DISTRIBUTED SYSTEMS

LAB ASSIGNMENT-3

SUBMITTED TO

Professor ARUNKUMAR.T

NAME:SINDUMANI.M

REGNO:19MIC0002

COURSE CODE:CSI3012

SLOT:L15+L16

1.DESIGN AND IMPLEMENT LAMPORT ALGORITHM FOR MUTUAL EXCLUSION

What is LAMPORT ALGORITHM?

Lamport proposed a bakery algorithm, a software solution, for the n process mutual exclusion problem. This algorithm solves a critical problem, following the fairest, first come, first serve principle.

The Lamport algorithm meets all the requirements of the critical section problem:

- **Mutual Exclusion:** We are aware that a process with the lowest number is permitted to enter its critical section when no processes are already running in it. If two processes share the same token number, then what? In that situation, the process with the lowest process ID is chosen as each process's process ID is unique and there will only ever be one process running in a process' crucial section at any given moment. Therefore, the condition of mutual exclusion is satisfied.
- **Progress:** A waiting process examines whether any other waiting processes have a higher priority to enter its critical section after choosing a token. If such a process does not exist, P will immediately start its critical phase. achieving the conditions for progress.
- **Bounded Waiting:** As awaiting, the process will enter its critical section when no other process is in its critical section and
  - If its token number is the smallest among other waiting processes.
  - If token numbers are the same, it has the lowest process ID among other waiting processes

ADVANTAGES:

- Lamport's algorithms are not starved for resources.
- This algorithm uses the FIFO method.
- Atomic registers are how this algorithm operates.
- For the general case of the N process, this algorithm is one of the most straightforward known solutions to the mutual exclusion problem.
- This algorithm makes sure that shared resources are used effectively in a multithreaded setting.

DISADVANTAGES:

Lamport's bakery algorithm is unreliable because it will stop working if any one of the processes fails. It enters and exits the crucial part with 3(N - 1) messages, which is a high message complexity.

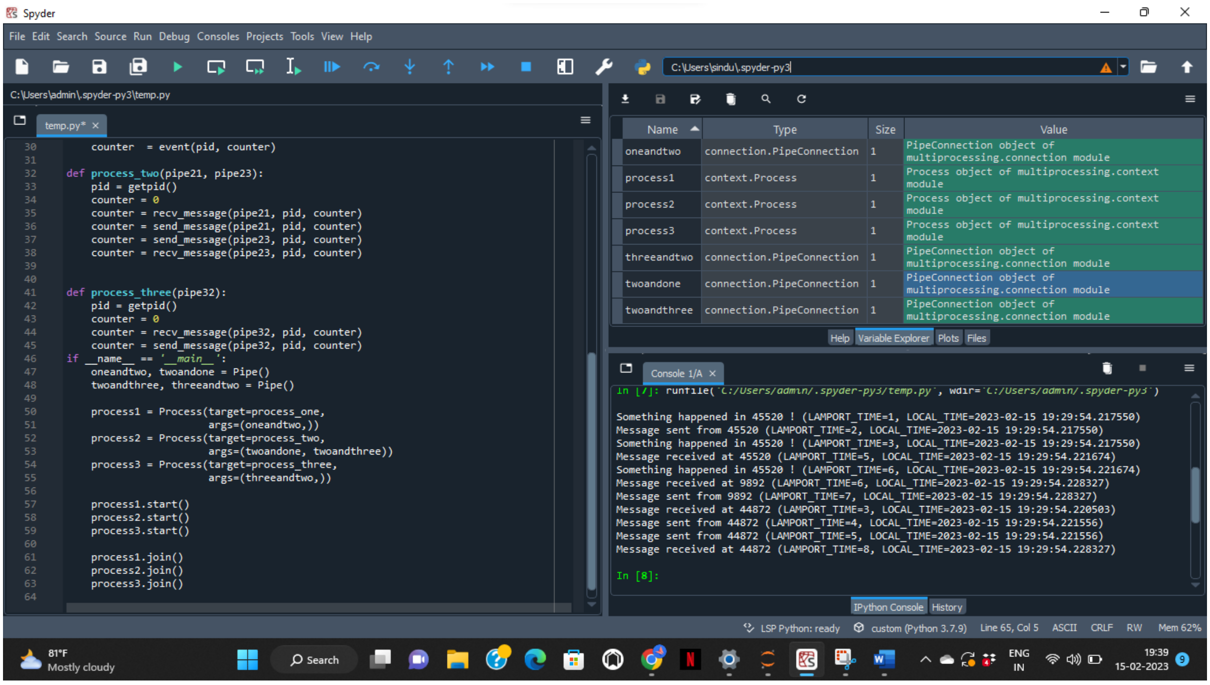
CODE:

```
from multiprocessing import Process, Pipe
from os import getpid
from datetime import datetime
def local_time(counter):
   return ' (LAMPORT_TIME={},
LOCAL_TIME={})'.format(counter,datetime.now())
def calc_recv_timestamp(recv_time_stamp, counter):
   return max(recv_time_stamp, counter) + 1
def event(pid, counter):
   counter += 1
   print('Something happened in {} !'.\
        format(pid) + local_time(counter))
   return counter
def send_message(pipe, pid, counter):
   counter += 1
   pipe.send(('Empty shell', counter))
   print('Message sent from ' + str(pid) + local_time(counter))
   return counter
def recv_message(pipe, pid, counter):
   message, timestamp = pipe.recv()
   counter = calc_recv_timestamp(timestamp, counter)
   print('Message received at ' + str(pid)  + local_time(counter))
   return counter
```

