Fault Tolerance by Multiprocessing

Outline
- Objective
- Condition
- Solution 1 - Repetition
- Solution 2 - TMR Multi-processing
- Solution 3 - Two processes with checker
- Comparison

Objective
- Tolerate faults during the mission time
- Perform FFT in a given time

Given
- One microprocessor (fast enough)
- No error correction circuits
- No hardware TMR
- VxWorks - multi-processing environment

Solution 1 - Repetition
- Repeat the job
  Call FFT function twice or three times

Drawback - Repetition
- Does not catch:
  - Error in the code - bit flip in RAM
  - Long transient error
    Transient error affects until the third FFT calculation
  - Control flow error
    Infinite loop caused by error
    (watch dog timer)
    branch to abnormal location

Objective
time

Given
Fault Tolerance by Multiprocessing

Solution 2 - Multi-processing

- Simulate TMR by multi-processing
- Three processes with three (same or different) copies of code doing the same job

Software TMR

- Catches:
  1. Errors in the code
     - One copy corrupted, but two copies correct.
  2. Transient errors
     - One incorrect result, two correct results.
  3. Control flow error
     - One process functions badly (infinite loop, abnormal jump) but two processes finish their work.

Round Robin Fashion Scheduling

- Assign the same time slot to each process
- Although one process is in infinite loop, other two processes continue and finish their jobs.
- Faulty process is killed after time out.

Voter

- Voter_healthy variable indicates if the voter is working correctly or not.
- Voter checker examines the voter. If there is an error, set voter_healthy = SICK
- If voter_healthy = SICK, call a spare voter.

Voter

```c
int Vote(unsigned *a, unsigned *na, unsigned *b, unsigned *a1, unsigned *a2)
{
    if (voterHealthy == SICK) /* This module is broken, call another */
        return (CRC_Vote1(a, na, b, a1, a2));
    if ((a != a1) || (a != a2)) /* error in pointer */
        return RETRY;
    if (*a == -*na) /* match with backup copy of 2's comp*/
        return OK;
    else {
        if (*a == *b) /* 1st and 3rd match */
            *na = -*a;
        else if (*a == -*b) /* 2nd and 3rd match */
            *a = *b;
        else
            return UNFIXED;
    }
    return FIXED;
}
```

Drawback - Software TMR

- Large memory overhead
  1. Three copies of code
  2. Three copies of data
Solution 3
Two processes with checker

- Two processes with:
  - checker that runs the code with a small data set

- The checker process exercises all the instructions of the code and compares run-time result with pre-calculated result.

Checker

- Exercise all instructions
- Take short time to finish
- Contain a pre-calculated result to compare with run-time result

Example - FFT

1. Two copies of FFT codes (same or different)
2. Two processes on two codes
3. Compare the results
4. If same, done
5. If different,
6. Run two checker processes on two codes
7. Take the one that has no error in the checker
8. Otherwise, try again

FFT - No Error

FFT - Error in the code
Fault Tolerance by Multiprocessing

Checker Example

```c
FftChecker()
{
    float rd[4] = {0.0, 0.5, 1.0, 0.5};
    float id[4] = {0.0, 0.5, 1.0, 0.5};
    float ro[4], io[4];
    int i;
    takeSem(CRC_dataSem);
    CRC_fft(4, 0, rd, id, ro, io);
    for (i = 0; i < 4; i++)
        checkerSig = checkerSig + ro[i] + io[i]; /* checksum calculation */
    semGive(CRC_dataSem);
    exit(0);
}
```

Two Processes with checker

- Tolerate:
  - Error in the code
  - Transient error
  - Long period error
  - Some Permanent errors
  - Infinite loop (by time out & round robin scheduling)
  - Less area overhead than TMR multiprocessing
  - If transient error, takes more time than TMR

Multiprocessing & Design Diversity

- Not two same copies of the code
- Instead, two different copies doing the same job - design diversity
- Example:
  - FFT with original data
  - FFT with negative data (two’s complement)
- Catch permanent errors

Result

<table>
<thead>
<tr>
<th>Scheme</th>
<th>size</th>
<th>area overhead</th>
</tr>
</thead>
<tbody>
<tr>
<td>fft</td>
<td>4084</td>
<td>-</td>
</tr>
<tr>
<td>TMR fft</td>
<td>11632</td>
<td>185%</td>
</tr>
<tr>
<td>fft with checker</td>
<td>9056</td>
<td>122%</td>
</tr>
</tbody>
</table>

Comparison

<table>
<thead>
<tr>
<th></th>
<th>area</th>
<th>error</th>
<th>code</th>
<th>trans.</th>
<th>control</th>
</tr>
</thead>
<tbody>
<tr>
<td>Repetition</td>
<td>1</td>
<td>x</td>
<td>x</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Software TMR</td>
<td>1.85</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td></td>
</tr>
<tr>
<td>Two processes</td>
<td>1.22</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td></td>
</tr>
<tr>
<td>With checker</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Summary

- Use multiprocessing technique to tolerate faults.
  - Three parallel processes
  - Two processes with checkers