Project Report

Operating System CS312

Group Members:

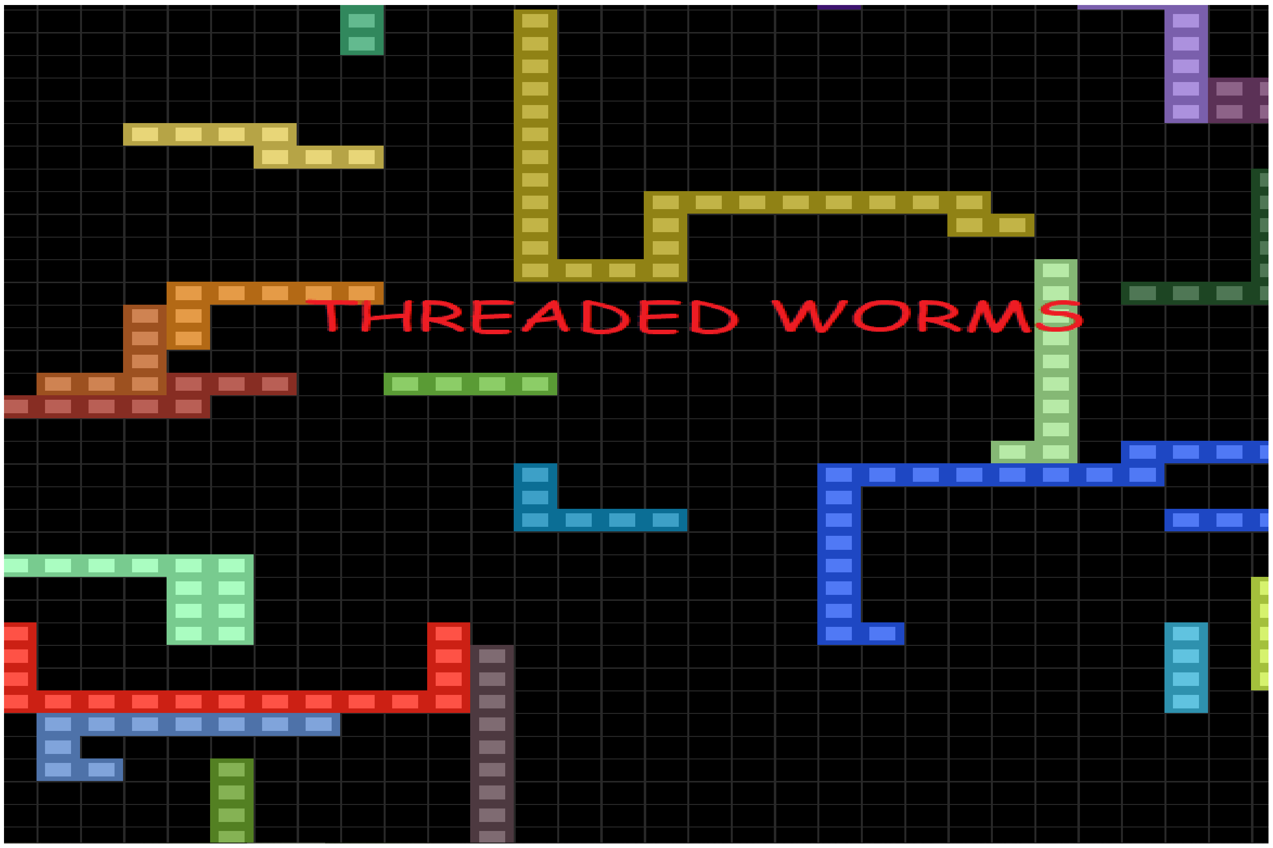
Zain Shahzad 20B-028-CS

Aliza Azam 20B-112-CS

Instructor: Miss Shabina Mushtaque

Topic: Multi-threading, mutex locking simulation

Date: 23,January,2023



Introduction:

