

Faculty of Engineering & Technology

Electrical & Computer Engineering Department

OS Project

Thread Management Simulator

Rivan jaradat :1200081

Rawan sous:1200129

Dana ismail:1200006

حفظ الترجمة

This week,we'll show each person's accomplishments

Rivan:

My new work :

def creatThread(self ,threadname,id,enteryPoint):

        thread=Thread(threadname,id,enteryPoint)

        self.append(thread)

    def terminate\_thread(self,Id):

       current = self.head

       previous = None

       while current:

            if current.id == Id:

                break

            previous = current

            current = current.next

    def thread\_command(self):

        i = 0

        while True:

            thread\_id = input(f'Enter thread ID for thread {i+1}: ')

            name = input(f'Enter thread name for thread {i+1}: ')

            entry\_point = input(f'Enter entry point for thread {i+1}: ')

            self.create\_thread(name, thread\_id, entry\_point)

            i += 1

until now I do this :

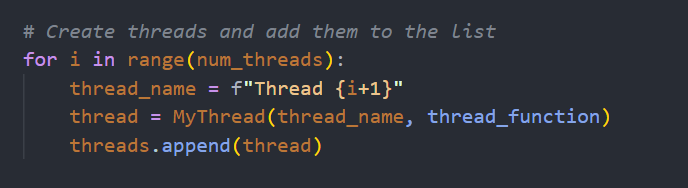
Node Class: This class represents a node in the linked list and contains attributes such as id, name, currentState, and next. Each node represents a thread and holds information about its ID, name, and current state.

Linked List Class: This class manages the threads using a linked list structure. It has methods to check if the list is empty, append new threads to the list, delete threads by ID, and print the list of threads. Additionally, it includes a method, creatThread, to create and append a new thread to the linked list based on the provided name, ID, and entry point function.

Thread Class: This class extends the threading.Thread class and represents an individual thread. It has attributes for thread\_name, thread\_ID, and entry\_point, which store the name, ID, and entry point function for the thread, respectively. The run method is overridden to execute the thread's entry point function when the thread is started.

**Dana’s work :**

Try to modify the code in order to add the threads using while loop



Using synchronization techniques like locks and semaphores, the provided Python code creates and controls multiple threads. To achieve mutual exclusion, it simulates thread execution and context switching and makes sure that only one thread can execute the crucial section at once. The code makes use of global variables, synchronization objects, a custom MyThread class, functions for thread execution, and context switching. The code enables flexibility in the number of threads generated by creating threads dynamically in a loop. Overall, it provides an example of how to manage synchronization and thread execution in a Python program with multiple threads.