFTWARE ENGINEERING**

**COLLEGE OF E&ME, NUST, RAWALPINDI**



**Subject Name: EC-221 Operating Systems**

**Project Report**

**Submitted To:**

**Professor Mehwish Naseer**

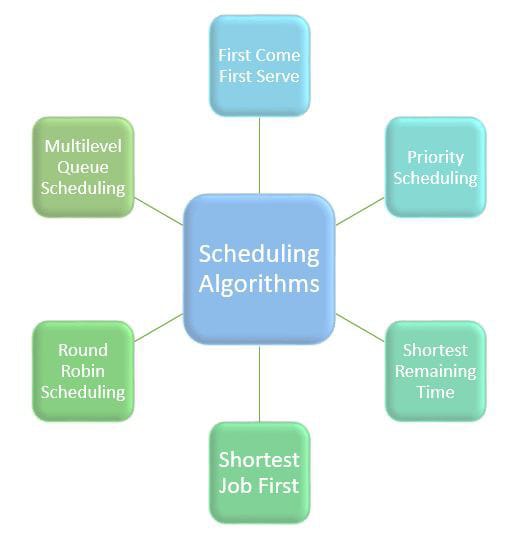
**Submitted By:**

**Usman Awan REG # 480836**

**DE - 45/SYN:A DEPT: Computer Engineering**

**Submission Date: 22/04/2025**

**Title: Operating System Scheduler**



**Table of contents:**

|  |  |  |
| --- | --- | --- |
| **S. no.** | **Contents** | **Page** |
| 1. | Operating System Scheduling(OSS) overview | 3 |
| 2. | Introduction to Problem | 4 |
| 3. | Objectives | 5 |
| 4. | Libraries and tools used | 5 |
| 5. | Design overview - UML | 6 |
| 6. | Source code overview   * Patient class * ResourcePool class * SimulationManager class * SimulationGUI class | 7 |
| 7. | Key features | 12 |
| 8. | Primary challenges | 12 |
| 9. | Functional overview - Packaging and Simulations | 13 |
| 10. | Conclusion and credits | 14 |

**Operating System Scheduler(OSS) overview:**

A diagram of operating systems

AI-generated content may be incorrect.**Operating System (OS)** plays a crucial role in managing processes and system resources to ensure smooth and efficient operation of a computer. One of its key functions is **process scheduling**, which decides the order in which processes access the CPU. The scheduler uses algorithms like **First-Come-First-Served (FCFS)**, **Round Robin**, **Shortest Job First (SJF)**, and **Priority Scheduling** to improve CPU utilization, reduce wait times, and enhance system responsiveness.

In addition to scheduling, the OS handles **resource management**, which involves the allocation and deallocation of critical system resources such as **memory**, **I/O devices**, and **CPU cycles**. Efficient resource management prevents conflicts and ensures fair access to resources among competing processes.

However, poor resource allocation can lead to **deadlocks**, where a group of processes is stuck waiting for each other’s resources indefinitely. To handle this, the OS may implement **deadlock prevention**, **avoidance**, or **detection and recovery** techniques. One popular deadlock avoidance method is the **Banker’s Algorithm**, which checks if resource allocation will leave the system in a safe state before proceeding. If not, the request is denied avoiding a potential deadlock.

Together, scheduling and resource management enable multitasking, maintain system stability, and ensure optimal performance of computing environments.

**----------------------------------------------------------------------------------------------------**

**Introduction to Problem:**

A diagram of a process

AI-generated content may be incorrect.This project is a Python-based simulation that models how an Operating System handles **resource synchronization, process synchronization**, and **deadlock management**. It implements a **three-level multilevel ready queue** system (q0, q1, and q2) for **process scheduling**. Processes are assigned to these queues based on their priority, with each queue supporting **time slicing scheduling** using a **5-second time quantum**. Each queue can hold up to five processes, and if a process doesn’t finish within its time slice, it is **pre-empted** to the next lower-priority queue. The CPU follows a fixed pattern: it executes **3 processes from q0**, then **2 from q1**, and **1 from q2**, before looping back to q0.

