PROGRAM-3

Aim:

Lamport logical clock synchronization between processes with different clocks and update intervals. The processes must exchange messages and correct clocks if required. After each interaction the process must show the clock after updating if required.

Theory:

A Lamport logical clock is an incrementing counter maintained in each process. Conceptually, this logical clock can be thought of as a clock that only has meaning in relation to messages moving between processes. When a process receives a message, it resynchronizes its logical clock with that sender (causality).

Algorithm:

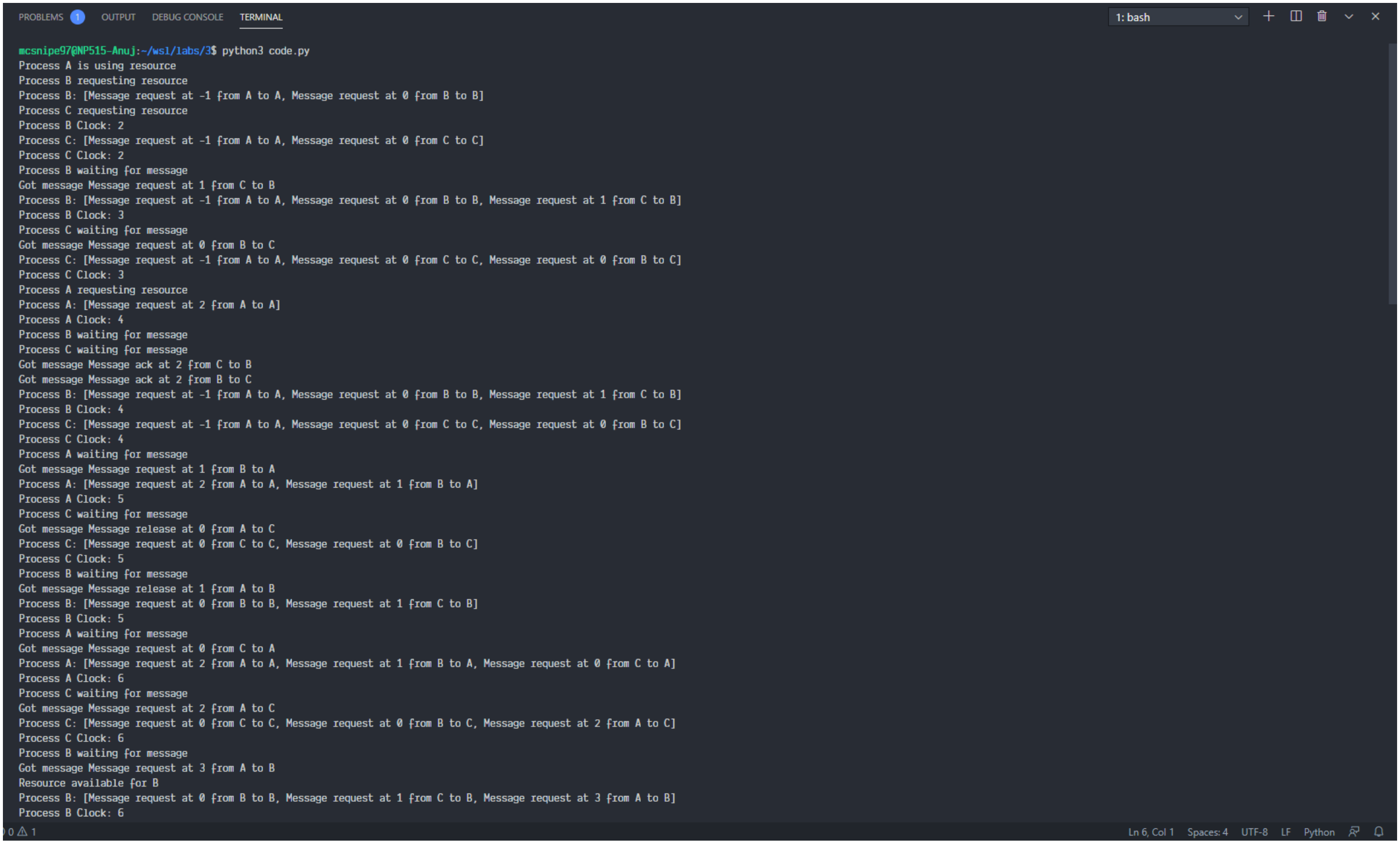
- All the process counters start with value 0.
- A process increments its counter for each event (internal event, message sending, message receiving) in that process.
- When a process sends a message, it includes its (incremented) counter value with the message.
- On receiving a message, the counter of the recipient is updated to the greater of its current counter and the timestamp in the received message, and then incremented by one.