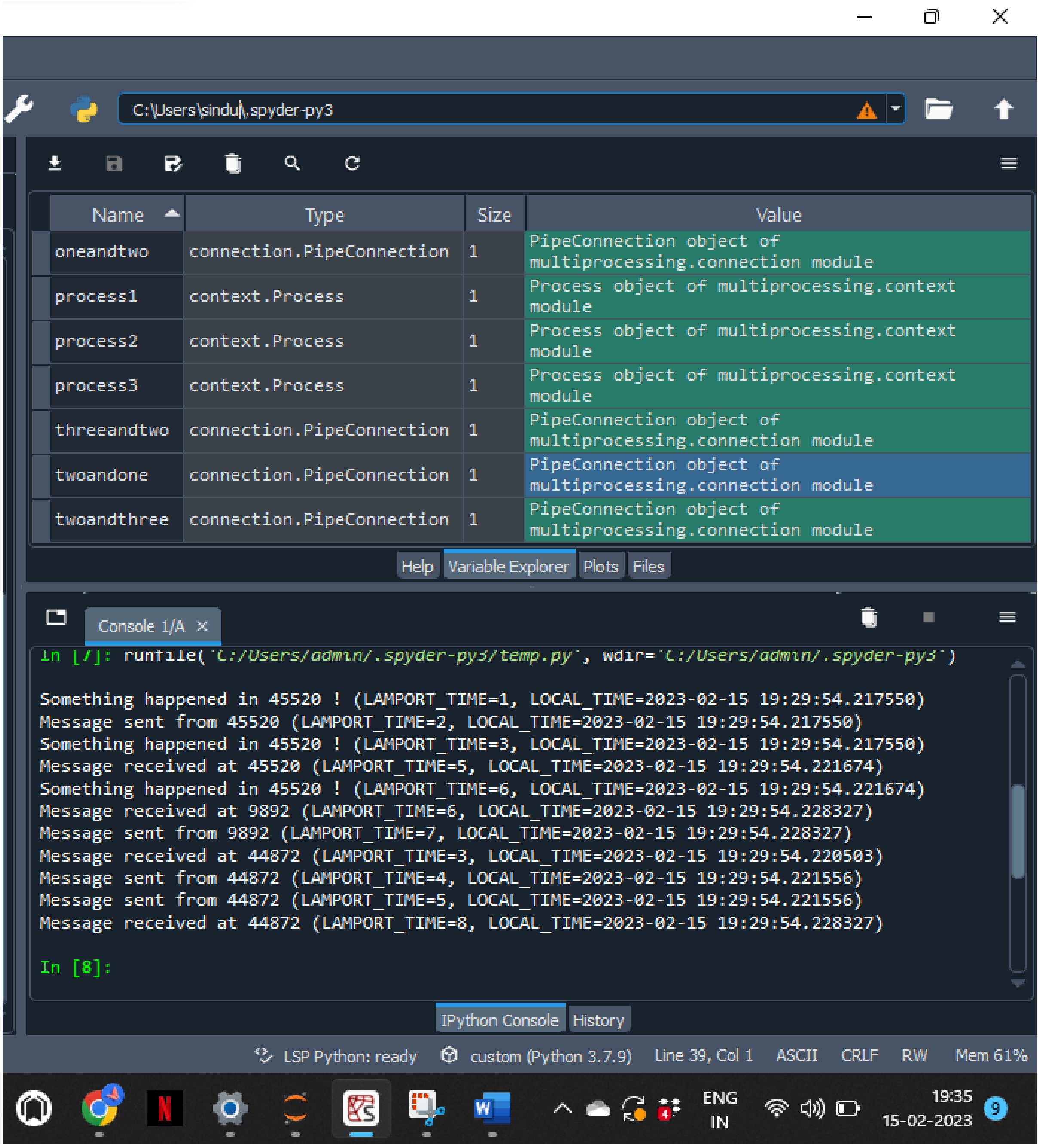
```
def process_one(pipe12):
    pid = getpid()
    counter = 0
    counter = event(pid, counter)
    counter = send_message(pipe12, pid, counter)
    counter  = event(pid, counter)
    counter = recv_message(pipe12, pid, counter)
    counter  = event(pid, counter)

def process_two(pipe21, pipe23):
    pid = getpid()
    counter = 0
    counter = recv_message(pipe21, pid, counter)
    counter = send_message(pipe21, pid, counter)
    counter = send_message(pipe23, pid, counter)
    counter = recv_message(pipe23, pid, counter)



def process_three(pipe32):
    pid = getpid()
    counter = 0
    counter = recv_message(pipe32, pid, counter)
    counter = send_message(pipe32, pid, counter)
if __name__ == '__main__':
    oneandtwo, twoandone = Pipe()
    twoandthree, threeandtwo = Pipe()
```

```
    process1 = Process(target=process_one,
                       args=(oneandtwo,))
    process2 = Process(target=process_two,
                       args=(twoandone, twoandthree))
    process3 = Process(target=process_three,
                       args=(threeandtwo,))

    process1.start()
    process2.start()
    process3.start()

    process1.join()
    process2.join()
    process3.join()
from time import sleep
def process_one(pipe12):
    pid = getpid()
    counter = 0
    counter = event(pid, counter)
    counter = send_message(pipe12, pid, counter)
    sleep(3)
    counter  = event(pid, counter)
    counter = recv_message(pipe12, pid, counter)
    counter  = event(pid, counter)
```

