

Project Title:

**The Room Party Problem**

Submitted by:

**Ebaa Haq 2021-CE-22**

**Maham Nadeem 2021-CE-10**

**Faiza Riaz 2021-CE-20**

**Sana Israr 2021-CE-55**

Submitted to:

**Ma’am Darakhshan Abdul Ghaffar**

Course:

**CMPE-331L: Operating Systems Lab**

Semester:

**Fall 2023**

Date of Submission:

**[December 27, 2023]**

**Department of Computer Engineering**

**University of Engineering and Technology, Lahore**

Table of Contents

[Abstract 2](#_Toc154428563)

[1. Introduction 3](#_Toc154428564)

[2. Problem statement 3](#_Toc154428565)

[3. Solution to the problem 4](#_Toc154428566)

[4. Results and Conclusion 9](#_Toc154428567)

[5. Future work and Applications in Operating systems 13](#_Toc154428568)

[References 14](#_Toc154428569)

# 

# Abstract

The "The Room Party Problem" is a classic synchronization problem that simulates a situation involving students and the Dean of Students in a college setting. The objective of this project is to develop a synchronization solution that enforces several constraints to ensure the orderly entry and exit of students and the Dean of Students into and out of a room, particularly when a party is in progress or a search operation is being conducted. The problem arises from a real-life incident and aims to provide a fair and efficient solution to resolve this dilemma.

# Introduction

Operating systems are fundamental to the efficient and reliable functioning of computer systems. Within the field of operating systems, synchronization mechanisms are crucial to manage shared resources and ensure concurrent processes run smoothly. This project proposal addresses the development and implementation of a synchronization solution for the "Room Party Problem," which is a complex concurrency problem inspired by real-life events at an academic institution. [1]

# Problem statement

In an academic setting, there was a controversial incident where a student alleged that their room had been searched in their absence by the Dean of Students Office. To address scenarios like this, we propose solving the "The Room Party Problem." This problem simulates a situation in which students gather in a room, and the Dean of Students needs to enter the room under specific conditions, either to search the room or break up a party.

Below are the constraints and scenarios for the Room Party Problem:

**Constraints:**

1. **Number of Students:**
   1. Any number of students can be in a room at the same time.
2. **Dean's Entry Conditions:**
   1. The Dean of Students can only enter a room if there are 50 or more than 50 students in the room (to break up the party).
3. **Dean Inside the Room:**
   1. While the Dean of Students is in the room, no additional students may enter, but students may leave.
4. **Dean's Exit Conditions:**
   1. The Dean of Students may not leave the room until all students have left.
5. **Single Dean:**
   1. There is only one Dean of Students, so exclusion among multiple deans is not a concern.

**Scenarios:**

1. **Dean Conducting a Search:**
   1. Scenario: The Dean arrives, finds no students in the room, conducts a search, and leaves.
2. **Dean Waiting Due to Few Students:**
   1. Scenario: The Dean arrives, finds less than 50 students, and has to wait for more students to arrive before breaking up the party.
3. **Dean Breaking Up the Party:**
   1. Scenario: The Dean arrives, finds 50 or more students, breaks up the party, and waits for all students to leave.
4. **Students Entering and Leaving:**
   1. Scenario: Students enter the room while the Dean is not present.
   2. Scenario: Students enter the room while the Dean is waiting.
   3. Scenario: Students leave the room while the Dean is inside.
5. **Dean's Actions After Party is Broken Up:**
   1. Scenario: The Dean leaves the room after conducting a search.

# Solution to the problem

The Room Party Problem necessitates a sophisticated synchronization solution to manage the interaction between students and the Dean of Students effectively. Our proposed solution involves the use of semaphores to enforce various constraints and ensure the orderly entry and exit of individuals from the room.