For **resource management**, the system includes **five types of resources** (r1 to r5), each with **five instances**. Processes can only execute if their required resources are available. To avoid **deadlocks**, the simulation uses **Banker's Algorithm**, which ensures that any resource allocation keeps the system in a safe state.

Each **process** in the simulation has a **unique ID**, a **priority level** (randomly assigned from 0 to 2), a **burst time** (randomly generated between 0 and 15 seconds), a list of **required resources**, and an **arrival time**. When the user presses a button in the GUI, **10 new processes** are generated with random attributes and placed into a **wait queue** if the scheduling queues are full.

The **graphical user interface (GUI)** displays a **Gantt chart** showing the last five executed processes, the **currently running process**, and the contents of all three scheduling queues. It also displays the **average turnaround time** and **average wait time**, which are updated each time a process completes. Additionally, a **resource allocation table** visually tracks the distribution of resource instances from unallocated to allocated, offering a clear view of system-wide resource usage.

**----------------------------------------------------------------------------------------------------**

**Objectives:**

* Simulate Process Scheduling using a three-level multilevel queue with time slicing and priority-based preemption.
* Implement Resource Management with multiple resource types and use Banker’s Algorithm to prevent deadlocks.
* Provide a Real-Time Graphical User Interface (GUI) that visualizes scheduling, process execution, and resource allocation.

**----------------------------------------------------------------------------------------------------**

A blue and yellow snake logo

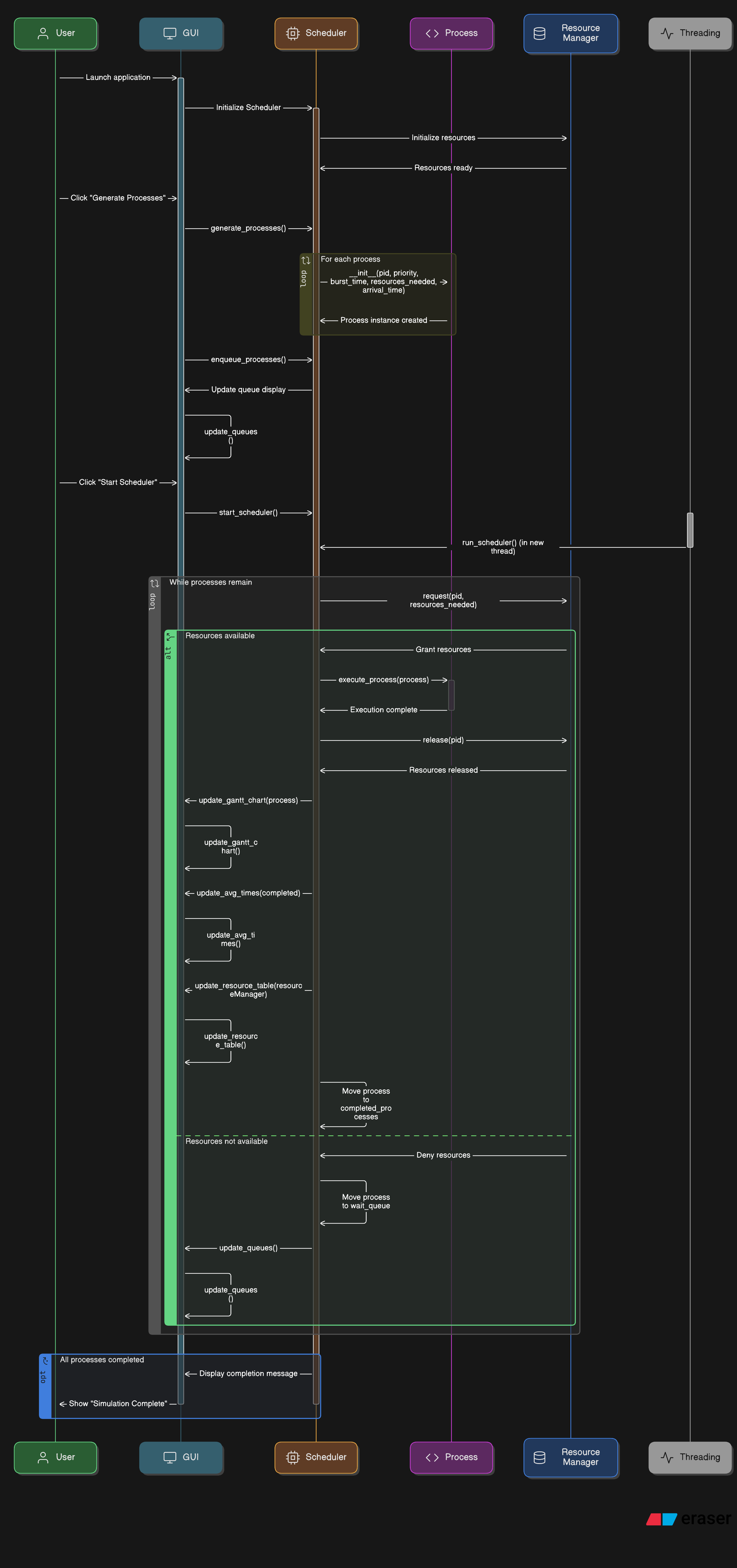
AI-generated content may be incorrect.**Libraries and tools used:**

