Advanced Network Security
2017

Introduction to Distributed Algorithms
Distributed Algorithms: they are everywhere

- Computer networks
  - message passing
  - routing BGP
  - DNS
- Multi-threaded applications
  - shared memory
  - IPC messages
  - user interfaces
  - OS
  - parallel computing
Simple example

Q: what is the outcome?
what is printed?

i ← 1
j ← ∅

thread i ← 2 end

thread j ← i end

print j

∅ print j

1

i ← 2

1

j ← i

print j

2

i ← 2

1

j ← i

print j

2
A slightly more complex example

\[ i \leftarrow 1 \]
\[ j \leftarrow 0 \]

Q: what is the output?

\[
\begin{align*}
\text{thread} & \quad i \leftarrow 2 \quad ; \quad \text{print } j \\
\text{thread} & \quad j \leftarrow i \quad ; \quad \text{print } j
\end{align*}
\]

These are two separate steps.

- Reading \( i \) happens before the assignment to \( j \).

\( \text{atomic?} \) (indivisible)
What about infinite executions

\[ i < 0 \]

thread while \( i = 0 \) do
print \( i \)
end

thread \( i \gets 1 \)

Q: what is the output?

\[
\begin{array}{cccc}
\text{0} & \text{1} & \text{0} & \text{0} \\
\text{0} & \text{0} & \text{1} & \text{0}
\end{array}
\]

fairness makes that this protocol always terminates.
Scheduling
- modelling construct; assumed to be there
  - determines the next action to be executed
    - non-deterministically
  - executed actions are called events

- fairness

→ a scheduler is fair if an action that is enabled will always be executed eventually.

→ semaphores may not be enabled
→ receipt of a message.
non-determinism vs randomisation

\[
i \leftarrow 0
\]

\[
\text{thread while } i < 0 \text{ do print } i
\]

\[
\text{print } i
\]

\[
i \leftarrow 1
\]

\[
0 \ 1 \ (1)
\]

\[
0 \ 0 \ 1 \ (1)
\]

\[
\vdots
\]

\[
terminates \ within \ n \ steps
\]

\[
1 - \frac{1}{2^n}
\]

\[
\text{while } i = 0
\]

\[
\text{do } i \in \{0, 43\} \text{ ; print } i
\]

\[
1 \ \frac{1}{2}
\]

\[
01 \ \frac{1}{2} \cdot \frac{1}{2}
\]

\[
00 \ 1 \ \frac{1}{2} \cdot \frac{1}{2} \cdot \frac{1}{2} \ldots
\]

\[
000 \ 1
\]

\[
\vdots
\]
Modelling distributed systems

- nodes \( V \)
  - execute a sequence of actions \( \Rightarrow \) events
    "sequential programs"
  - communicate with other nodes
    - sending or receiving messages
    - Through shared memory

- Graph \( G = (V, E) \), nodes \( V \) and edges \( E \)
  - \( |V| = N \)
  - \((v, w) \in E \) if \( v \) can communicate with \( w \)

This can be directed or undirected.

- Shared memory is often directed!
Causality: ordering events → partial order

- Let $A$ be the set of events, and let $a, b \in A$
- Define the 'happened before' relation $a \rightarrow b$ as follows
  - $a$ is executed before $b$ on the same node
  - if $a$ is send event, and $b$ is the corresponding receive
  - if $a$ is a write event, and $b$ is the corresponding read.
- If $a \rightarrow b$ and $b \rightarrow c$ then $a \rightarrow c$

\[
\begin{array}{cccc}
\text{node 1} & \text{send} & \text{node 2} & \text{receive} \\
\hline
a \rightarrow b \rightarrow c \rightarrow d & x \rightarrow f \rightarrow g
\end{array}
\]
  \[\Rightarrow a \rightarrow g\]

- We assume $a \not\rightarrow x$ and $x \rightarrow e$ -- yes!
- If $a \not\rightarrow x$ or $x \not\rightarrow a$
  - then $a$ and $x$ are concurrent, and we write $a \parallel b$
Execution

- An execution leads to set of actions A
- They can be partially ordered using $\rightarrow$
- This partial order $\langle A, \rightarrow \rangle$
can be extended arbitrarily to a total order
  $\langle A, \Rightarrow \rangle$ called the execution
- $\Rightarrow$ has to be consistent with $\rightarrow$