Distributed Systems – Theory
6. Clock synchronization - logical vs. physical clocks

Synchronization: single CPU sys vs. dist sys.

- Single CPU:
  - critical regions, mutual exclusion, and other synchronization probls are solved using methods such as semaphores and monitors.
- DS:
  - semaphores and monitors are not appropriate since they rely on the existence of shared memory
  - Problems to be tackled with:
    - Time
    - Mutual exclusion
    - Election algorithms
    - Atomic transactions
    - Deadllocks

Building algs. for clock synchronization

- Centralized approach:
  - Collect all the information about the system in one place,
  - Then let some process examine it and make a decision as is done in uni-proc. case.
  - Point of failure: the centralizer

- Distributed algs. have the following properties:
  1. The relevant information is scattered among multiple machines.
  2. Processes make decisions based only on local information.
  3. A single point of failure in the system should be avoided.

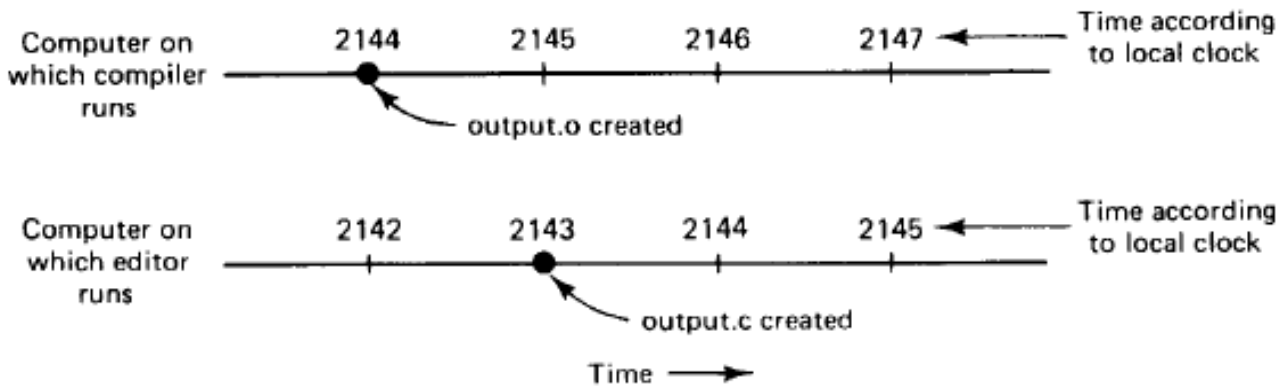
     Why? DS should be more reliable than the individual machines

     If one goes down, the rest should be able to continue to function
  4. No common clock or other precise global time source exists.

DS algs – the 4th property

- Counterex: centralized sys -- In a centralized sys: time is unambiguous.
  - when a process wants to know the time, it makes a system call and the kernel tells it.
  - if process A asks for the time, and later process B asks for the time, the value that B gets > the value A got.
- In a DS, *achieving agreement on time is not trivial*.
- Example: lack of global time knowledge on a cluster of Unix machines

The make example



- Make examines the times at which all the source and object files were last modified
  - Make goes through all the source files to find out which ones need to be recompiled and calls the compiler to recompile them.
- Normal:
  - if the source file input.c has time 2151 and the corresponding object file has time 2150, make knows that input.c has been changed since input.o was created, and thus input.c must be recompiled.
  - if output.c has time 2144 and output.o has time 2145, no compilation is needed here.
- De-synchronization:
  - Short after real time 2144 as above output.c is modified on machine with a slower clock
  - Make will not call the compiler.
  - The resulting executable binary program will then contain a mixture of object files from the old sources and the new sources.