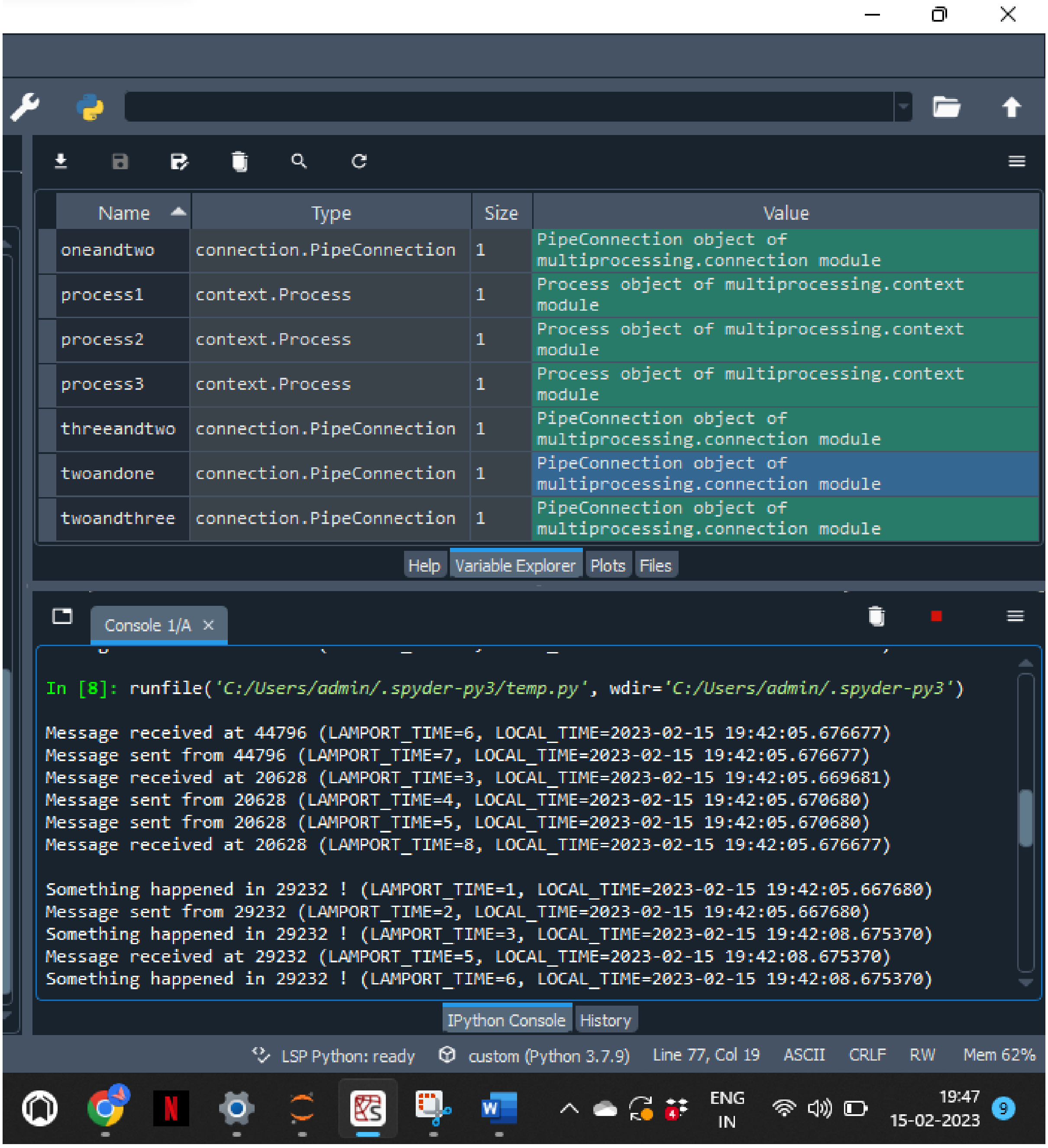
IMPLEMENTATION:



