**MUTUAL EXCLUSION**

Mutual Exclusion is a concept in concurrent programming to ensure that multiple processes do not have access to the same resources at the same time. It helps prevent race conditions and ensure that data integrity in a distributed system.

For this concept, I will be implementing the ring based algorithm using **Python** to illustrates how this algorithm presents mutual exclusions:

**RING BASED ALGORITHM**

The ring based algorithm takes a form where the process are arranged in a ring form and a token which grants access to enter its critical section is passed from one process to another. Once a process gets the token, it enters it critical section and executes it until it is done. Once it is done, it releases the token and passes it to the next process in the ring.

I will be creating two classes namely:

1. Process class:This class handles the processes making requests and also execution of these processes at their critical sections.
2. Ring Class: This class is represents the ring form in which the process are arranged. The token is passed to the processes and also released when it is done executing its critical section.

The classes are shown below:

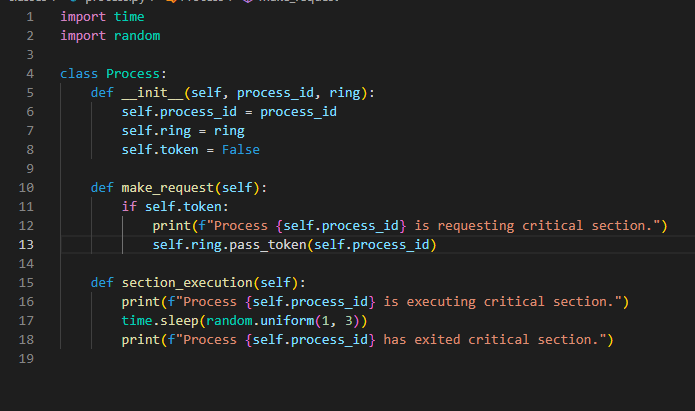


Figure 1: Process class

The code snippet above displays the process class with the process\_id attribute and the ring it belong to. It has two methods which are:

1. **Method to make request:** if the process has the token, it makes request to enter its critical section. It can only make request to enter its critical section if it currently has the token.
2. **Method to execute critical section:** If the pass\_token() method has validated that the process making a request to enter its critical section actually has the token, it permits it to enter its critical section.

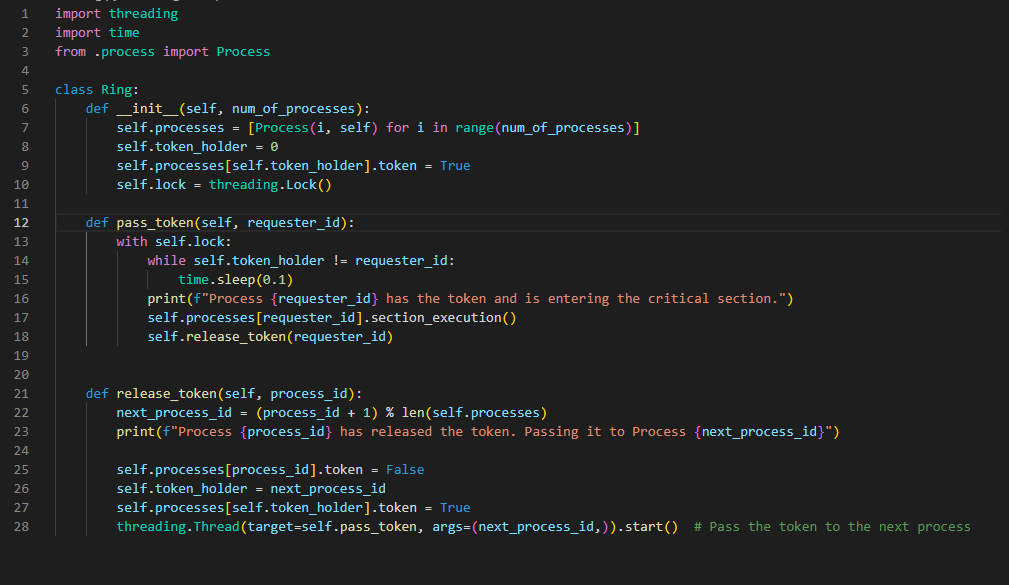


Figure 2: Ring class

The figure above displays the ring class which defines the attribute of the ring which includes the processes, the current process holding the token, a checker to know if a process currently hold the token and a lock to lock a resource once a process enters its critical section.