**Key Components of the Solution:**

1. **Semaphores:**
   1. Utilizing semaphores for mutual exclusion and coordination.
   2. Semaphore **mutex** ensures exclusive access to critical sections.
   3. Semaphores **turn**, **clear**, and **lieIn** facilitate controlled interactions between students and the dean.
2. **Dean's Entry:**
   1. The solution allows the dean to enter the room under specific conditions:
      1. If the room is empty, the dean conducts a search.
      2. If the number of students is less than 50, the dean cannot enter and suggests adding more students.
      3. Otherwise, the dean can enter, prompting the students to leave the room orderly.
3. **Students' Entry and Party Management:**
   1. Students can enter the room and initiate a party if the dean is not present.
   2. If the dean is in the room, students coordinate their actions to signal the dean to break up the party when the count reaches 50.
4. **User Interface (GUI):**
   1. The GUI component provides a user-friendly interface, enhancing the simulation experience by allowing users to visualize and simulate the interactions between students and the dean. The GUI contributes to a more engaging and informative user experience.
5. **Code**

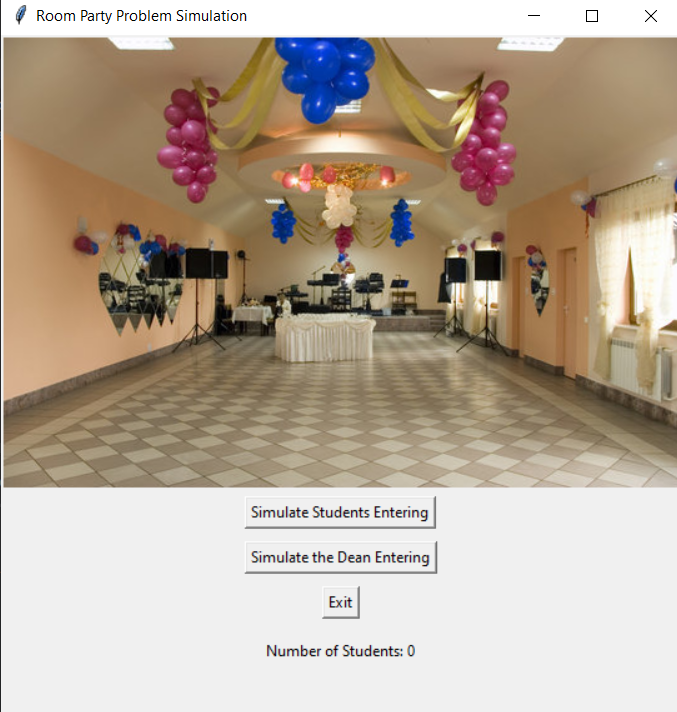
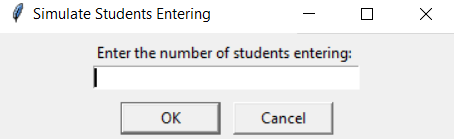
The code of Solution is:

*"""  
Created on 15th November 2023 18:56:34  
  
 Project Title:  
 The Room Party Problem  
 From:  
 The Little Book of Semaphores  
 Allen B. Downey  
 Version 2.2.1  
  
 Submitted by:  
  
 Ebaa Haq 2021-CE-22  
 Faiza Riaz 2021-CE-20  
 Maham Nadeem 2021-CE-10  
 Sana Israr 2021-CE-55  
  
 Submitted to:  
 Ma’am Darakhshan Abdul Ghaffar  
  
 Course:  
 CMPE-331L: Operating Systems Lab  
  
 Semester:  
 Fall 2023 (5th)  
  
  
  
"""*import tkinteras tk  
from PIL import Image, ImageTk  
from tkinterimport simpledialog, messagebox  
import threading  
  
class RoomPartyProblem:  
def \_\_init\_\_(self):  
self.students = 0  
self.dean\_state = 'not here'  
self.mutex = threading.Semaphore(1)  
self.turn = threading.Semaphore(1)  
self.clear = threading.Semaphore(0)  
self.lieIn = threading.Semaphore(0)  
  
def dean\_enters(self):  
self.mutex.acquire()  
  