Code:

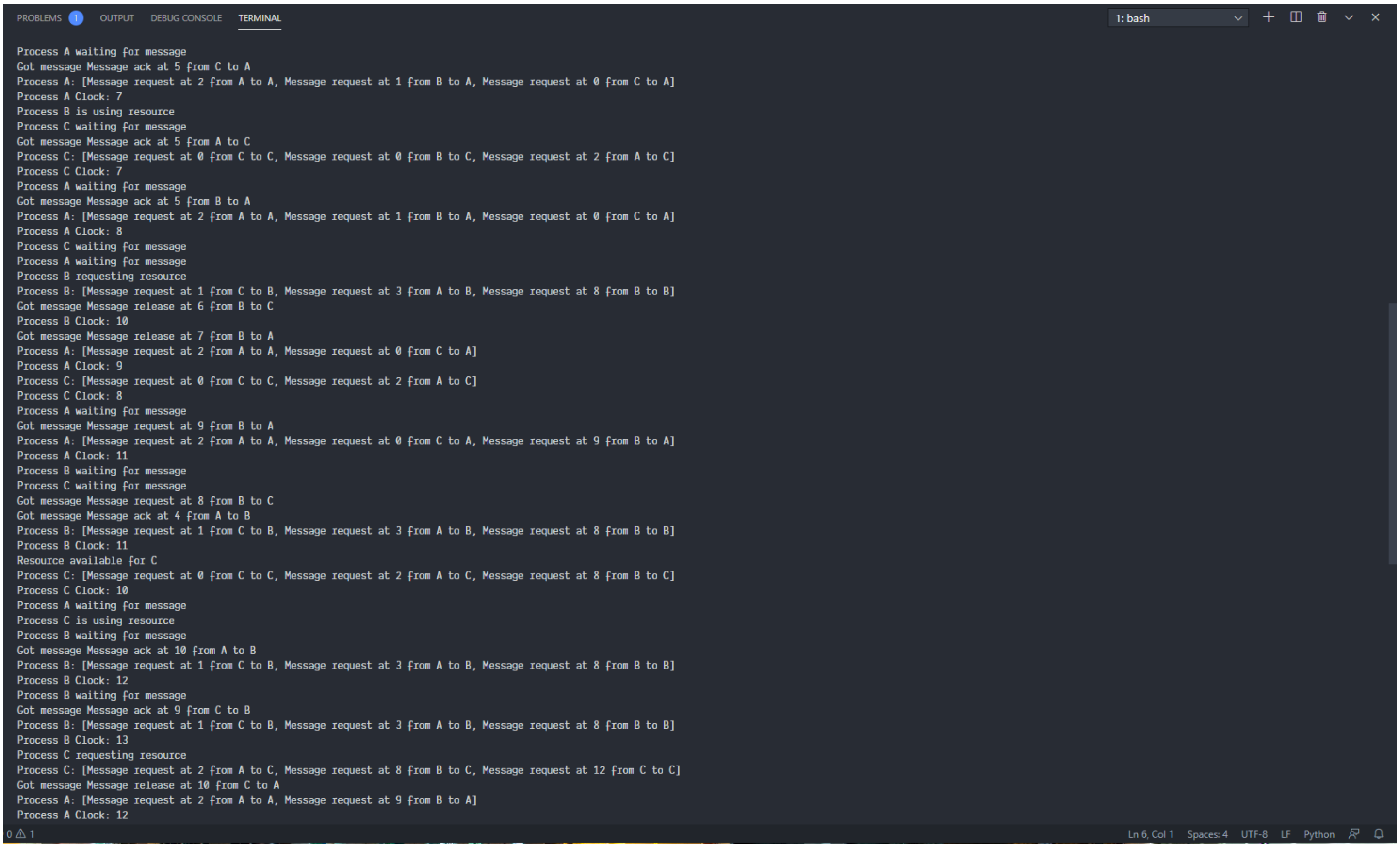
```
import signal
import sys
import time
import threading
from queue import Queue

initially_granted_proc = "A"
procs = {"A", "B", "C"}
resource_usage_counts = {"A": 0, "B": 0, "C": 0}
message_queues = {"A": Queue(), "B": Queue(), "C": Queue()}


class Message(object):
    def __init__(self, msg_type, timestamp, sender, receiver):
        self.msg_type = msg_type
        self.timestamp = timestamp
        self.sender = sender
        self.receiver = receiver

    def __repr__(self):
        return "Message {} at {} from {} to {}".format(
            self.msg_type, self.timestamp,
            self.sender, self.receiver)


class Process(threading.Thread):
    def __init__(self, name, initially_granted, other_processes):
        super(Process, self).__init__()
        self.name = name
        self.has_resource = initially_granted == name
        self.other_processes = other_processes
        self.lamport_clock = 0  # tick after each "event"
        self.request_queue = []
        self.requested = False
        self.request_queue.append(Message("request",
                                          -1, initially_granted,
initially_granted))

    def remove_request(self, msg_type, sender):
        index_of_req = -1
        for i in range(len(self.request_queue)):
            if self.request_queue[i].msg_type == msg_type and \
               self.request_queue[i].sender == sender:
                index_of_req = i
                break
        if i == -1:
            print("Unable to remove")
        else:
            del self.request_queue[i]

    def use_resource(self):
        print("Process {} is using resource".format(self.name))
        resource_usage_counts[self.name] += 1
        time.sleep(2)

    def process_message(self, msg):
        # Based on msg_type handle appropriately
        if msg.msg_type == "request":
            # Put in our request queue and send an ack
            # to the sender
            self.request_queue.append(msg)
            for proc in self.other_processes:
                if proc == msg.sender:
                    message_queues[proc].put(Message(
                        "ack", self.lamport_clock,
                        self.name, msg.sender))
        elif msg.msg_type == "release":
            # Got a release, remove it from our queue
            self.remove_request("request", msg.sender)
        elif msg.msg_type == "ack":
            pass
        else:
            print("Unknown message type")

    def run(self):
        while True:
            if self.has_resource:
                self.use_resource()
```

```python
                self.remove_request("request", self.name)
                # Tell everyone that we are done
                for proc in self.other_processes:
                    message_queues[proc].put(Message(
                        "release", self.lamport_clock,
                        self.name, proc))
                    self.lamport_clock += 1
                self.has_resource, self.requested = False, False
                continue
            # Want to get the resource
            if not self.requested:
                # Request it
                print("Process {} requesting resource".format(
