1

Chapter-6

6.1. Faults and Fault Tolerance:-

Fault:-

- The appearance of defect during the operation of a software system
- Faults can be accidentally or intentionally introduced into a system.

Failure:-

- When the behaviour of a system deviates from that which is specified for it, this is called a failure.

Failed System:-

- A system that cannot satisfy one or more of the requirements listed in the formal system specification

- Reliability:-

- This is the probability that software system will operate without failure for a specified period of time.

Fault Avoidance:-

- It is a process oriented concept that prevents faults from being introduced into the software
- By carefully designing and manufacturing systems, faults can be avoided.

Fault tolerance:-

- It is (a product orient concept) that deals with the tendency to functions in the presence of hardware or software failures.
- Fault tolerance in the computer system can be achieved through redundancy in hardware, software, information and/or computations.

- In real-time systems, fault tolerance includes design choices that transform hard real-time deadlines into soft ones.
- There are two types of fault tolerant system.
  1) Spatial fault tolerance.
  99) Temporal fault tolerance.

9) Temporal fault tolerance

- It involves techniques that allow for tolerating missed deadlines.
- Of the two, temporal fault-tolerance is more difficult to achieve because it requires careful algorithm design.

6.1.1 II) Spatial fault tolerance:-

- It includes methods involving redundant hardware or software.

- The reliability of most hardware can be increased using redundant hardware.
- Here, two or more pairs of redundant hardware devices provide inputs to the system.
- Each devices compares its output to its companion.

- If the results are unequal, the pair declares itself in error and the outputs are ignored.
- Or, an alternative is a third
- Or a third device is used to determine which of the other two is correct.
- In both cases, it increases the cost, space & power requirements.

- Voting schemes can also be used in software to increase algorithm robustness.
- Here, inputs are processed from more than one source and reduced to some sort of best estimate of the actual value.
  for example, an aircraft's position can be determined via information from satellite, positioning system, inertial navigation data and ground information.

Checkpoints

- It is a way to increase fault-tolerance.
- Here intermediate results are written at fixed locations in code to memorize.
- These locations in a code are called checkpoints.
- Checkpoints can be used during system operation and during system verification.
- If checkpoints are used for testing, then this code is known as test probe.

4.

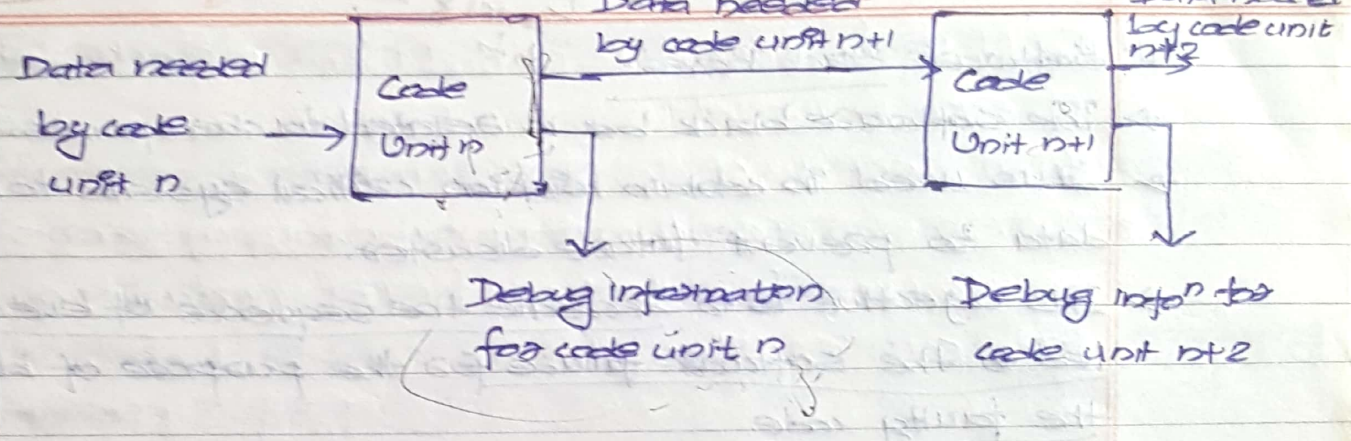

fig: Checkpoint implementation.

Recovery Block Approach:-

- Fault tolerance can be further increased by using checkpoints in conjunction with predetermined reset points in software.
- These reset points mark recovery blocks in the software.
- At the end of each recovery block, the checkpoints are tested for "acceptableness".
- If the results are not acceptable, the processing resumes from the prior recovery block.

