ASYNCHRONOUS EXECUTION WITH SYNCHRONOUS COMMUNICATION

When all the communication between pairs of processes is by using synchronous sendand receive primitives, the resulting order is synchronous order. The algorithms run on asynchronous systems will not work in synchronous system and vice versa is also true**.**

Realizable Synchronous Communication (RSC)

A-execution can be realized under synchronous communication is called a realizable with synchronous communication (RSC).

- An execution can be modeled to give a total order that extends the partial order (E, $\prec$).
- In an A-execution, the messages can be made to appear instantaneous if there exist a linear extension of the execution, such that each send event is immediately followed by its corresponding receive event in this linear extension.

Non-separated linear extension is an extension of (E, $\prec$) is a linear extension of (E, $\prec$) such that for each pair (s, r) $\in$ T, the interval { x $\in$ E s $\prec$ x $\prec$ r } is empty.

A A-execution (E, $\prec$) is an RSC execution if and only if there exists a non-separated linear extension of the partial order (E, $\prec$).

- In the non-separated linear extension, if the adjacent send event and its corresponding receive event are viewed atomically, then that pair of events shares a common past anda common future with each other.

Crown

Let E be an execution. A crown of size k in E is a sequence $<(s^i, r^i), i \in \{0,\ldots, k-1\}>$ of pairs of corresponding send and receive events such that: s0 $\prec$ r1, s1 $\prec$ r2, sk−2 $\prec$ rk−1, sk−1 $\prec$ r0.

The crown is $<(s^1, r^1) (s^2, r^2)>$ as we have s1 $\prec$ r2 and s2 $\prec$ r1. Cyclic dependencies may exist in a crown. The crown criterion states that an A-computation is RSC, i.e., it can be realized on a system with synchronous communication, if and only if it contains no crown.

Timestamp criterion for RSC execution

An execution (E, ≺) is RSC if and only if there exists a mapping from E to T (scalar timestamps) such that

- For any message M, T(S(M)) = T(r(M));
- For each (a, b) in (E x E) \ T, a b T(a) < T(b)

Hierarchy of ordering paradigms

The orders of executions are:

- Synchronous order (SYNC)
- Causal order (CO)
- FIFO order (FIFO)
- Non FIFO order (non-FIFO)
  - For an A-execution, A is RSC if and only if A is an S-execution.
  - RSC ⊂ CO ⊂ FIFO ⊂ A.
  - The above hierarchy implies that some executions belonging to a class X will not belong to any of the classes included in X. The degree of concurrency is most in A and least in SYNC.
  - A program using synchronous communication is easiest to develop and verify.
  - A program using non-FIFO communication, resulting in an A execution, is hardest to design and verify.

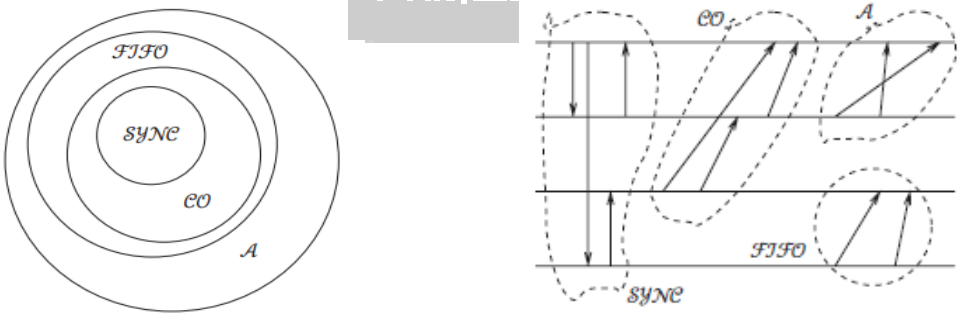

Fig : Hierarchy of execution classes

Simulations

- The events in the RSC execution are scheduled as per some non-separated linear extension, and adjacent (s, r) events in this linear extension are executed sequentially in the synchronous system.
- The partial order of the asynchronous execution remains unchanged.
- If an A-execution is not RSC, then there is no way to schedule the events to make them RSC, without actually altering the partial order of the given A-execution.
- However, the following indirect strategy that does not alter the partial order can beused.
- Each channel $C_{i,j}$ is modeled by a control process $P_{i,j}$ that simulates the channel buffer.
- An asynchronous communication from i to j becomes a synchronous communication from i to $P_{i,j}$ followed by a synchronous communication from $P_{i,j}$ to j.
- This enables the decoupling of the sender from the receiver, a feature that is essential in asynchronous systems.

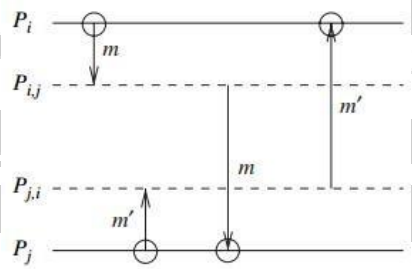

Fig : Modeling channels as processes to simulate an execution using asynchronous primitives on synchronous system

Synchronous programs on asynchronous systems

- A (valid) S-execution can be trivially realized on an asynchronous system by scheduling the messages in the order in which they appear in the S-execution.

- The partial order of the S-execution remains unchanged but the communication occurs on an asynchronous system that uses asynchronous communication primitives.
- Once a message send event is scheduled, the middleware layer waits for acknowledgment; after the ack is received, the synchronous send primitive completes.