* **Hardware/Software Required:**
  + **Hardware: PC**
  + **Software: Python3.13 , OpenJDK 23, Pycharm Community 2024.3.5 IDE**
* A diagram of a software

  AI-generated content may be incorrect.**Root Packages used:**
  + **tkinter**
  + **random**
  + **time**
  + **threading**

**----------------------------------------------------------------------------------------------------**

**Design overview – UML:**

****

**----------------------------------------------------------------------------------------------------**

**Source code overview:**

*# libraries*import tkinter as tk  
from tkinter import ttk  
import random  
import time  
import threading  
  
*# define class Process*class Process:  
 *# constructor* def \_\_init\_\_(self, pid, priority, burst\_time, resources\_needed, arrival\_time):  
 *# attributes* self.pid = pid  
 self.priority = priority  
 self.burst\_time = burst\_time  
 self.remaining\_time = burst\_time *# initially burst\_time equals remaining time and* self.resources\_needed = resources\_needed  
 self.arrival\_time = arrival\_time  
 self.start\_time = None  
 self.end\_time = None  
 self.wait\_time = 0  
  
*# define class Resource Manager*class resourceManager:  
 *# constructor* def \_\_init\_\_(self):  
 *# attributes* self.total = [5]\*5 *# total resources = 5* self.available = [5]\*5 *# total instances of each resource = 5* self.allocation = {pid: [0]\*5 for pid in range(1, 11)} *# initially no resource isnallocated  
  
 # class methods  
 ############################################################* def request(self, pid, resources\_needed): *# use Banker's algorithm to avoid deadlocks  
 # check if request is safe and grant if possible* if self.\_is\_safe(pid, resources\_needed):  
 for a in range(5): *# grant resources if resources are available* self.available[a] -= resources\_needed[a]  
 self.allocation[pid][a] += resources\_needed[a]  
 return True  
 return False  
 *############################################################* def \_is\_safe(self, pid, request): *# helper method for the request method  
 # check if granting request keeps system in safe state* temp = self.available[:] *# store resource list in temp variable* for a in range(5):  
 if request[a] > temp[a]: *# check instances of each resource* return False  
 return True  
 *############################################################* def release(self, pid): *# release resources when execution is over  
 # release all resources held by the process* if pid in self.allocation:  
 for a in range(5):  
 self.available[a] += self.allocation[pid][a]  
 self.allocation[pid] = [0]\*5  
  
  
*# define class Scheduler*class Scheduler:  
 *# constructor* def \_\_init\_\_(self, gui):  
 *# attributes* self.q0 = []  
 self.q1 = []  
 self.q2 = []  
 self.wait\_queue = []  
 self.resource\_manager = resourceManager()  
 self.completed\_processes = []  
 self.gui = gui  
 self.time\_quantum = 5  
 self.pid\_counter = 1  
 self.running = False  
 self.lock = threading.Lock()  
  
