Harold Molina-Bulla DTSC-UC3M

MULTIPROCESSWITH LOCKS AND SEMAPHORES

SESSION 7

2022

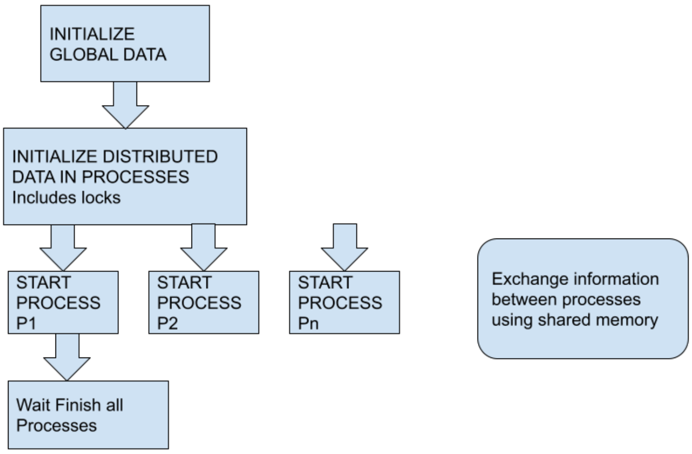
Harold Molina-Bulla DTSC-UC3M

ABOUT MIMD

- MIMD: MULTIPLE INSTRUCTION, MULTIPLE DATA
  - We was working with SIMD (Single Instruction, Single Data)
- The general purpose processors, like our computer's processor, are able to execute multiple programs in parallel: that is Multiple Instruction
- If our general parallel program works with parallel treads needs to be synchronized
- In order to wait until the end of the processes, it is necessary block the execution "joining" to the child processes.

2022

Harold Molina-Bulla DTSC-UC3M



INITIALIZE GLOBAL DATA
INITIALIZE DISTRIBUTED DATA IN PROCESSES Includes locks
START PROCESS P1
START PROCESS P2
START PROCESS Pn
Exchange information between processes using shared memory
Wait Finish all Processes

Harold Molina-Bulla DTSC-UC3M

LOCK

- Locks are used to restrict access to shared memory variables to other processes.
- There are 2 types of locks, depending the influenced variables.
    - Only one variable affected: Invoking the method get_lock()
        - The lock blocks meanwhile execute the inner block
    - All the shared variables in a block: Using an multiprocess object Lock()
        - You get the lock invoking method Lock.aquire() and release the access to the variables invoking the method release()

Harold Molina-Bulla DTSC-UC3M

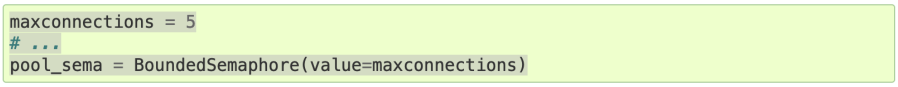
SEMAPHORES

- The semaphore allows execute part of the code if the semaphore is available.
- It is used to restrict the access to limited resources.
- It is a counter, which decrease (by 1) if the semaphore is acquired by a process and increase if the semaphore is released.
- If the semaphore counter is 0, the sub-process can not acquire the semaphore and will wait.

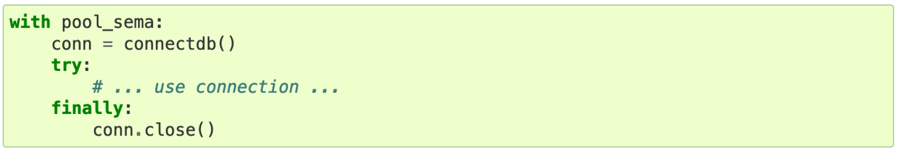
Harold Molina-Bulla DTSC-UC3M

BOUNDEDSEMAPHORE

- BoundedSemaphore is a special heir of Semaphore Object



```
maxconnections = 5
# ...
pool_sema = BoundedSemaphore(value=maxconnections)
```



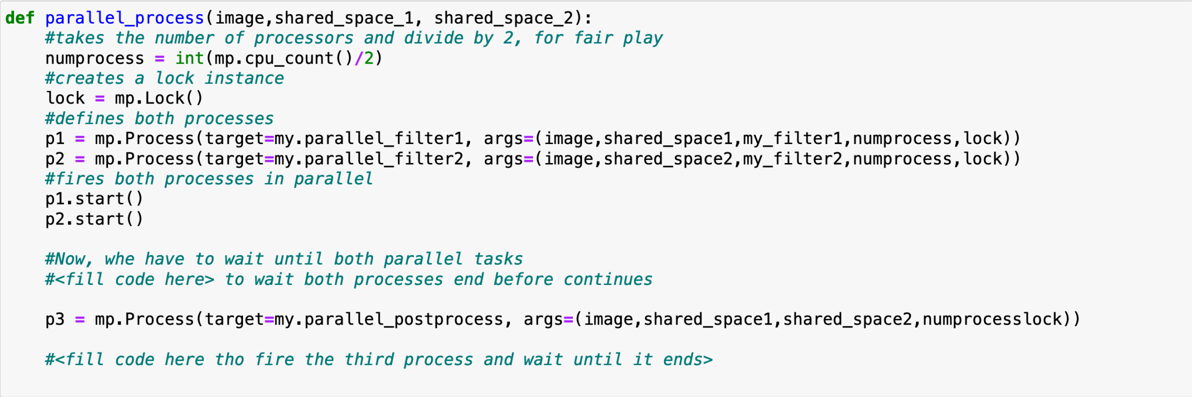
```
with pool_sema:
    conn = connectdb()
    try:
        # ... use connection ...
    finally:
        conn.close()
```

Harold Molina-Bulla DTSC-UC3M

THINKS ABOUT TAKE CARE

- How to distribute resources:
  - Memory, Computational Cores
- How to exchange information
  - Shared Memory Structures (global locks)
- When finish the processes
  - Well limited execution time/cycles
  - Exchanging information between processes
  - Kill processes

Harold Molina-Bulla DTSC-UC3M



```
def parallel_process(image,shared_space_1, shared_space_2):
    #takes the number of processors and divide by 2, for fair play
    numprocess = int(mp.cpu_count()/2)
    #creates a lock instance
    lock = mp.Lock()
    #defines both processes
    p1 = mp.Process(target=my.parallel_filter1, args=(image,shared_space1,my_filter1,numprocess,lock))
    p2 = mp.Process(target=my.parallel_filter2, args=(image,shared_space2,my_filter2,numprocess,lock))
    #fires both processes in parallel
    p1.start()
    p2.start()

    #Now, whe have to wait until both parallel tasks
    #<fill code here> to wait both processes end before continues

    p3 = mp.Process(target=my.parallel_postprocess, args=(image,shared_space1,shared_space2,numprocesslock))

    #<fill code here tho fire the third process and wait until it ends>
```

WINDOWS ISSUES

- Due how MS Windows implements multithreading, and separate memory areas should take care about:
    - If uses nested parallelism, needs to define different modules for different levels.
    - Needs multiple initialization levels.