Distributed Computing Problem Set #1

Kai Sun (孙锴)

5110309061

**Problem 1**

The protocol works as follows:

1. We color the process with two kinds of color: blue and red.
2. When sending a message, if the process is blue, then it sends the message with a blue flag; and if the process is red, then it sends the message with a red flag.
3. At first, all processes are colored blue except p0(p0 is red). p0 sends “take a snapshot” to all the other processes.
4. Every blue process records the messages which is sent by itself and the blue messages it receives.
5. When pi receives a red message, if pi is blue, then it takes its snapshot, changes its color to red, sends both the snapshot and the messages it has recorded to p0, and sends “take a snapshot” to the other processes.
6. p0 collects the messages until it has received the messages from all the other processes. Then it has collected all the states: local states are the snapshots; channel states can be calculated (for any channel i -> j, its states = {messages sent by pi} – {messages received by pj}).

Proof of correctness:

Need to prove

We prove it by contradiction. If it does not hold, then there exists i and j s.t and . Let be the time that process i takes its snapshot on process i. Because , the following two claims are true: (1) is earlier than the time is executed (2) all the messages sent by process i after are red. Because , is executed conditioned on process j having received the message that tells has been executed. So before is executed on process j, j must have received a red message from i, which makes process j take its snapshot. But that snapshot does not include which leads to a paradox.

**Problem 2**

1. (3)

2. (1)

3. (2)

4. (4)

5. (1)

6. (2)

**Problem 3**

1.

(a)

If Inp=”ready”

Send “yes” for 11 times.

If “yes” from the other general is delivered, decide “attack”.

If “no” from the other general is delivered, decide “not attack”

If Inp=”not ready”

Send ”no” for 11 times

Decide ”not attack”

It is obvious that the above protocol satisfies Agreement, Validity and Termination.

(b)

If Inp=”ready”

Keep sending “yes”

If “yes” from the other general is delivered, decide “attack”.

If “no” is delivered, decide “not attack”.

If Inp=”not ready”

Keep sending “no”

Decide “not attack”

Validity: It is easy to check the above protocol that If both inputs are “not ready” to attack, then no general decides “attack”. And If both inputs are “ready” and every message sent is delivered, then no general decides “not attack”.

Agreement: We prove by contradiction. WLOG, assume general A decides “attack” and general B decides “not attack”, we can separately check the following 4 cases and all of them lead to a paradox:

(i)A is ready and B is not ready

(ii)A is not ready and B is ready

(iii)A is ready and B is ready

(iv)neither A nor B is ready

Termination:

Our protocol will terminate after everyone delivers one message, so termination is clear.

2.

(a)

If Inp=”ready”

Send “yes” for 11 times.

If “yes” from the other general is delivered, decide “attack”.

If “no” from the other general is delivered, decide “not attack”

Halt

If Inp=”not ready”

Send ”no” for 11 times

Decide ”not attack”

Halt

Same as 1, it is obvious that the above protocol satisfies Agreement, Validity, Termination. And it is also obvious that the above protocol satisfies Halt.

(b)No such algorithm exists.

We prove by contradiction.

Assume we have a solution and n is the smallest number of messages needed.

Let be the n-th message.

The state of the ’s sender does not depend on whether is delivered.

The state of the ’s receiver cannot depend on whether is delivered because whether is delivered may be different in different execution.

So without , both the sender and the receiver can also get their decisions, which implies that we can construct a solution which needs only n-1 messages. But n is the smallest number!

**Problem 4**

X: If a process broadcasts and a process delivers a message before broadcasting a message , no (correct) process delivers unless it has delivered .

Proof:

Causal Order => FIFO Order + X:

By definition of Casual Order, FIFO Order and X, we can see that Casul Order implies and FIFO Order and X.

FIFO Order + X=>Causal Order:

If the broadest of a message causally precedes the broadcast of a message , there must exist a finite sequence of messages such that for any , one of the following happens:

1. and are broadcasted by the same process and is broadcasted before
2. A process delivers before it broadcasts

In the first case, according to FIFO Order, no process delivers before delivering . As we assume that there are no failures, every (correct) process delivers before .

In the second case, according to X, no process delivers before delivering . As we assume that there are no failures, every (correct) process delivers before .

By (1)(2), we know that for every i, every (correct) process delivers before . So every correct process delivers before delivering , which is Causal Order.

**Problem 5**

We can modify the vector clock as follows:

When broadcasting messages:

VC(e)[i]:=VC[i]+1

When delivering messages :

VC(e):=max(VC, TS())

And in the other cases:

VC(e)[i]:=VC[i]

Then for each process i, maintains an array D[1..n] of counters

D[i]=TS()[i] where is the last message delivered from process i;

Process k delivers from as soon as both of the following conditions are satisfied:

D[j]=TS()[j]-1

D[k]>=TS()[k],