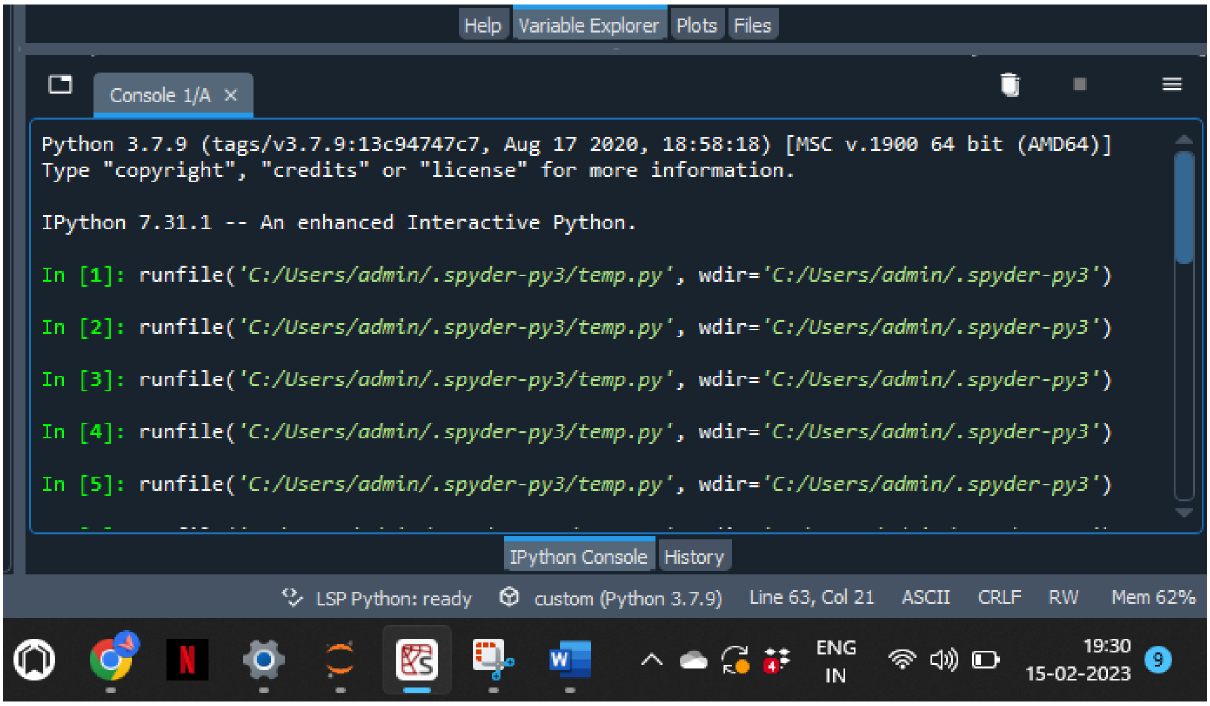
i)The type of events and the updated Lamport timestamp.



