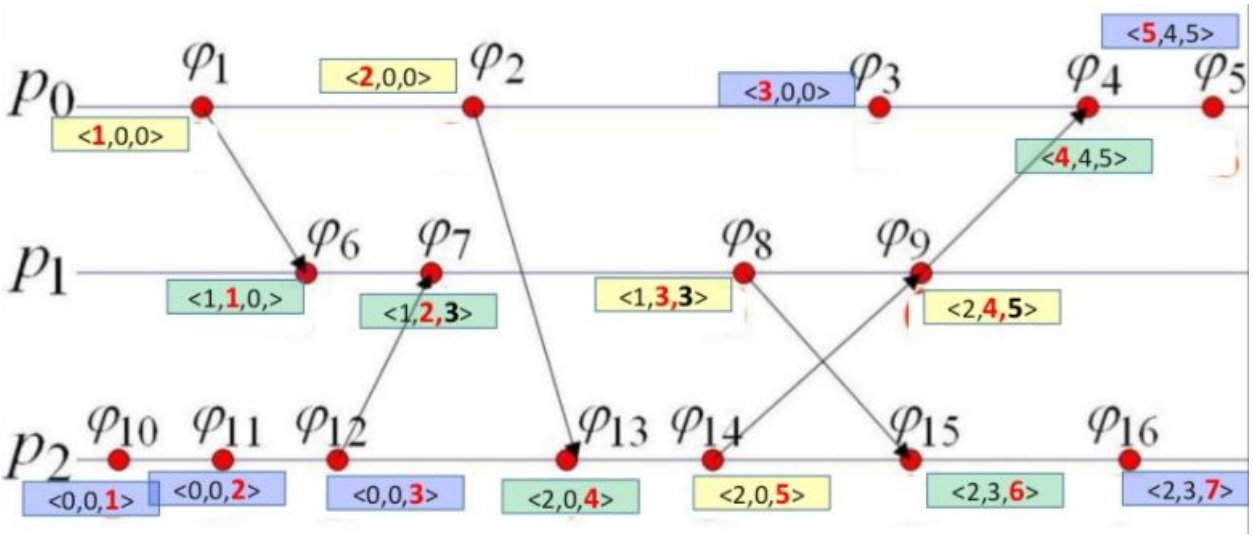


Question 1 Implement the design of a simulation that will Calculate vector clocks [No code]

The following should be present:

1- Interactions

Algo will initialize 3 processors, interacting with the processors and their message(message type). The processor's will have a 1 to 1 relationship with the vector clocks and buffer. Processor will also interact with message retrieving the appropriate processors message. Finally, each message contains a message type enum, which is a 1-1 relationship.

2- What we need to store

We first need to store the events that each process has for execution. We then need to store the vector clock for each process as it get's updated with events within each process. While a send message we need to store and send the message as well as the current state of vector clock of the process with the send event.

4- What needs to change

The state of the vector clock needs to change as each event occurs in a process. The vector clock will change based on what type of an event occurs: computation, send or receive.

On a computation event the ith place on the vector clock of the process where the computation event is happening will increment its clock value by 1: V_i [i]= V_i [i]+1. Example when φ10 in P2 occurred the the vector clock went from <0,0,**0**> to <0,0,**1**> .

On a send event the ith place on the vector clock of the process where the computation event is happening will increment its clock value by 1: V_i [i]= V_i [i]+1 . Then the vector clock state is sent with the message to the receiving event. Example when φ1 sends a message to φ6 the vector clock for P0 is updated from <**0**,0,0> to <**1**,0,0> then <**1**,0,0> is sent along with a message to the recipient φ6 in P2.

On receive event, the recipient of the sent message will update its vector clock based on the vector clock it is sent as well as the present one from it's own process. The ith place of the vector clock for the recipient process will increment by one: V_i [i]= V_i [i]+1. Then the other index in the vector clock will be updated with the max of the current vector clock compared to the one from the message: V_i [j]=max(V_message [j],V_i [j])for j ≠i

5- What will be the input? [The input will be different each time, user should have the ability to control input and get different results]

The input will be a file with the execution plan, which should be read in and executed. Here is an example:

P0:φ1; φ2→φ3; φ4;φ5

6- What will be the output?

Output will be the final vector clock reading when the process thread finishes.

7- Decide which event happened Before, which event is concurrent.

Happen before can be defined as the following:

If a happens before b denoted by (a→b) then either

1. a and b are events in the same process given that a process is sequential a must have happened before b.
2. a and b are connected through a communication event where a is the sender of a message and b is the recipient of the message anywhere in the system.

Another property of the happen before relation is that it is transitive. If a→b and b→c then a→c.

In our example of the three process (P0, P1, P2) we can see happen before relations in the same process like:

φ1→φ2; φ2→φ3; φ4→φ5

φ10→φ11; φ11→φ12

Examples of happen before relations through a communication event are:

φ1→φ6; φ2→φ3; φ12→φ7; φ8→φ15; φ14→φ9; φ9→φ4

Examples of happen before through the transitive property:

φ2→φ13 and φ13→φ14 therefore φ2→φ14
φ14→φ9 and φ9→φ4 therefore φ14→φ4

In our implementation of vector clock calculations, we can compare the vector clocks that are produced to for each event and determine if they have a happen before relation. We will do an element by element comparison of the two vector clocks. If each element of timestamp V_i is less than or equal to the corresponding element of timestamp V_j than this relation is a happen before relation.

