Federico Cinelli

CS-510

**Documentation Semaphores to Prevent Deadlocks**

Semaphores are a fundamental tool for preventing deadlocks and managing concurrent access to shared resources in multi-threaded applications. A semaphore acts as a counter that limits the number of threads that can access a particular resource at the same time. Each thread must acquire the semaphore before entering the critical section and release it afterward. This mechanism ensures that no more threads than the specified limit can use the resource simultaneously, which prevents conditions like circular waits, a common cause of deadlocks. By controlling access in this way, semaphores maintain system stability and prevent threads from waiting indefinitely.

The initialization value of a semaphore directly affects how process synchronization occurs. For example, initializing a semaphore with a value of three allows up to three threads to access the shared resource at the same time. If the value is lower, fewer threads can access the resource concurrently, which can increase waiting times but improve control and safety. Conversely, increasing the semaphore value allows more threads to enter simultaneously, which may improve throughput but risks resource contention if too many threads compete for limited resources. Adjusting the semaphore value, therefore, requires balancing concurrency with resource limitations to maintain system performance and prevent potential deadlocks.

Running the Python program is straightforward. Users should save the code file as semaphore\_example.py and run it in a Python-supported IDE, such as VS Code or PyCharm. The program creates multiple threads that attempt to access a shared resource. The console output demonstrates how the semaphore restricts the number of threads in the critical section, showing which threads are waiting, which have entered, and which have released the resource. This structured output provides clear visualization of process synchronization and semaphore functionality.