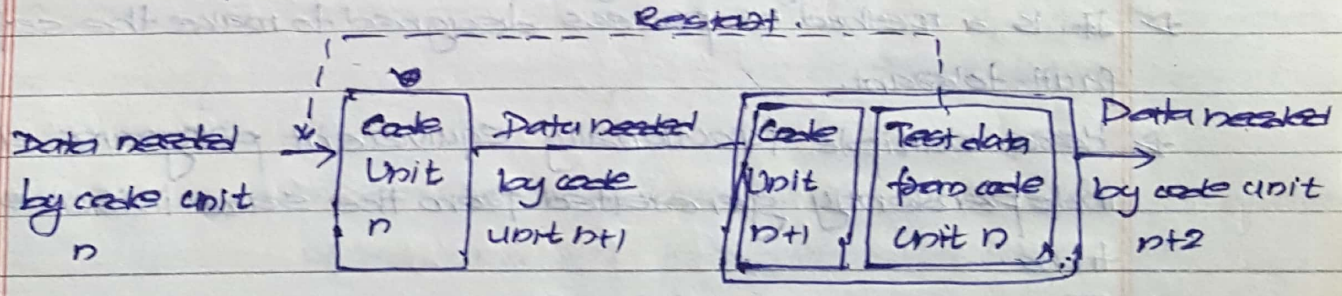

fig: Recovery block approach.

5. plane recorder ⟷ decor

Software Black Boxes:-

- The software black box is related to checkpoints.
- It is used in certain mission critical systems to recover data to prevent future disaster.
- Its objective is to recreate the sequence of events that led to the software failure for the purpose of identifying the faulty code.
- The software black-box recorder is a read checkpoint that records and stores behavioural data during program execution.
- The recovery begins after the system has failed and software black box has been recovered.
- The software black box decoder includes a series of techniques that allow the understanding of the data in the decoder. Then,
- The decoder allows the isolation of likely cause of failure.

Imp N-version Programming:- (Multiversion Programming)

- It is a method or process designed to make the software fault-tolerant.
- Here, multiple functionally equivalent programs are independently generated from the same initial specifications.
- The concept of N version programming was introduced with the conclusion that "Independence of programming efforts will greatly reduce the probability of identical software faults that occurs in two or more versions of the program.

6 Specification

~~The general steps to the Version programming are.~~

- To obtain N-version programming, multiple versions of same module can be obtained by coading that module by different teams of programmers.
- The diversity in same module can be obtained by employing different algorithms for obtaining the same solution or by choosing different programming languages.

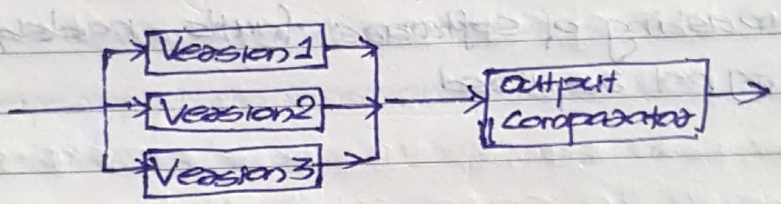

fig: N-version programming

The general steps of N-version programming are:-

1. An initial specification of the intended functionality is developed.
2. From the specifications, two or more versions of the program are independently developed, each by a group that doesn't interact with others
3. The implementations of these functionally equivalent programs use different algorithms and programming languages.
4. At various points of the program, special mechanisms are built into the software which allows the program to be governed by the N-version execution environment.
5. The special mechanisms include: comparison vectors, comparison status indicators and synchronization mechanisms.
6. The resulting programs are called N-version software.

7.

7. The N-version execution environment checks output of each individual N-version programs and makes the final decision of the N-version programs.
8. The decision algorithm may vary ranging from simple as accepting from the most frequently occuring output to more complex algorithm.

Advantages of NVP

- Immediate masking of software faults - no delay in operation.
- Self-checking not required

Disadvantages:

- Extremely dependent on input specifications.

(ચાલુતાવેળાં H/W માટે Software ને Diagonise
8. , ઇરીલાને બતાવે ગરેટ H/W આવેલો fault શોધાવડનુ

Built-In-Test Software:-

- It enhance fault-tolerance by providing ongoing ~~diagn~~ diagnostics of the underlying hardware for processing by the software.
- Built-In-Test Software is extremely important in ongoing systems.
  for eg; If an I/O channel is not functioning correctly, the software may be capable to shut off the channel a.
- BITS causes high response time. So, while selecting BITS its ~~resp~~ effect on response time & CPU utilization should be considered.

6.2 Systems Integration:-

- Integration is the process of combining partial functionality to form the overall functionality.
- Real-time systems are usually embedded and the integration process involves both software units & hardware units.
- Each of these units has been developed by different teams or individuals within the project organization.
- Although it is assumed that each units have been tested and verified separately, the overall behaviour of the system cannot be tested until the system is wholly integrated.
- Software integration can be complicated if ~~the~~ both h/w and s/w are new.

9.

6.2.1 Goals of System Integration:-

- The software integration activity has the most uncertain schedule and is typically the cause of project cost overruns.
- The specification, design, implementation and testing practices used throughout the software project lifecycle determine the failure or success of the system integration phase.
- Hence, by the time of software integration, it may be very difficult to fix problems.
- So, many programming practices were developed to ensure fewest errors in the source code at the arrival of this stage.
- Such programming practices includes; extreme Programming light weight methodologies & so on.