if self.students == 0:  
messagebox.showinfo("Dean", "Dean arrives for searching purposes.")  
self.search\_room()  
self.dean\_state = 'not here'  
self.mutex.release()  
elifself.students<50:  
messagebox.showinfo("Dean", "Dean: Number of students is less than 50. Dean cannot enter. Add more students.")  
self.mutex.release()  
else:  
messagebox.showinfo("Dean", "Dean arrives.")  
self.dean\_state = 'in the room'  
while self.students>0:  
num\_students\_to\_remove = simpledialog.askinteger("Dean", "Dean: How many students do you want to remove?")  
if num\_students\_to\_removeis None:  
num\_students\_to\_remove = 0  
if num\_students\_to\_remove>self.students:  
num\_students\_to\_remove = self.students  
self.students -= num\_students\_to\_remove  
print(f"Removed{num\_students\_to\_remove} students. Updated number of students: {self.students}")  
  
# Update the label after removing students in the GUI  
gui\_instance.update\_removed\_label(num\_students\_to\_remove, self.students)  
  
self.break\_up\_party()  
self.turn.acquire() # lock the turnstile  
self.mutex.release()  
self.clear.acquire() # and get mutex from the student.  
self.ask\_dean\_leave()  
self.turn.release() # unlock the turnstile  
  
def update\_labels(self, num\_removed\_students):  
# Update the label displaying the number of removed and remaining students in the GUI  
gui\_instance.update\_removed\_label(num\_removed\_students, self.students)  
  
def break\_up\_party(self):  
messagebox.showinfo("Dean", "Dean: Breaking up the party.")  
  
def ask\_dean\_leave(self):  
leave\_decision = messagebox.askyesno("Dean", "Dean: Do you want to leave the room?")  
if leave\_decision:  
messagebox.showinfo("Dean", "Dean: Leaving the room.")  
else:  
self.menu()  
  
def search\_room(self):  
messagebox.showinfo("Dean", "Dean: Searching the room.")  
  
def student\_enters(self, num\_students, dean\_waiting=False):  
if dean\_waiting:  
self.mutex.acquire()  
  
if num\_studentsis not None:  
print(f"{num\_students} students enter. Students in the room:", self.students + num\_students)  
  
if self.dean\_state == 'in the room':  
self.mutex.release()  
self.turn.acquire()  
self.turn.release()  
self.mutex.acquire()  
self.students += num\_students  
  
if self.students>= 50 and self.dean\_state == 'waiting':  
self.lieIn.release() # and pass mutex to the dean  
print("Student: Signaling Dean to break up the party.")  
else:  
print("Student: Having a party.")  
self.mutex.release()  
  
self.mutex.acquire()  
self.students -= num\_students  
print("Students leave. Students in the room:", self.students)  
  
if self.students == 0 and self.dean\_state == 'waiting':  
self.lieIn.release() # and pass mutex to the dean  
print("Student: Signaling Dean to leave after all students left.")  
self.ask\_dean\_leave()  
elifself.students == 0 and self.dean\_state == 'in the room':  
self.clear.release() # and pass mutex to the dean  
print("Student: Signaling Dean to leave after breaking up the party.")  
self.ask\_dean\_leave()  
else:  
self.mutex.release()  
  
elifdean\_waiting:  
self.mutex.release()  
  
else:  
print("Student: Dean is not in the room. Having a party.")  
self.students += num\_students# Add students to the party count  
self.mutex.release()  
  
def run\_simulation(self):  
self.menu()  
  
def menu(self):  
while True:  
print("\nMenu:")  
print("1. Simulate Students Entering")  
print("2. Simulate the Dean Entering")  
print("3. Exit")  
  
 choice = input("Enter your choice (1, 2, or 3): ")  
if choice == '1':  
num\_students = int(input("Enter the number of students entering: "))  
self.student\_enters(num\_students)  
elifchoice == '2':  
self.dean\_enters()  
elifchoice == '3':  
break  
 else:  
print("Invalid choice. Please enter 1, 2, or 3.")  
  