                    self.name))
                self.request_queue.append(Message(
                    "request", self.lamport_clock,
                    self.name, self.name))
                # Broadcast this request
                for proc in self.other_processes:
                    message_queues[proc].put(Message(
                        "request", self.lamport_clock,
                        self.name, proc))
                    self.lamport_clock += 1
                self.requested = True
            else:
                # Just wait until it is available by processing messages
                print("Process {} waiting for message".format(self.name))
                msg = message_queues[self.name].get(block=True)
                # Got a message, check if the timestamp
                # is greater than our clock, if so advance it
                if msg.timestamp >= self.lamport_clock:
                    self.lamport_clock = msg.timestamp + 1
                print("Got message {}".format(msg))
                self.process_message(msg)
                self.lamport_clock += 1
                # Check after processing if the resource is
                # available for me now, if so, grab it.
                # We need earliest request to be ours and check that we
                # have received an older message from everyone else
                if self.check_available():
                    print("Resource available for {}".format(self.name))
                    self.has_resource = True
            print("Process {}: {}".format(self.name, self.request_queue))
            print("Process {} Clock: {}".format(self.name, self.lamport_clock))
            time.sleep(1)

    def check_available(self):
        got_older = {k: False for k in self.other_processes}
        # Get timestamp of our req
        our_req = None
        for req in self.request_queue:
            if req.sender == self.name:
                our_req = req
        if our_req is None:
            return False
        # We found our req make sure it is younger than
        # all the others and we have an older one from
        # the other guys
        for req in self.request_queue:
            if req.sender in got_older and req.timestamp > our_req.timestamp:
                got_older[req.sender] = True
        if all(got_older.values()):
            return True
        return False


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")))

# Daemonizing threads means that if main thread dies, so do they.
# That way the process will exit if the main thread is killed.
t1.setDaemon(True)
t2.setDaemon(True)
t3.setDaemon(True)

try:
    t1.start()
    t2.start()
    t3.start()
    while True:
        # Need some arbitrary timeout here, seems a bit hackish.
        # If we don't do this then the main thread will just block
        # forever waiting for the threads to return and the
        # keyboardinterrupt never gets hit. Interestingly regardless of the
        # timeout, the keyboard interrupt still occurs immediately
        # upon ctrl-c'ing
        t1.join(100)
        t2.join(100)
        t3.join(100)
except KeyboardInterrupt:
    print("Ctrl-c pressed")
    print("Resource usage:")
    print(resource_usage_counts)
    sys.exit(1)
```