To combine topics related to concurrent or parallel computing

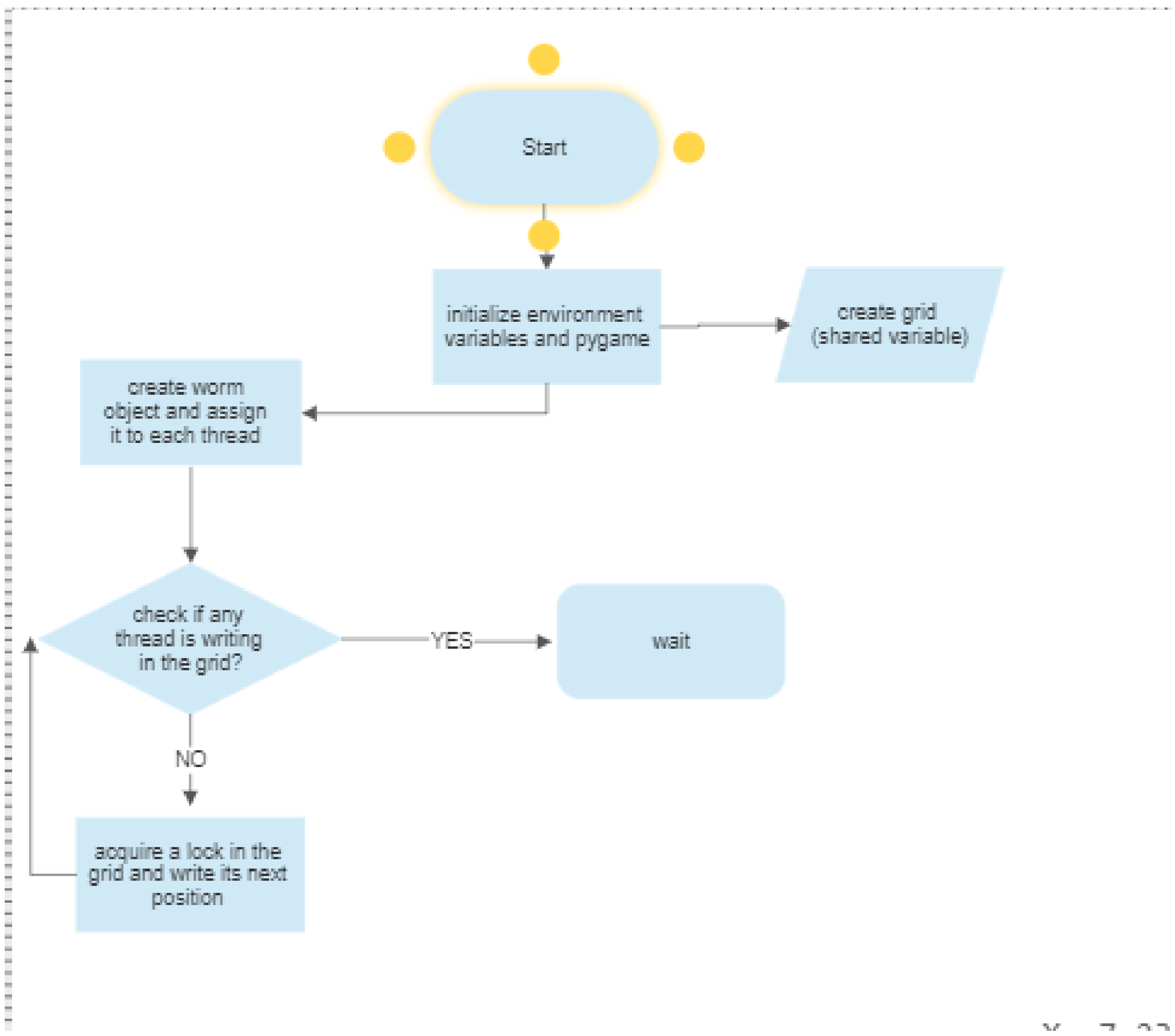
Try to visualize multi-threading in python within the constraints of python GIL

The program demonstrates multithreaded programming and using locks to make the code thread-safe (worms will never go through each other.)