  
  
# GUI Class  
classRoomPartyProblemGUI:  
def \_\_init\_\_(self, problem\_instance):  
self.root = tk.Tk()  
self.root.title("Room Party Problem Simulation")  
self.problem\_instance = problem\_instance  
  
# Load the image using Pillow  
self.pil\_image = Image.open("img.jpg")  
# Convert the Pillow image to PhotoImage  
self.image = ImageTk.PhotoImage(self.pil\_image)  
  
# Create an image label  
self.image\_label = tk.Label(self.root, image=self.image)  
self.image\_label.pack()  
  
# Create buttons  
self.button\_students = tk.Button(self.root, text="Simulate Students Entering", command=self.simulate\_students)  
self.button\_students.pack(pady=5)  
  
self.button\_dean = tk.Button(self.root, text="Simulate the Dean Entering", command=self.schedule\_dean\_enters)  
self.button\_dean.pack(pady=5)  
  
self.button\_exit = tk.Button(self.root, text="Exit", command=self.exit\_simulation)  
self.button\_exit.pack(pady=5)  
  
# Add a label for displaying the number of students  
self.label\_students = tk.Label(self.root, text=f"Number of Students: {self.problem\_instance.students}")  
self.label\_students.pack(pady=10)  
  
# Add a label for displaying the removed and remaining students  
self.label\_removed\_students = tk.Label(self.root, text="")  
self.label\_removed\_students.pack(pady=5)  
  
def simulate\_students(self):  
num\_students = simpledialog.askinteger("Simulate Students Entering", "Enter the number of students entering: ")  
self.problem\_instance.student\_enters(num\_students)  
self.label\_students.config(text=f"Number of Students: {self.problem\_instance.students}")  
messagebox.showinfo("Success", f"Successfully added {num\_students} students!")  
  
def schedule\_dean\_enters(self):  
self.problem\_instance.dean\_enters()  
  
def update\_removed\_label(self, num\_removed\_students, num\_remaining\_students):  
# Update the label displaying the number of removed and remaining students in the GUI  
self.label\_removed\_students.config(text=f"Removed{num\_removed\_students} students. Updated number of students: {num\_remaining\_students}")  
  
def exit\_simulation(self):  
if self.problem\_instance.students>0:  
messagebox.showinfo("Error", "Cannot exit. Students are still in the room.")  
else:  
self.root.destroy()  
  
def run\_simulation(self):  
self.root.mainloop()  
  
# Run the simulation  
problem\_instance = RoomPartyProblem()  
gui\_instance = RoomPartyProblemGUI(problem\_instance)  
gui\_instance.run\_simulation()

# Results and Conclusion

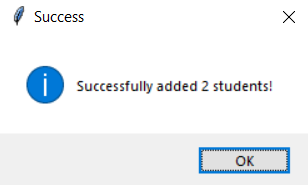
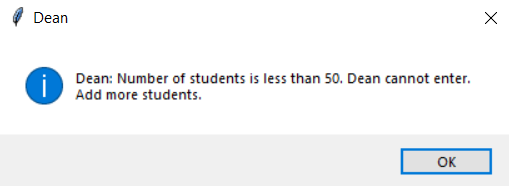
The implementation of the Room Party Problem simulation yields insightful results regarding the effectiveness of the synchronization solution. Key observations include:

* Orderly entry and exit of students and the dean.
* Successful party interruption and search operations.
* Responsive GUI updates to reflect real-time changes in the number of students.



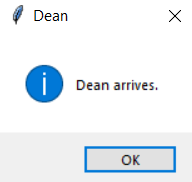
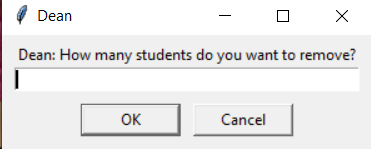
**Main Window of The Room Part Problem**