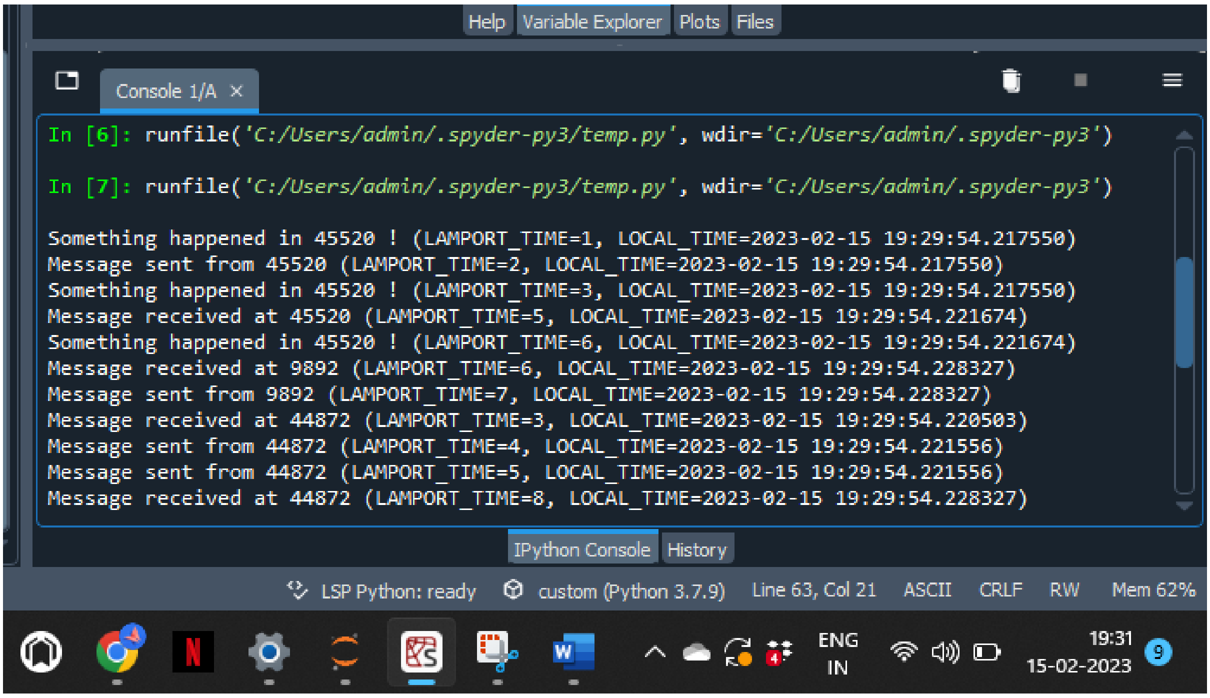
ii) Order of the events has changed, while our timestamps have stayed the same.