Libraries Used:

- Pygame for simulation and GUI
- Python's multi-threading library for multi-threading and accessing the mutex locks functionalities
- Python 3

Program Flow:


Start
initialize environment variables and pygame
create grid (shared variable)
create worm object and assign it to each thread
check if any thread is writing in the grid?
YES
wait
NO
acquire a lock in the grid and write its next position

- The program will first initialize a pygame window in which there's a tutorial that gives a basic overview of the functionalities.
- A shared data structure GRID is created which will be accessible to every thread within the process
- Next user will define how many threads (worms) you wanted initialize the required threads
- Each thread will manage one worm and have 2 functionalities:

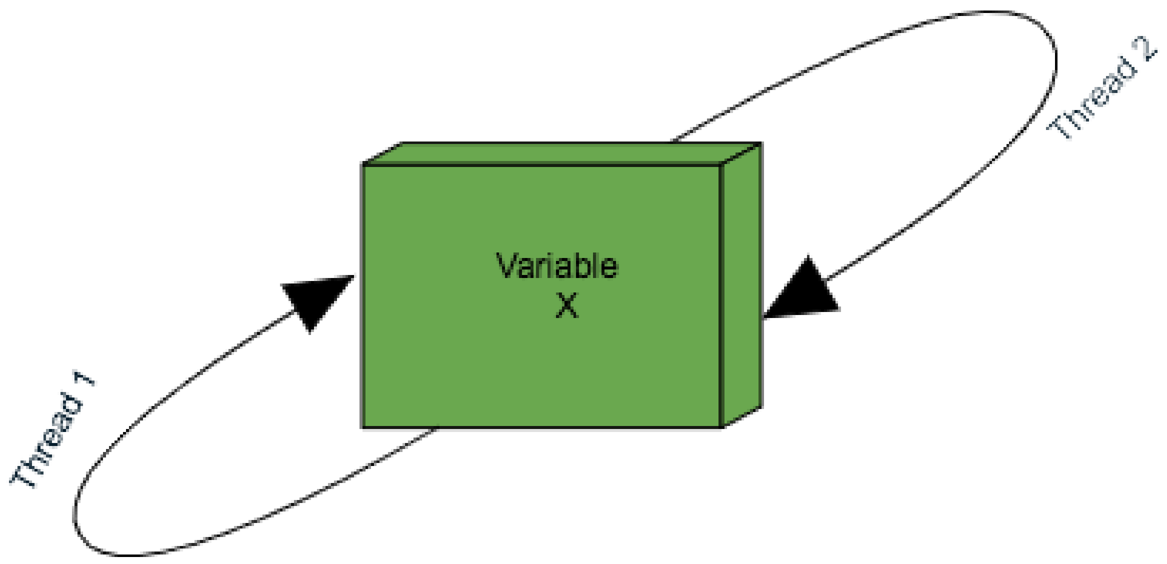
  1) Calculate the next position of the worm within the shared data structure (CPU Bound)

  2) Send an IO request to read write data (IO Bound)

- Each thread will acquire a lock on the data structure read it if there's a worm that has already written on that location, the previous worm will avoid it and move on to the next free position on the shared data structure grid

Shared Grid Data Structure and Locking:

Each thread acquires a lock within the grid and writes certain co-ordinates onto it (its next position) and the lock is then released. The next thread will first read the data within the grid and check if the position it wants to write on is occupied or free? If its not free (occupied location + 1) all of this will be happening after locking the grid to avoid race conditions and make sure our program is thread safe



- Due to the limitation of the Cython compiler the threads created in python aren't fully running in concurrency, python's Global Interpreter Lock disallows threads to truly run in parallel on a CPU. However, we can achieve some degree of parallelism by having multiple threads one for logic other for I/O.

Each worm thread has two behaviors,

- Calculate logic of next position
- Do an IO request fetch and store data to memory

We achieve parallelism when thread 2 is making an IO request to store some data or read the data from the grid data structure, simultaneously thread 1 will be calculating the next logic for where to move next within the grid. The crux is that whenever there's an IO request from a thread we utilize threading and move the IO to a different thread and occupy the CPU with the next thread that needs to calculate logic of movement.

Concurrency in Python

Multiprocessing