**Dialog box of Students Entering**



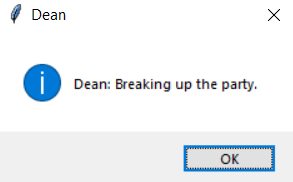
**Message Box of Dean is Waiting**

**Message box of Students Entering Success**

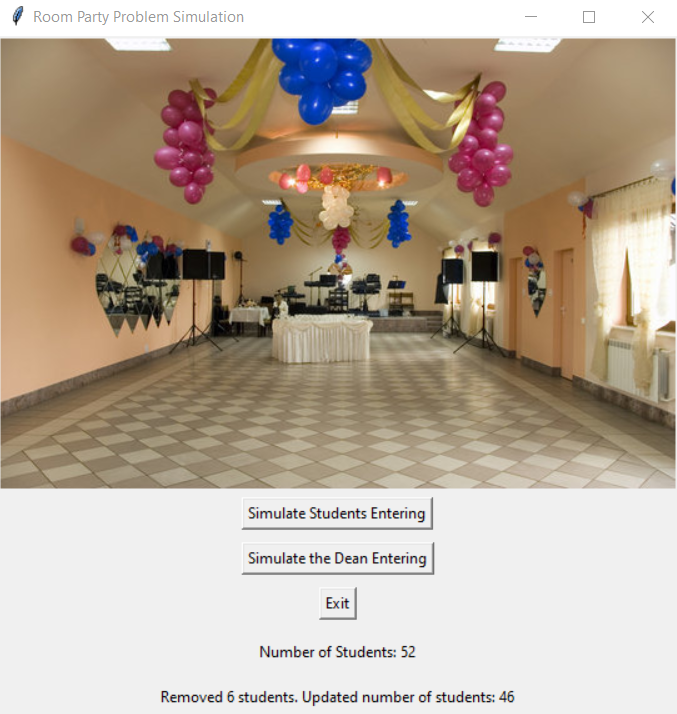


**Dialog Box of Students Leaving**

**Message Box that Dean arrives**



**Message Box of Breaking up the Party**



**Main Window after Breaking up the Party**

1. **Future work and Applicationsin Operating systems**

The future work of The Room Party Problem is:

1. **Dynamic Room Characteristics:**

The future work for the Room Party Problem involves incorporating dynamic characteristics of the room, introducing elements such as varying room sizes, multiple rooms, or changing conditions within the room. This extension aims to enhance the realism and challenge of the synchronization problem, making it adaptable to diverse scenarios.

1. **Graphical User Interface (GUI) Enhancements:**

Improving the user interface (GUI) of the simulation is identified as a key area for future work. Enhancements may include creating a more interactive and visually informative experience for users. This could involve better visualization of the room layout, integration of status indicators, and providing real-time updates during the simulation.

Its Applications in Operating System are:

1. **Concurrency Management in Real-Life Scenarios:**

The synchronization solution developed for the Room Party Problem can find applications in real-life scenarios requiring concurrent access to shared resources. Beyond room parties, this concept can be extended to scenarios such as resource allocation in a shared workspace, contributing to effective concurrency management.

1. **Education and Training:**

The Room Party Problem serves as a valuable teaching tool for operating systems and concurrency concepts. Future work involves developing educational materials and simulations that assist students in understanding synchronization challenges within a practical context, facilitating effective education and training.

1. **Event Management Systems:**

The concept of the Room Party Problem can be extended to simulate event management systems. This application involves synchronizing attendees, organizers, and specific conditions within the system. The developed synchronization principles can be instrumental in designing software for coordinating large-scale events seamlessly.

1. **Security and Access Control:**

Adapting the synchronization principles to scenarios involving security and access control is a promising application. This includes controlling entry to secure areas based on specific conditions or permissions. The Room Party Problem's synchronization solution can contribute to robust security and access control mechanisms.

1. **Smart Building Systems:**

The synchronization techniques developed for the Room Party Problem can be applied to simulate smart building systems. This application envisions different entities, such as occupants and security personnel, requiring coordinated access to rooms or facilities based on various conditions. It offers practical solutions for managing smart building environments.

1. **Collaborative Environments:**

Exploring the application of the synchronization solution to simulate collaborative environments is crucial. This involves scenarios where multiple users or entities interact within a shared space while adhering to specific rules or conditions. The Room Party Problem's synchronization approach can enhance the coordination and collaboration in such environments.

# References

1. [Downey, A. B. (Version 2.2.1). The Little Book of Semaphores.](https://greenteapress.com/wp/semaphores/)
2. <https://blog.ksub.org/bytes/2016/05/22/the-room-party-problem/>

[3] <https://www.tutorialspoint.com/semaphores-in-operating-system>