01_HW_05

March 16, 2022

1 Distributed Systems (LTAT.06.007)

1.0.1 Seminar 5: Lamport Clock Algorithm and Vector Clock Algorithm in python

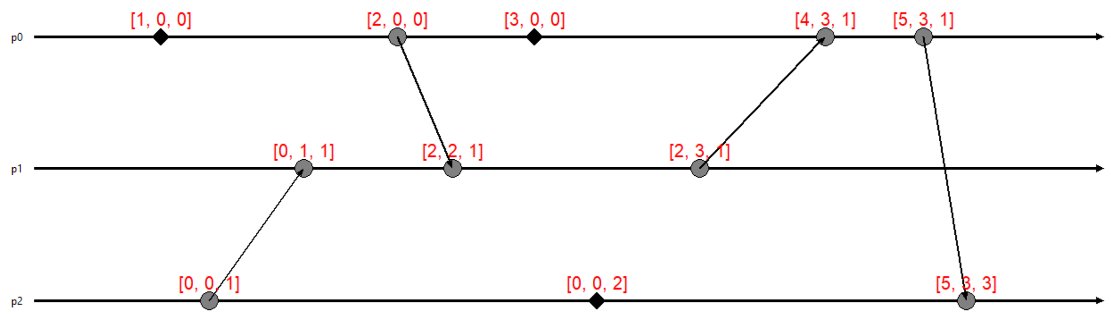
Student : ChengHan Chung

1.1 Task-1: Based on "lamport1.py" in exercise-1, please change the lamport clock algorithm to vector clock algorithm.

1.1 The modified code In function **calculate_timestamps** shown as following below:

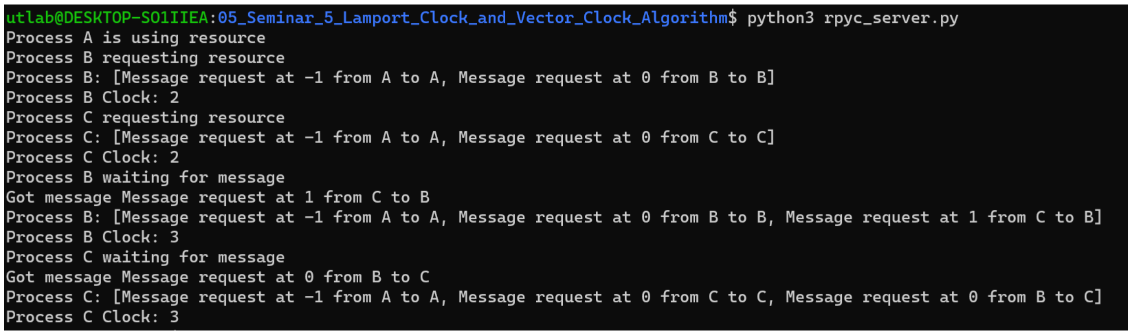
```
[ ]: for event in sorted_events:
         event_type, x, process_number = event[0], event[1], event[2]
         new_vector = []
         if event_type == "L":
             current_vectors[process_number, process_number] += 1
             new_vector = list(current_vectors[process_number])
         elif event_type == "S":
             current_vectors[process_number, process_number] += 1
             messages[event[3]] = current_vectors[process_number,process_number]
             new_vector = list(current_vectors[process_number])
             # Store vectors from sender
             send_vector = list(current_vectors[process_number])
         else:
             current_vectors[process_number, process_number] += 1
             receive_vector=list(current_vectors[process_number])
             # Update receiver
             new_vector = [
                 max(send_vector[0],receive_vector[0]),
                 max(send_vector[1],receive_vector[1]),
                 max(send_vector[2],receive_vector[2])
             ]
             current_vectors[process_number]=new_vector
         timestamps.append((x, process_number, new_vector))
```