It also shows the 2 main methods involved in the ring structure:

1. **Method to pass the token:** The method checks if the process requesting to enter its critical section is the one currently with the token. if it is, it sends a message that it is with the token and about to enter its critical section. It then enters its critical section and when it is done, it releases the token to the next process in the ring.
2. **Method to release the token:** When this method is called, the first thing that happens is that it increments the process that just executed its critical section in order for it to move to the next process who needs to enter its critical section.

It then changes the state of the previous process to show that it no longer holds the token, it changes the token holder to the current process and the state to reflect that it now holds the token.

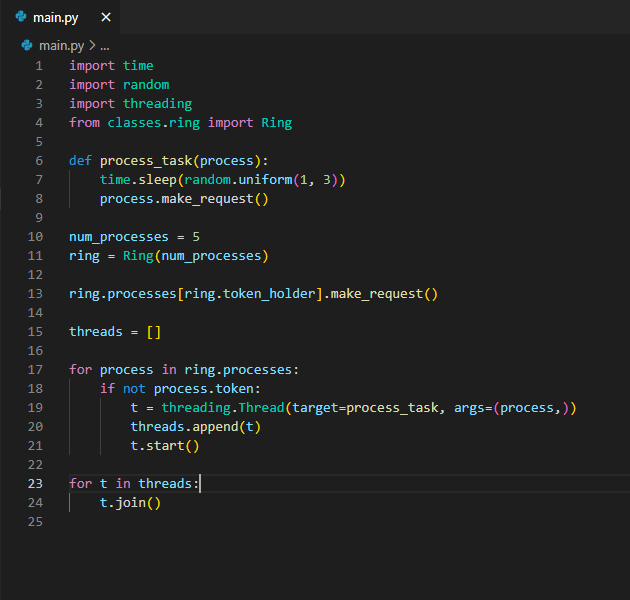


Figure 3: Main.py

The code snippet above illustrates the main entry point that utilizes the classes. It instantiates the number of processes which will be involved in the ring and passes it through the Ring class as shown in line 11.

It then ensures that the first process to make request is the one with currently with the token. The code from line 17 ensures that the function from line 6 which ensure that the process can request to enter its critical section is only executed if it has the token. Then, it starts the thread.

**OUTPUT**

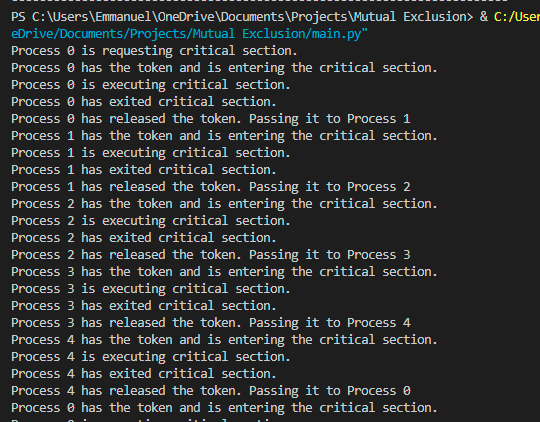


Figure 4: Output of Mutual Exclusion

The output shows mutual exclusion using the ring algorithm. When it has reached the final process, it moves to the next process which is the process it has visited already.

Conditions can be applied to address this:

1. Using a counter: This will involve knowing how many times a process is to enter its critical section. So once it has completed the max number of times it is to enter its critical section, it will no longer be allowed to keep the token. Instead, it will pass it to the next process that needs it.
2. Limiting the number of cycles: We can also set the token to be passed in the cycle at a fixed number of times.

**REASON FOR APPROACH**

1. This approach is quite easy to implement which also gives a simplified messaging protocol to notify the processes when another process is about to enter its critical section.
2. Its also avoid deadlock and starvation. This means that there is no process that waits for a resource indefinitely and no process that doesn’t get to enter its critical section.

The code for this project can be found in the repository below

[mutual exclusion repo](https://github.com/Ajayi-Emmanuel/Mutual_Exclusion_Distributed_System)