Assignment 5

Justin Williams (20365214)
Vincent Coste (20365463)

1.
Using a logical clock says that for events, a and b in a process; a → b which implies that C(a) < C(b) ie. κ_0.
If in the same process, two events occur such that event1 → event2 then the clock for the process, call it $C_i(.)$: $C_i(event1) < C_i(event2)$ ie. κ_1
If for two different processes, an event in P_1 *causes* an event in P_2 then $C_1(event1) < C_2(event2)$ ie κ_2
Now if every processes' event has a timestamp, then from κ_1 and κ_2 the timestamp from event1 in P1 is less than the timestamp for event2, and this is true for every such pair of events. This is the same as $C_1(.) < C_2(.)$ where $C_i(.)$ represents the time of an event in process i. Also it is obvious that event1 → event2. This means that κ_0 is satisfied.

2.
Using vector clocks, every process keeps track of the other interacting processes. By definition, or from κ_0, a → b implies C(a) < C(b). If $V_i[j]$ denotes the number of events that process P_i knows that took place in process P_j, if a takes place in P_i and b takes place in P_j and a → b then VC(a)[j] < VC(b)[j].
Secondly, for VC(a) < VC(b) then for all events, i, VC(a)[i] ≤ VC(b)[i] and there must exist an event j, such that VC(a)[j] < VC(b)[j] which is true if a → b.

3.
The protocol on slide 22 can achieve causally ordered multicast by using logical clocks in the way that it describes. If two messages are sent such that $C(m_1) < C(m_2)$ the protocol says that if process P_i receives message m_1 it is placed in its queue, q_i according to its timestamp and is acknowledged to every other process. This means that every other process also acknowledges the message to process P_i. Since every other process acknowledges m_1 in the same way, every other process will realize that $C(m_1) < C(m_2)$, thus achieving totally ordered multicast.

4.
- Creating the ring:
When several processes need to access the same resource, a logical ring has to be created. To do so, one process P sends a message to all of the other processes, and the ones that will need access to the resource send a message back with their id. P then creates a linked list with itself and the process with the smallest id (or highest priority). It sends to the next process a token with the linked list, this is done for the next process to know it is part of the ring. This is done until the head is reached again.
- Deciding which process has access to the critical region:

A token is passed around the ring between processes. When a process has the token, and if it needs to access the shared resource, it does. When it is done with it, the token is passed along to the next element in the circular linked list (i.e tail points to head). If a process has the token but does not need access to the resources it passes it along to the next process. (there might be a set amount of time for it to keep it).
- Space and time efficiency:
If we have n processes that want to access one resource, the space complexity is O(n) and the time complexity is O(n) as each process has to be traversed to fully create the ring

5.
The number of messages between request and fulfilment is unbounded because a process could keep requesting access without ever getting a majority and becoming the coordinator.
In order to have an upper bound, we have to limit the number of times a process can request to be the coordinator before it gives up.
When a process makes a request for a resource, it needs to send a timestamp along with it. In the case that the request is not granted, it requests for it again but with the timestamp of the first time is was denied. The process keeps requesting it until it gets access. This sets a bound to the number of messages because when a resource gets a request by more than one process, the one with the oldest one should get accepted, in other words we are setting a priority for the process that made the request the earliest.
A lower bound for the number of messages would be 3m with m being the number of messages that have elected the process as coordinator (m has to be a majority of processes, m >= n/2).
A higher bound 3*m*n*k with m the number of processes that elect the process as coordinator, n the total number of processes, and k the number of times an election has to happen.

6.
The problem with the token ring algorithm is that the token is passed along indefinitely when none of the processes need to access the resource. There needs to be a mechanism to determine when the token has gone around the ring once. At that point, the last process that used the resource signals to the other processes that the token is not being passed along anymore. When a process needs access again, it sends a message to the next process in the ring requesting for the token to be sent along once more. This will start the algorithm again, and the process that needs access will eventually get the token.
The lower bound is 1, this happens when the token is still being passed along and the process that needs access is next in line.
The upper bound is n, this happens when the token is not passed through the ring anymore and it needs to notify it to start over the algorithm.

7.
Since we are setting all variable x, y, z to 1 at one point and that there is at least one print statement (or read) at the end, the output of the last two digits should be ‘11’. As a result, 001110 is not a legal output because of the last two digits.