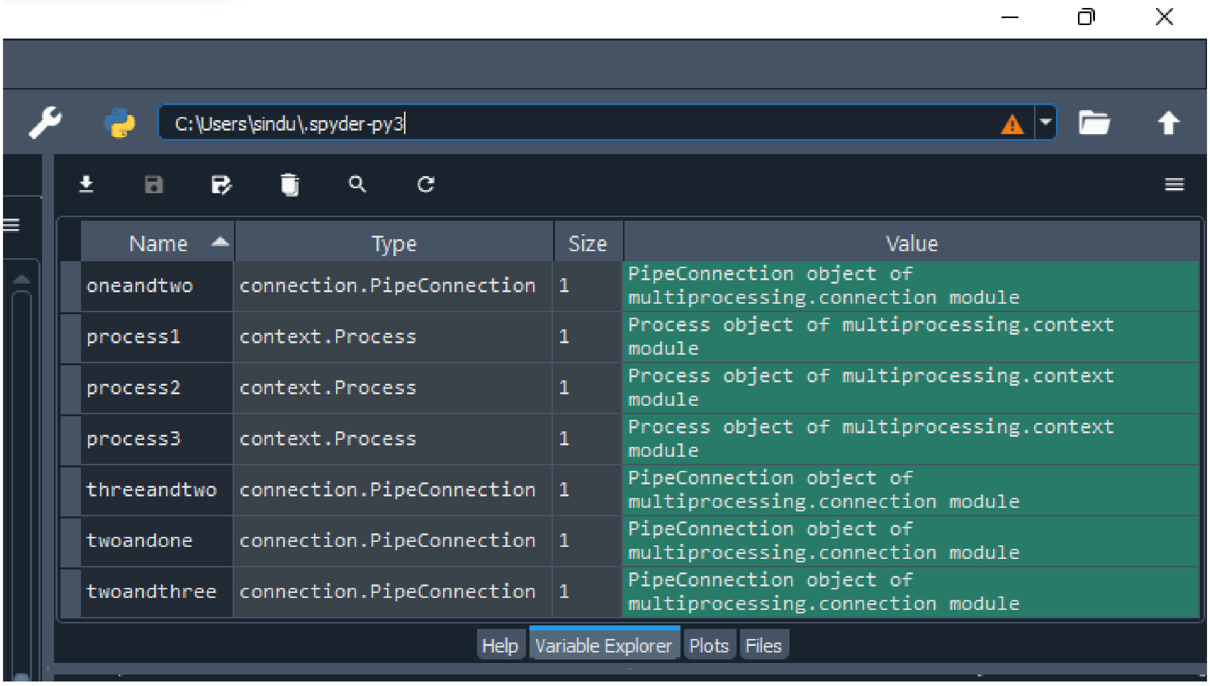
OUTPUT:



i)The type of events and the updated Lamport timestamp.



ii)Execution of process



iii) Order of our events has changed, while our timestamps have stayed the same.