 *# class methods  
 ############################################################* def generate\_processes(self): *# generate processes  
 # generate 10 random processes and put them in the wait queue* for a in range(10):  
 pid = self.pid\_counter *# assign id in correspondence to last used id* priority = random.randint(0, 2) *# queue q0-q2 correspond to priority 0-2* burst\_time = random.randint(1, 10) *# max 10s  
  
 # each process can request 2 instances of 5 different resources at max* resources\_needed = [random.randint(0, 2) for a in range(5)]  
  
 arrival\_time = time.time() *# assign current time to process  
  
 # initialize the process* process = Process(pid, priority, burst\_time, resources\_needed, arrival\_time)  
  
 *# update attributes of scheduler class* self.wait\_queue.append(process)  
 self.pid\_counter += 1  
  
 *# update attributes of GUI class* self.gui.update\_queues()  
 *############################################################* def enqueue\_processes(self): *# put processes in queue  
 # move processes from wait queue to appropriate ready queues* while self.wait\_queue: *# iterate till wait queue list is not empyy* temp\_process = self.wait\_queue[0]  
 queue = getattr(self, f"q{temp\_process.priority}") *# put process in required queue* if len(queue) < 5: *# check if queue is full* queue.append(self.wait\_queue.pop(0))  
 else:  
 break  
  
 *# update attributes of gui class* self.gui.update\_queues()  
 *############################################################* def start\_scheduler(self): *# start OS scheduling* if not self.running:  
 self.running = True  
 threading.Thread(target=self.run\_scheduler).start()  
 *############################################################* def run\_scheduler(self): *# schedules processes  
 # main scheduler loop: executes 3 from q0, 2 from q1, 1 from q2* while self.running:  
 for q, count in zip([self.q0, self.q1, self.q2], [3, 2, 1]):  
 for a in range(count):  
 if q: *# if target queue is not-empty, execute its process* process = q.pop(0)  
 if self.resource\_manager.request(process.pid, process.resources\_needed):  
 self.execute\_process(process)  
 else: *# if target queue is empty, move to next queue* q.append(process)  
  
 *# update attributes of scheduler class* self.enqueue\_processes()  
  
 *# delay to avoid inconsistent behaviour* time.sleep(1)  
 *############################################################* def execute\_process(self, process): *# execute process  
 # run process for up to time quantum = 5s* self.gui.update\_current\_process(process) *# update gui* if process.start\_time is None: *# record start time* process.start\_time = time.time()  
 start = time.time() *# start time slice  
  
 # execute process till time quantum or till it's done* while process.remaining\_time > 0 and time.time() - start < self.time\_quantum:  
 time.sleep(1)  
 process.remaining\_time -= 1  
 self.gui.update\_current\_process(process) *# update remaining time* if process.remaining\_time <= 0: *# if process completed* process.end\_time = time.time() *# record end time* self.completed\_processes.append(process) *# append process to complete queue* self.resource\_manager.release(process.pid) *# release processes resources* self.gui.update\_gantt\_chart(process) *# update gantt chart* else: *# if process did not complete  
 # preempt to lower priority queue* new\_priority = min(process.priority + 1, 2) *# update priority according to queue* process.priority = new\_priority  
 getattr(self, f"q{new\_priority}").append(process)  
 self.resource\_manager.release(process.pid)  
  
 *# update attributes of gui class* self.gui.update\_queues()  
 self.gui.update\_avg\_times(self.completed\_processes)  
 self.gui.update\_resource\_table(self.resource\_manager)  
  
*# define class GUI*class GUI:  
 *# constructor* def \_\_init\_\_(self, root):  
 *# attributes* self.root = root  
 self.root.title("OS Simulation")  
 self.root.configure(bg="#f0f8ff")  
 self.scheduler = Scheduler(self)  
 self.setup\_ui()  
  
 *# class methods  
 ############################################################* def setup\_ui(self): *# gui layout  
 # buttons to generate processes and start scheduler* self.button = tk.Button(self.root, text="Generate Processes", command=self.scheduler.generate\_processes, bg="#add8e6", font=("Arial", 12))  
 self.button.pack(pady=5)  
  
 self.start\_button = tk.Button(self.root, text="Start Scheduler", command=self.scheduler.start\_scheduler, bg="#90ee90", font=("Arial", 12))  
 self.start\_button.pack(pady=5)  
  