Computer's clock

- Each computer has a circuit for keeping track of time
  - The word "clock" is used to refer to these devices, but they are not actually clocks in the usual sense: timer is perhaps a better word.
- A computer timer is usually a precisely machined quartz crystal.
  - When kept under tension, quartz crystals oscillate at a well-defined frequency that depends on the kind of crystal, how it is cut, and the amount of tension.
- Associated with each crystal are two registers, a counter and a holding register.
  - Each oscillation of the crystal decrements the counter by one.
  - When the counter gets to zero, an interrupt is generated and the counter is reloaded from the holding register.
- In this way, it is possible to program a timer to generate an interrupt 60 times a second, or at any other desired frequency.
  - Each interrupt is called one *clock tick.*

Clock skew

- Although the frequency at which a crystal oscillator runs is usually fairly stable, it is impossible to guarantee that the crystals in different computers all run at exactly the same frequency.
- When a system has n computers,

  all n crystals will run at slightly different rates,

  causing the (software) clocks gradually to get out of sync and give different values when read out.
- This difference in time values is called *clock skew*.
- Consequence:

  programs that are based on the time associated with a file, object, process, or message can fail – e.g. “make” example

Relative times

- "Make" probl. in case of single computer & a single clock is solved:
  - all processes on the machine use the same clock, they will still be internally consistent
  - it does not matter much if this clock is off by a small amount
  - all that really matters are the *relative times*.
- Lamport – 1978
  - showed that clock synchronization is possible and presented an algorithm for achieving it.
  - Ideea: clock synchronization need not be absolute.
    - If two processes do not interact, it is not necessary that their clocks be synchronized because the lack of synchronization would not be observable and thus could not cause problems.
    - what usually matters is not that all processes agree on exactly what time it is, but rather, that they agree on the order in which events occur
      - "make" example: what counts is whether output.c is older or newer than output.o not their absolute creation times.

Logical clocks & physical clocks

- For many appls:
  - it is sufficient that all machines agree on the same time.
  - it is not essential that this time also agree with the real time

  E.g. make example - it is adequate that all machines agree that it is 10:00 even if it is really 10:02.

  Meaning: it is the *internal consistency of the clocks* that matters, not whether they are particularly close to the real time.

  For these algorithms it is conventional to speak of the clocks as

  ***logical clocks*.**
- Counter-ex:

  when the additional constraint is present that the clocks
  - must not only be the same,
  - but also must not deviate from the real time by more than a certain amount,

  the clocks are called ***physical clocks***.

Lamport alg: synchronize logical clocks

Lamport defined a relation called ***happens-before*.**

- The expression $a \rightarrow b$
  - is read "*a* happens before *b*"
  - means that all processes agree that first event *a* occurs, then afterward, event *b* occurs.
- The happens-before relation can be observed directly in two situations:
  1. If *a* and *b* are events in the same process, and *a* occurs before *b* then $a \rightarrow b$ is true.
  2. If *a* is the event of a message being sent by one process, and *b* is the event of the message being received by another process, then $a \rightarrow b$ is also true.
     - meaning: a message cannot be received before it is sent, or even at the same time it is sent, since it takes a finite amount of time to arrive

Properties of happens-before relation

- Transitive
  - if $a \rightarrow b$ and $b \rightarrow c$, then $a \rightarrow c$.

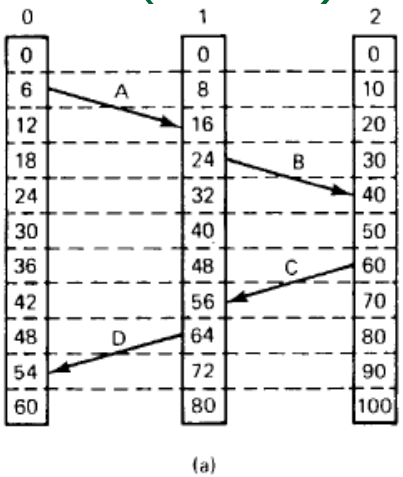
- If two events, *x* and *y*, happen in different processes that do not exchange messages (not even indirectly via third parties),
  - then $x \rightarrow y$ is not true, but neither is $y \rightarrow x$.
  - Events are said to be concurrent