1.2 The result shown as following below:

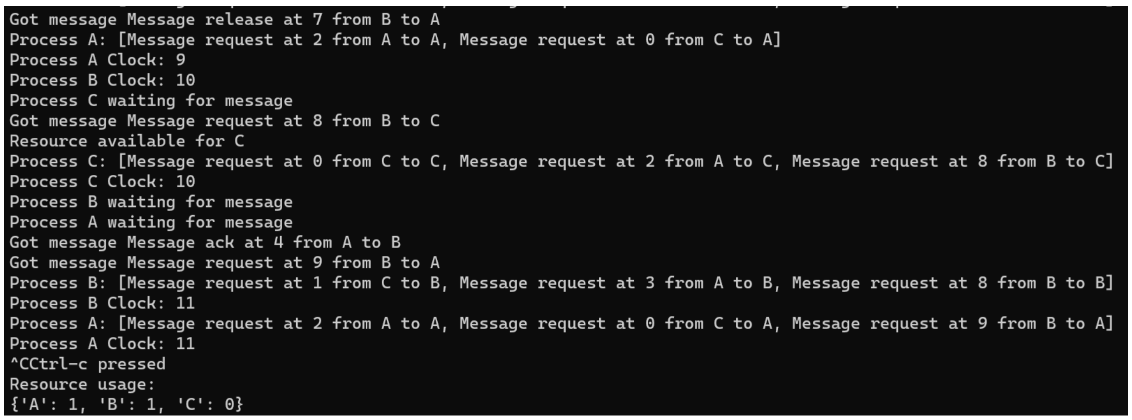


1.2 Task-2: From previous seminars, we already learn how to implement some functions with multi-threads by using RPyC ThreadedServer in python. For this task:

- Integrate the function from `lamport2.py` into the `class` module of RPyC **server communication**
  - Write codes **similar** to below lines into client side to invoke the method from the remote server, therefore the server can create 3 threads for 3 processes. In this way, the class Process could be nested into server class.
    - t1 = Process("A", initially_granted_proc, list(procs - set("A")))
    - t2 = Process("B", initially_granted_proc, list(procs - set("B")))
    - t3 = Process("C", initially_granted_proc, list(procs - set("C")))
- Note that you can make small changes about the source code while you integrate it in the server side. you can also consider exposed implementations.
- The result in server should be similar to this (No print statement is required in the client side).



```
utlab@DESKTOP-SO1IIEA:05_Seminar_5_Lamport_Clock_and_Vector_Clock_Algorithm$ python3 rpyc_server.py
Process A is using resource
Process B requesting resource
Process B: [Message request at -1 from A to A, Message request at 0 from B to B]
Process B Clock: 2
Process C requesting resource
Process C: [Message request at -1 from A to A, Message request at 0 from C to C]
Process C Clock: 2
Process B waiting for message
Got message Message request at 1 from C to B
Process B: [Message request at -1 from A to A, Message request at 0 from B to B, Message request at 1 from C to B]
Process B Clock: 3
Process C waiting for message
Got message Message request at 0 from B to C
Process C: [Message request at -1 from A to A, Message request at 0 from C to C, Message request at 0 from B to C]
Process C Clock: 3
```



```
Got message Message release at 7 from B to A
Process A: [Message request at 2 from A to A, Message request at 0 from C to A]
Process A Clock: 9
Process B Clock: 10
Process C waiting for message
Got message Message request at 8 from B to C
Resource available for C
Process C: [Message request at 0 from C to C, Message request at 2 from A to C, Message request at 8 from B to C]
Process C Clock: 10
Process B waiting for message
Process A waiting for message
Got message Message ack at 4 from A to B
Got message Message request at 9 from B to A
Process B: [Message request at 1 from C to B, Message request at 3 from A to B, Message request at 8 from B to B]
Process B Clock: 11
Process A: [Message request at 2 from A to A, Message request at 0 from C to A, Message request at 9 from B to A]
Process A Clock: 11
^CCtrl-c pressed
Resource usage:
{'A': 1, 'B': 1, 'C': 0}
```