 *# queues display* self.queues\_label = tk.Label(self.root, text="Queues", font=("Arial", 14, "bold"), bg="#f0f8ff")  
 self.queues\_label.pack()  
 self.queues\_text = tk.Text(self.root, height=10, width=100, bg="#e6f7ff")  
 self.queues\_text.pack()  
  
 *# current process display* self.current\_process\_label = tk.Label(self.root, text="Current Process", font=("Arial", 14, "bold"), bg="#f0f8ff")  
 self.current\_process\_label.pack()  
 self.current\_process\_text = tk.Text(self.root, height=2, width=100, bg="#fffacd")  
 self.current\_process\_text.pack()  
  
 *# gantt chart display* self.gantt\_label = tk.Label(self.root, text="Gantt Chart", font=("Arial", 14, "bold"), bg="#f0f8ff")  
 self.gantt\_label.pack()  
  
 self.gantt\_text = tk.Text(self.root, height=2, width=100, bg="#ffe4e1") *# Corrected to self.gantt\_text* self.gantt\_text.pack()  
  
 *# average times display* self.average\_label = tk.Label(self.root, text="Avg Turnaround and Wait Time", font=("Arial", 14, "bold"),  
 bg="#f0f8ff")  
 self.average\_label.pack()  
  
 self.avg\_text = tk.Text(self.root, height=2, width=100, bg="#fafad2") *# Corrected to self.avg\_text* self.avg\_text.pack()  
  
 *# Resource table display* self.resource\_label = tk.Label(self.root, text="Resource Allocation Table", font=("Arial", 14, "bold"), bg="#f0f8ff")  
 self.resource\_label.pack()  
 self.resource\_text = tk.Text(self.root, height=6, width=100, bg="#e0ffff")  
 self.resource\_text.pack()  
 *############################################################* def update\_queues(self): *# update queue on gui  
 # refresh all queues and wait queue display* text = ""  
 for a in range(3):  
 queue = getattr(self.scheduler, f"q{a}")  
 text += f"Queue q{a}: {[p.pid for p in queue]}\n"  
 text += f"Wait Queue: {[p.pid for p in self.scheduler.wait\_queue]}"  
 self.queues\_text.delete(1.0, tk.END)  
 self.queues\_text.insert(tk.END, text)  
 *############################################################* def update\_current\_process(self, process): *# update current process on gui  
 # Display currently running process with live remaining time* self.current\_process\_text.delete(1.0, tk.END)  
 self.current\_process\_text.insert(tk.END, f"Process {process.pid}, Priority {process.priority}, Remaining {process.remaining\_time}s")  
 *############################################################* def update\_gantt\_chart(self, process): *# update gantt chart  
 # show last 5 executed processes* self.gantt\_text.insert(tk.END, f"P{process.pid} ")  
 lines = self.gantt\_text.get(1.0, tk.END).split()  
 if len(lines) > 5:  
 self.gantt\_text.delete(1.0, tk.END)  
 self.gantt\_text.insert(tk.END, " ".join(lines[-5:]))  
 *############################################################* def update\_avg\_times(self, completed): *# update average time on gui  
 # update average turnaround and wait time display* if not completed:  
 return  
 total\_ta\_t = sum(p.end\_time - p.arrival\_time for p in completed)  
 total\_w\_t = sum(p.start\_time - p.arrival\_time for p in completed)  
 avg\_tat = total\_ta\_t / len(completed)  
 avg\_w\_t = total\_w\_t / len(completed)  
 self.avg\_text.delete(1.0, tk.END)  
 self.avg\_text.insert(tk.END, f"Avg TAT: {avg\_tat:.2f}s, Avg WT: {avg\_w\_t:.2f}s")  
 *############################################################* def update\_resource\_table(self, rm): *# update resource table on gui  
 # Display current available resources and allocation table* text = f"Available: {rm.available}\n"  
 text += f"Allocations:\n"  
 for pid, alloc in rm.allocation.items():  
 text += f"P{pid}: {alloc}\n"  
 self.resource\_text.delete(1.0, tk.END)  
 self.resource\_text.insert(tk.END, text)  
  
