Vector clocks

- The timestamp $C_i$ of an event $a$ is a vector of length $n$
  - $C_i[i]$ is $P_i$’s own logical clock
  - $C_i[j]$ is $P_i$’s best guess of logical time at $P_j$’s

- Implementation rules:
  - events $a$ and $b$ are on same process: $C_i[i] = C_i[i] + d$
  - $a$ is the sending and $b$ the receiving of a message $m$: $\forall k, C_j[k] = \max(C_j[k], t_m[k])$
Example

\[ P_1 \]

\[ (1,0,0) \quad e_{11} \quad (2,0,0) \quad e_{12} \quad (3,4,1) \quad e_{13} \]

\[ P_2 \]

\[ (0,1,0) \quad e_{21} \quad (2,2,0) \quad e_{22} \quad (2,3,1) \quad e_{23} \quad (2,4,1) \quad e_{24} \]

\[ P_3 \]

\[ (0,0,1) \quad e_{31} \quad (0,0,2) \quad e_{32} \]
Vector clock: timestamp comparison

• Vector timestamps can be compared in the obvious way:
  
  - \( t^a = t^b \) iff \( \forall i, \ t^a[i] = t^b[i] \)
  
  - \( t^a \neq t^b \) iff \( \exists i, \ t^a[i] \neq t^b[i] \)
  
  - \( t^a \leq t^b \) iff \( \forall i, \ t^a[i] \leq t^b[i] \)
  
  - \( t^a < t^b \) iff \( ( t^a \leq t^b \land t^a \neq t^b ) \)

• Important observation:
  
  - \( \forall i, \forall j : C_i[i] \geq C_j[i] \)
Causally related events

• In a system with vector clocks:
  – $a \rightarrow b$ iff $t^a < t^b$

• Practical consequence: by comparing vector timestamps we can tell if two events are causally related:
  – $t^a < t^b \implies a \rightarrow b$
Proof Outline

• Proof Obligation
  (1) $a \rightarrow b$ implies $t^a < t^b$
  (2) $t^a < t^b$ implies $a \rightarrow b$

Cases to Consider
(a) a and b on the same process
(b) a and b on different processes
What is data race?

• A data race occurs when there are two concurrent accesses to a shared location, and at least one of them is “write”.
  – Based on the timing order, there are three possible races: “write-write race”, “read-write race”, “write-read race”
  – There is no “read-read race”, since two concurrent “read” accesses are harmless.

A data race detection tool needs to find out all the three types of races!
What is a data race? (Cont’d)

• A simplified example

<table>
<thead>
<tr>
<th>Thread 1</th>
<th>Thread 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>X++</td>
<td>Z=0</td>
</tr>
<tr>
<td>Y=33</td>
<td>T=X</td>
</tr>
</tbody>
</table>

Two concurrent accesses:
X++ : write operation        T=X: read operation
An important Property of vc (Cont’d)

if \( t^a < t^b \), then \( a \rightarrow b \)

if \( t^a \) is concurrent with \( t^b \),
then event a and event b happened simultaneously, which indicate a potential race
Data race detection using vector clock

- How to detect data races using vector clock?

simply speaking, if two memory accesses to a shared variable are concurrent, then report them as data race.

Now we explain this idea in more details...
Data race detection by vc (Cont’d)

• Usually, shared variable are supposed to be protected by locks

    Lock()
    Write x
    Y = x + 5
    Unlock()

    Lock()
    Write x
    Unlock()

this is perfectly safe
Open Questions

• How to use Vector Clocks
  – What are messages here?

• How to be efficient about this
  – There can be too many memory accesses
Basic Approach (DJIT+ Algo)

• Each thread $t$ maintains a vector clock $C_t$
  – $C_t(u)$ denotes last operation at thread $u$
• For each lock $m$, maintain a vector clock $L_m$
• When thread $t$ releases lock $m$, update $L_m$ to be $C_t$
• When thread $i$ acquires lock $m$
  – $C_i = \max(C_i, L_m)$ (same as on receiving a message)
Basic Approach (Contd)

• For every variable $x$, maintain $R_x$ and $W_x$ also
• $R_x(t)$ records clock of last read on $x$ by $t$
• $W_x(t)$ records clock of last write on $x$ by $t$
Checking Race Conditions

• We need a happened before relationship between writes from other threads and a new read
  – Thread u reads x
  – Check if Wx is less than Cu

• We also need a happened before relationship between writes from different threads
  – Thread u writes x
  – Check if Wx is less than Cu

• We also need a happened before relationship between previous reads and a new write
  – Thread u writes x
  – Check if Rx is less than Cu