Fail-Stop Processes


- **FSP-Properties**
  - Halt-on-Failure Property
    » It will halt before performing an erroneous state transition visible to other proc's.
  - Failure Status Property
    » Any non-faulty process can detect the halting of any other process.
  - Stable Storage Property
    » Part of the processes memory is “stable”, i.e.
      - unaffected by failure
      - readable by other processors

Given FSPs, design a reliable system

- Non-trivial problem! (e.g. Hypercube)
  » needs re-routing (optimal)
  » reconfiguration
  » reallocation

How does one implement a FSP?

- Impossible with finite hardware
- Build a $k$-FSP
- Fails safe for $f \leq k$
Fail-Stop Processes

- Assume stable storage, then the behavior of a FSP is characterized by:
  
  IF  
  k+1 requests AND 
  requests are identical AND 
  requests are from different processes AND 
  NOT failed
  
  THEN 
  process operation
  
  ELSE 
  failed=TRUE

- Stable storage assumption may be quite optimistic.
- Special design-considerations are necessary.

Fail-Stop Processes

- K-FSP are based on two types of real processes

1. \( P(FSP) = \{P_1, P_2, \ldots, P_{k+1}\} \)
   - e.g. usual definition of a processor (CPU)

2. Storage processes \( S(FSP) = \{S_1, P_2, \ldots, S_{2k+1}\} \)
   - memory unit
   - memory management
Fail-Stop Processes

- Block Diagram

\[
\begin{array}{ccc}
P(1) & P(2) & P(k+1) \\
\downarrow & \downarrow & \downarrow \\
Byz. \ safe \ message \ exchange \\
\downarrow & \downarrow & \downarrow \\
S(1) & S(2) & S(2k+1) \\
\downarrow & \downarrow & \downarrow \\
M(1) & M(2) & M(2k+1)
\end{array}
\]

Assumptions

- Network Assumptions
  » Messages are delivered uncorrupted
  » Origin of messages can be authenticated by receiver
- Operating Assumptions
  » P\(_s\) fail independently
  » Failure of P is detected by S-Processes when P-Processes try to write.
  » Disagreement on a write request is confirmed by the S-Processes.
  » Agreement on a request must be reached before executing the write.
  » Only \(M_1, M_2, \ldots, M_{2k+1}\) are visible to outside (of FSP).
Fail-Stop Processes

- Redundant in all P-Processes:
  - P broadcasts write request to all S's
  - S's exchange values+vote (Byzantine safe). P is commander, S's are lieutenants.

- Operation

  IF
  all S agree
  THEN
  write
  ELSE
  stop machine

Fail-Stop Processes

- Stable Storage
  - Majority of copies are correct and identical.
  - A non-faulty FSP can always write to its own stable storage.
  - Any non-faulty process can read any stable storage.
  - Value of a memory location is $maj(M_1, \ldots, M_{2k+1})$
  - An S-proc can write:
    IF exactly 1 request is received from each P
    AND all proc's are identical
    THEN write
    ELSE set a “failed” flag in memory and stop
Fail-Stop Processes

- On the Number of Processors
  - Assume the application needs N processors
    » If we want to tolerate k faults we need \( N + k \) FSPs
    » i.e. \( (N + k) \) k-FSPs
  - Naive implementation
    » to implement 1 FSP
      ■ \( k + 1 \) P-Proc's and \( 2k + 1 \) S-Proc's = \( 3k + 2 \)
    » then to implement the \( N+k \) FSPs
      ■ \( (N + k)(3k + 2) \) that’s a lot of processors!

- It could be considered wasteful to dedicate an entire processor to running an S-Process.
- Therefore assume a single processor is able to run \( s \) S-Processes.

  - Assume P-Proc’s are not delayed by choice of \( s \).
    ⇒ now need only \( \left\lceil (N + k)/s \right\rceil (2k + 1) \) processors for S-Processes.

  - Note: faults not independent anymore.
  - But still \( 2k + 1 \) replication of S-Processes
    ⇒ given k-faults still \( k + 1 \) ⇒ majority!