*# Entry point*if \_\_name\_\_ == "\_\_main\_\_":  
 root = tk.Tk()  
 gui = GUI(root)  
 root.mainloop()

**----------------------------------------------------------------------------------------------------**

**Key features:**

* GUI interface for user friendly interaction.
* Regulate Process generation
* Gantt chart for scheduling interpretation
* Summary report

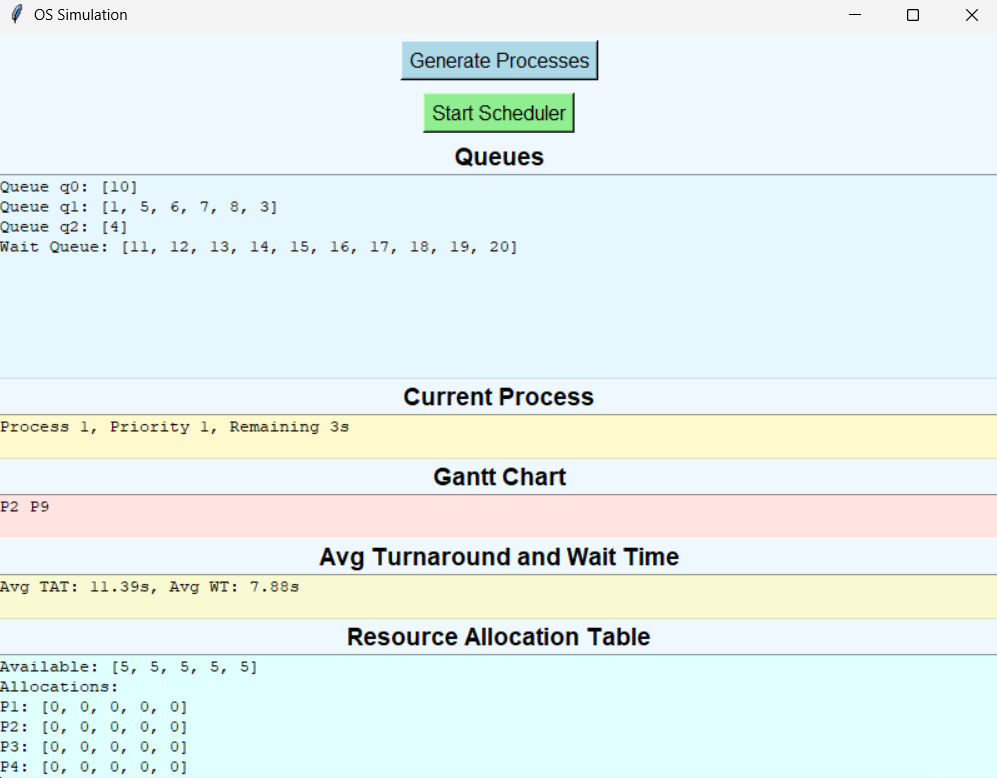
**------------------------------------------------------------------------------------------------**

**Primary challenges:**

* Implementation of multilevel queue
* Implementation of deadlock avoidance using Banker’s algorithm.
* Implementation of time slicing for each queue
* Implementation if GUI to display the whole simulation in a user-friendly way.

**------------------------------------------------------------------------------------------------**

**Functional overview - GUI:**

****

**----------------------------------------------------------------------------------------------------**

**Conclusion and credits:**

In making this project, all the fundamentals of the operating systems learnt throughout the semester were revitalized. I learnt many new concepts like programming multi-queue systems and resource management systems using the python programming language. Overall, it was a great learning experience.

**Special Thanks!**

Professor. Mehwish Naseer

**References:**

https://en.wikipedia.org/wiki/Operating\_system

https://en.wikipedia.org/wiki/Scheduling\_(computing)

---------------------The END--------------------

**----------------------------------------------------------------------------------------------------**