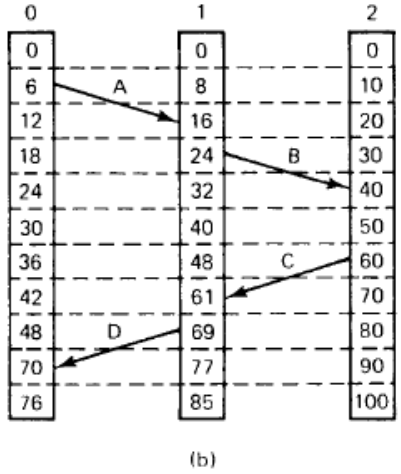
Logic clock

- For every event *a* we assign a time value *C(a)*
  on which all processes agree.
- The property that if $a \rightarrow b$, then *C(a)<C(b)*.
  - Meaning:
    - if *a* and *b* are two events within the same process and *a* occurs before *b*, then *C(a)<C(b)*.
    - if *a* is the sending of a message by one process and *b* is the reception of that message by another process, then *C(a)* and *C(b)* must be assigned in such a way that everyone agrees on the values of *C(a)* and *C(b)* with *C (a) <C(b)*.
- *C* must always go forward (increasing),
  never backward (decreasing).
  - Corrections to time can be made by adding a positive value, never by subtracting one.

Example (1/2)

Non-sync





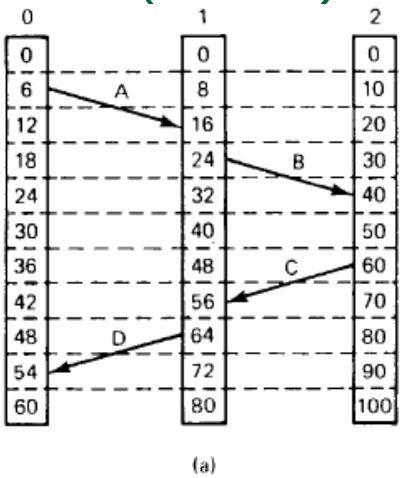
Sync

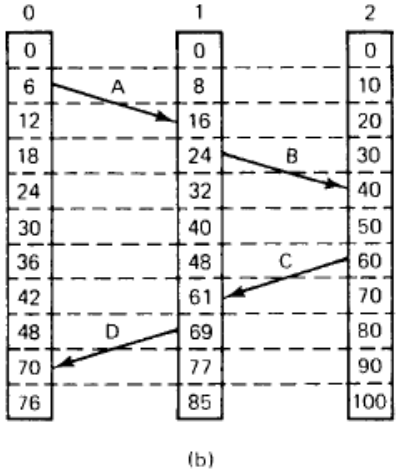
Left:

- three processes that run on different machines, each with its own clock, running at its own speed.
- when the clock has ticked 6 times in process 0, it has ticked 8 times in process 1 and 10 times in process 2.
- each clock runs at a constant rate, but the rates are different due to differences in the crystals.
- time 6: process 0 sends message A to process 1.
- the clock in process 1 reads 16 when it arrives.
- If the message carries the starting time, 6, in it, process 1 will conclude that it took 10 ticks to make the journey.
- according to this reasoning, message B from 1 to 2 takes 16 ticks, again a plausible value.
- message C from 2 to 1 leaves at 60 and arrives at 56. Impossible!
- message D from 1 to leaves at 64 and arrives at 54. Impossible!

Example (2/2)

Non-sync





Sync

Right - Lamport's solution:

- follows directly from the happened-before relation.
- since C left at 60, it must arrive at 61 or later.
- therefore, each message carries the sending time, according to the sender's clock.
- when a message arrives and the receiver's clock shows a value prior to the time the message was sent, the receiver fast forwards its clock to be one more than the sending time.
- on the right we see that C now arrives at 61. Similarly, D arrives at 70.

Adjustments to meet the requirements for global time

- Between every two events, the clock must tick at least once
- Two events cannot occur at exactly the same time.
  - Approach: attach the number of the process in which the event occurs to the low-order end of the time, separated by a decimal point.
    - e.g. events happen in processes 1 and 2, both with time 40, the former becomes 40.1 and the latter becomes 40.2.

-> Assign time to all events in a distributed system subject to the following conditions:

1. If *a* happens before *b* in the same process, $C(a) < C(b)$.
2. If *a* and *b* represent the sending and receiving of a message, $C(a) < C(b)$.
3. For all distinct events *a* and *b*, $C(a)$ is not equal with $C(b)$.