Concurrent events denoted as a||b are not sequential events. a and b are not connected either directly or transitively in any shape or form, and have no order between them. When looking at vector timestamps you cannot say that one vector timestamp is less than the other.

Some concurrent events from our example:

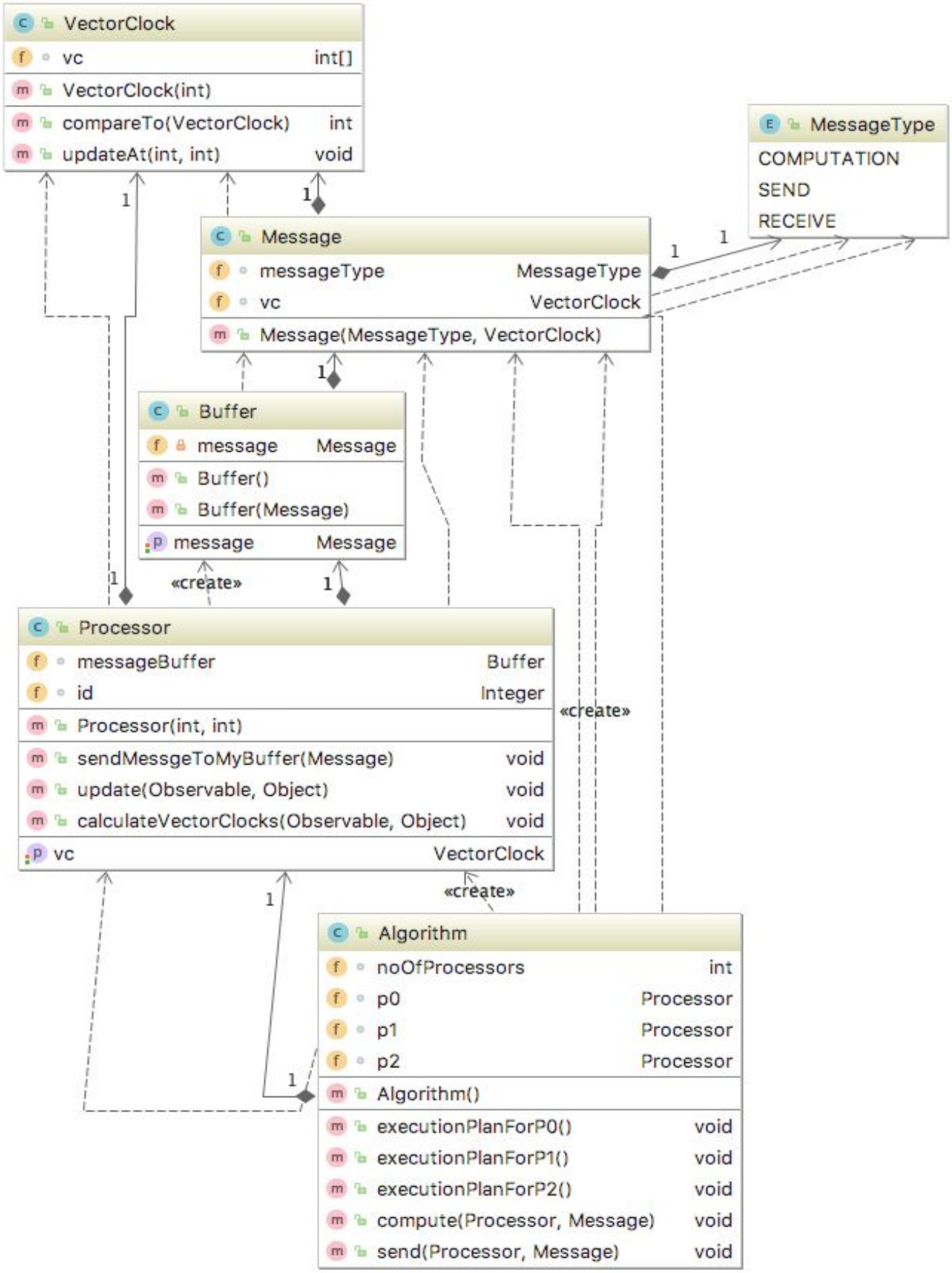
φ1 || φ10 :: (1,0,0) (0,0,1)

φ1 || φ11 :: (1,0,0) (0,0,2)

φ6 || φ12 :: (1,1,0) (0,0,3)

φ2 || φ8 :: (2,0,0) (1,1,3)

In our implementation concurrent events can be found between two given vector by a comparison of each element to its corresponding element in the other vector. If some of the elements in V_i are greater than while others are less than the corresponding element in V_j we can say that the events are concurrent.


VectorClock
vc int[]
VectorClock(int)
compareTo(VectorClock) int
updateAt(int, int) void
MessageType
COMPUTATION
SEND
RECEIVE
Message
messageType MessageType
vc VectorClock
Message(MessageType, VectorClock)
Buffer
message Message
Buffer()
Buffer(Message)
message Message
«create»
Processor
messageBuffer Buffer
id Integer
Processor(int, int)
sendMessgeToMyBuffer(Message) void
update(Observable, Object) void
calculateVectorClocks(Observable, Object) void
vc VectorClock
«create»
«create»
Algorithm
noOfProcessors int
p0 Processor
p1 Processor
p2 Processor
Algorithm()
executionPlanForP0() void
executionPlanForP1() void
executionPlanForP2() void
compute(Processor, Message) void
send(Processor, Message) void
1