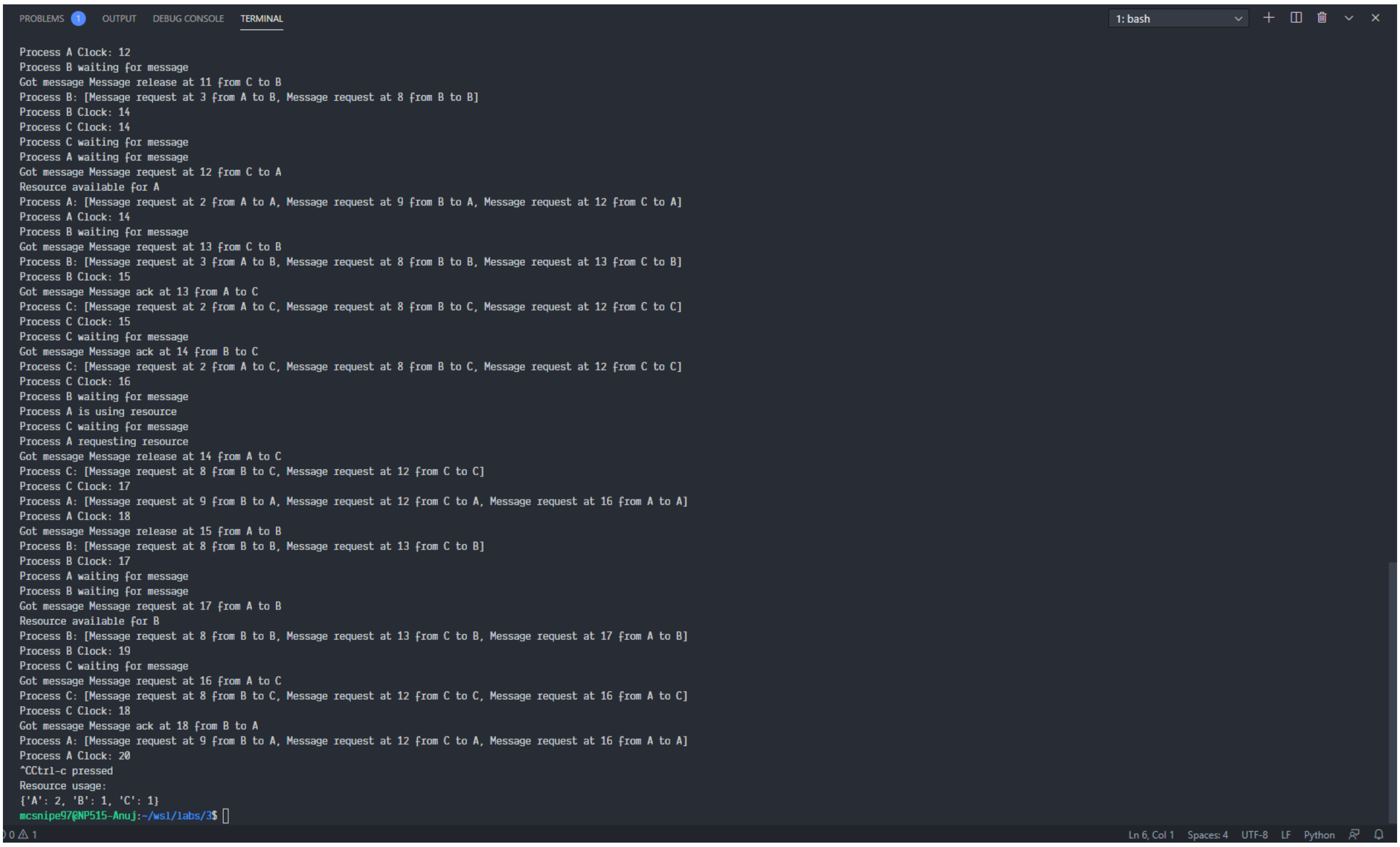
Output:



```
mcsnipe97@NP515-Anuj:~/wsl/labs/3$ python3 code.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 C requesting resource
Process B Clock: 2
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
Process A requesting resource
Process A: [Message request at 2 from A to A]
Process A Clock: 4
Process B waiting for message
Process C waiting for message
Got message Message ack at 2 from C to B
Got message Message ack at 2 from B to C
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: 4
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: 4
Process A waiting for message
Got message Message request at 1 from B to A
Process A: [Message request at 2 from A to A, Message request at 1 from B to A]
Process A Clock: 5
Process C waiting for message
Got message Message release at 0 from A to C
Process C: [Message request at 0 from C to C, Message request at 0 from B to C]
Process C Clock: 5
Process B waiting for message
Got message Message release at 1 from A to B
Process B: [Message request at 0 from B to B, Message request at 1 from C to B]
Process B Clock: 5
Process A waiting for message
Got message Message request at 0 from C to A
Process A: [Message request at 2 from A to A, Message request at 1 from B to A, Message request at 0 from C to A]
Process A Clock: 6
Process C waiting for message
Got message Message request at 2 from A to C
Process C: [Message request at 0 from C to C, Message request at 0 from B to C, Message request at 2 from A to C]
Process C Clock: 6
Process B waiting for message
Got message Message request at 3 from A to B
Resource available for B
Process B: [Message request at 0 from B to B, Message request at 1 from C to B, Message request at 3 from A to B]
Process B Clock: 6
```



```
Process A waiting for message
Got message Message ack at 5 from C to A
Process A: [Message request at 2 from A to A, Message request at 1 from B to A, Message request at 0 from C to A]
Process A Clock: 7
Process B is using resource
Process C waiting for message
Got message Message ack at 5 from A to C
Process C: [Message request at 0 from C to C, Message request at 0 from B to C, Message request at 2 from A to C]
Process C Clock: 7
Process A waiting for message
Got message Message ack at 5 from B to A
Process A: [Message request at 2 from A to A, Message request at 1 from B to A, Message request at 0 from C to A]
Process A Clock: 8
Process C waiting for message
Process A waiting for message
Process B requesting resource
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]
Got message Message release at 6 from B to C
Process B Clock: 10
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 C: [Message request at 0 from C to C, Message request at 2 from A to C]
Process C Clock: 8
Process A waiting for message
Got message Message request at 9 from B to A
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
Process B waiting for message
Process C waiting for message
Got message Message request at 8 from B to C
Got message Message ack at 4 from A to B
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
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 A waiting for message
Process C is using resource
Process B waiting for message
Got message Message ack at 10 from A to B