6.2.2 System Unification:-

- Fitting the pieces of the system together form its individual components is a tricky business.
- Parameter mismatching, variable name mistyping and calling sequence errors are some of the problems encountered during system integration.

→ The system unification is the process of linking together the tested software modules down in the ordinary fashion form the source code library

- During the linking process, the errors such as memory

assignment violations, page link errors, etc occurs...

- So, these problems should be resolved. Once resolved, the loadable module/code can be downloaded to the target machine.
- Once the load module has been created and loaded into the target machine, testing of timing and hardware/software interaction begin.

6.2.3 System Verification:-

- The system testing for the embedded system is a tedious process and often requires days or weeks.
- During system testing, a test log must be maintained as shown below:

Test number	Reference No.	Test Name	Pass/Fail	Date	Tested
S121	3.2.2.2	XYZ	Pass	5/16/2016	PL

fig: A sample test log.

- If a system test fails, Then it is corrected by identifying the problem.
- During the correction some module may have been changed. So, regression testing should be done for all the module that have been changed during correction.
- The regression testing should also be done on the system to ensure that no side effect has been introduced during error repair.

6.2.9. System Integration Tools.

Imp

MOLLSH

1. Multimeter
2. Oscilloscope
3. Logic Analyzer
4. In-circuit Emulator
5. Software Simulator
6. Hardware prototype.

1. Multimeter:-
- It is important tool in embedded system where the software reads & controls analog values through hardware.
- It measures voltage, current or power,
- It is used to validate analog input to or output from the system.

2. Oscilloscope. Validate interrupt & Clock
- It is like multimeter which can be used for validating interrupt integrity, monitoring clock, etc.

— software debugge

3. Logic Analyzer. - analyze each instruction code
- It is important tool in embedded system for debugging software.
- It is use to capture data or events, measure individual instruction times and so on.

during

- set program counter
- module level debugging
- software patching
- read/write data from/to memory

4. In-Circuit Emulator (ICE)

- It is important tool in embedded system as it provide the ability to set the program counter, insert into and read from memory, and so on during module level debugging and system integration.
- It uses special hardware in conjunction with software to emulate the target CPU, for providing the above mentioned features.
- It is useful in software patching and for single-stepping through critical portions of code.
- It isn't useful in timing test because timing changes can be introduced by the emulator.

5. Software Simulators:- simulate the integration process to find defects which will help in debugging

- When integrating and debugging embedded systems, software simulators are often needed to stand in for hardware or inputs that do not exist or that are not available.

6. Hardware Prototypes:-

- In the absence of the actual hardware system, simulation hardware may be preferable.
- These devices are required when it would be impossible to test the software in actual hardware, or when the software is ready before the hardware.

classmate
Date
Page

Patching

- The process of correcting the code directly on the target machine is called patching
- It allows minor errors detected during the integration process to be corrected directly on the target machine, without undergoing the tedious process of correcting the source code and creating a new load module.
- It is also useful in repairing software remotely, for example: space-borne applications.

- Patching requires mainly 2 things.
  - expert command of the opcodes for the target machine
  - accurate memory map which includes the contents of each address in memory and a method for inserting directly into memory.

- There are 2 types of patch.
  - In-line patch
  - Oversized patch.

1. In-line patch

- If the patch ~~requires~~ fits to the memory space that is currently occupied by the code to be replaced. Then it is called in-line patch.

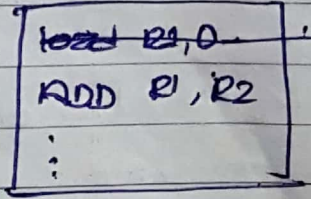

fig: Inline patch

for eg: If 1 was supposed to be added to register 1 (R1) instead of a '0'. This error can be changed easily by altering the memory location containing the LOAD R1, 0 instruction to LOAD R1, 1

2 Oversized Patch

- If the patch requires more memory than is currently occupied by the code to be replaced, it is considered an oversized patch.
- In this case, a JUMP to some unused portion of memory is required, followed by the patched code followed by the return jump to the next significant location.

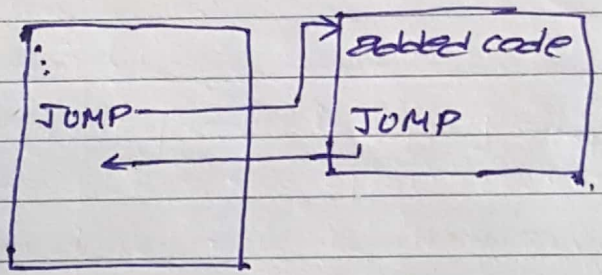

fig: Oversized patch.

- The loading of patches during system integration is often automated.
- However, large number of patches causes confusion. So, careful record should be maintained. to all the patches made as it helps in maintenance.
- Final testing should be done on patched system.

- The patching of the software written in object-oriented languages is very difficult because of the lack of straightforward mapping from the source code to the object code.

15.

The probe effect

→ The effect on the component or system by the measurement instrument when the component or system is being measured.