-> A total ordering of all events in the system.

The need for physical clocks

- Lamport's algorithm for logical clock sync
  - gives an unambiguous event ordering,
  - the time values assigned to events are not necessarily close to the actual times at which they occur.
- In some systems like real-time systems, the actual clock time is important!

-> For these systems external physical clocks are required.

- For reasons of efficiency and redundancy,

  multiple physical clocks are generally considered desirable, which yields two problems:

  1. How do we synchronize them with real-world clocks, and
  2. How do we synchronize the clocks with each other?

  [Next week!]

How time is actually measured? 1. The Solar Second

- Since 17th century, time has been measured astronomically
  - The event of the sun's reaching its highest apparent point in the sky is called the transit of the sun – this event occurs at about noon each day.
  - The interval between two consecutive transits of the sun is called the solar day.
  - Since there are 24 hours in a day, each containing 3600 seconds, the *solar second* is defined as exactly 86400th of a solar day.
- The 1940s: the period of the Earth's rotation is not constant!
  - Earth is slowing down due to tidal friction and atmospheric drag.
  - Geologists believe that 300 million years ago there were about 400 days per year in mathematical sense.
  - The length of the year, that is, the time for one trip around the sun, is not thought to have changed; the day has simply become longer.
  - In addition to this long-term trend, short-term variations in the length of the day also occur, probably caused by turbulence deep in the earth's core of molten iron.
  - Astronomers compute the length of the day by measuring a large number of days and taking the average before dividing by 86,400 – the resulting quantity is called the *mean solar second*.

How time is actually measured? 2. TAI

- Invention of the atomic clock in 1948 => measure time much more accurately,
  - independent of the wiggling and wobbling of the earth,
  - counting transitions of the cesium 133 atom.
- physicists: defined the second to be the time it takes the cesium 133 atom to make exactly 1,770 transitions.
  - The choice was made to make the atomic second equal to the mean solar second in the year of its introduction.
  - More than 50 laboratories around the world have cesium 133 clocks.
  - Periodically, each laboratory tells the Bureau International de Heure (BIH) in Paris how many times its clock has ticked.
  - The BIH averages these to produce *International Atomic Time*, which is abbreviated TAI.
  - TAI is just the mean number of ticks of the cesium 133 clocks since midnight on Jan. 1, 1958 (the beginning of time) divided by 1,770.

How time is actually measured? TAI problem

- TAI is highly stable and available to anyone who wants to go to the trouble of buying a cesium clock,
- BUT there is a serious problem with it: 86,400 TAI seconds is NOW about 3 msec less than a mean solar day because the mean solar day is getting longer all the time.
- Approach?
  - Solution of Pope Gregory XIII: in 1582 decreed that 10 days be omitted from the calendar.
    - This event caused riots in the streets because landlords demanded a full month's rent and bankers a full month's interest, while employers refused to pay workers for the 10 days they did not work
    - The Protestant countries, as a matter of principle, refused to have anything to do with papal decrees and did not accept the Gregorian calendar for 170 years.

How time is actually measured? 3. UTC

- BIH solves the problem by introducing *leap seconds* whenever the discrepancy between TAI and solar time grows to 800 msec.
- This correction gives rise to a time system based on constant TAI seconds but which stays in phase with the apparent motion of the sun.
- It is called *Universal Coordinated Time*, abbreviated as UTC.
- UTC is the basis of all modern civil timekeeping.
- It has essentially replaced the old standard, Greenwich Mean Time (GMT), which is astronomical time.

Knowing the UTC

- Most electric power companies base the timing of their 60-Hz or 50-Hz clocks on UTC
- When BIH announces a leap second, the power companies raise their frequency to 61 Hz or 51 Hz for 60 or 50 sec, to advance all the clocks in their distribution area.
- 1 sec is a noticeable interval for a computer & OS that needs to keep accurate time over a period of years must have special software to account for leap seconds as they are announced
- UTC is broadcasted by radio, satellites etc with an accuracy of 0.5msec
  - Need an accurate knowledge of the relative position of the sender and receiver, in order to compensate for the signal propagation delay.