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: 12
Process B waiting for message
Got message Message ack at 9 from C to B
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: 13
Process C requesting resource
Process C: [Message request at 2 from A to C, Message request at 8 from B to C, Message request at 12 from C to C]
Got message Message release at 10 from C to A
Process A: [Message request at 2 from A to A, Message request at 9 from B to A]
Process A Clock: 12
```



```
Process A Clock: 12
Process B waiting for message
Got message Message release at 11 from C to B
Process B: [Message request at 3 from A to B, Message request at 8 from B to B]
Process B Clock: 14
Process C Clock: 14
Process C waiting for message
Process A waiting for message
Got message Message request at 12 from C to A
Resource available for A
Process A: [Message request at 2 from A to A, Message request at 9 from B to A, Message request at 12 from C to A]
Process A Clock: 14
Process B waiting for message
Got message Message request at 13 from C to B
Process B: [Message request at 3 from A to B, Message request at 8 from B to B, Message request at 13 from C to B]
Process B Clock: 15
Got message Message ack at 13 from A to C
Process C: [Message request at 2 from A to C, Message request at 8 from B to C, Message request at 12 from C to C]
Process C Clock: 15
Process C waiting for message
Got message Message ack at 14 from B to C
Process C: [Message request at 2 from A to C, Message request at 8 from B to C, Message request at 12 from C to C]
Process C Clock: 16
Process B waiting for message
Process A is using resource
Process C waiting for message
Process A requesting resource
Got message Message release at 14 from A to C
Process C: [Message request at 8 from B to C, Message request at 12 from C to C]
Process C Clock: 17
Process A: [Message request at 9 from B to A, Message request at 12 from C to A, Message request at 16 from A to A]
Process A Clock: 18
Got message Message release at 15 from A to B
Process B: [Message request at 8 from B to B, Message request at 13 from C to B]
Process B Clock: 17
Process A waiting for message
Process B waiting for message
Got message Message request at 17 from A to B
Resource available for B
Process B: [Message request at 8 from B to B, Message request at 13 from C to B, Message request at 17 from A to B]
Process B Clock: 19
Process C waiting for message
Got message Message request at 16 from A to C
Process C: [Message request at 8 from B to C, Message request at 12 from C to C, Message request at 16 from A to C]
Process C Clock: 18
Got message Message ack at 18 from B to A
Process A: [Message request at 9 from B to A, Message request at 12 from C to A, Message request at 16 from A to A]
Process A Clock: 20
^CCtrl-c pressed
Resource usage:
{'A': 2, 'B': 1, 'C': 1}
mcsnipe97@NP515-Anuj:~/wsl/labs/3$ 
```

Conclusion:

The algorithm of Lamport timestamps is a simple algorithm used to determine the order of events in a distributed computer system. As different nodes or processes will typically not be perfectly synchronized, this algorithm is used to provide a partial ordering of events with minimal overhead, and conceptually provide a starting point for the more advanced vector clock method. We successfully implemented Lamport Clock.