© 2014 IJIRT | Volume 1 Issue 5 | ISSN : 2349-6002

DEBUGGING

Nonika Sharma, Priyanka Sahni
Information Technology
Dronacharya College Of Engineering, Gurgaon

Abstract- **Debugging is a methodical process to find and reduce the number of bugs, or defects in a computer programme. Debugging becomes harder when various subsystems are tightly coupled, as changes in one may cause bugs to emerge in another. Numerous books have been written about debugging , as it involves numerous aspects, including interactive debugging, control flow, integration testing, log files, monitoring, memory dumps, profiling , statistical process control, and special design tactics to improve detection while simplifying changes. In our research paper we're going to focus on techniques of debugging , debugging for embedded system ,what is anti-debugging.**

I. INTRODUCTION

As software and electronic systems have become generally more complex, the various common debugging techniques have expanded with more methods to detect anomalies, assess impact, and schedule software patches or full updates to a system. The words "anomaly" and "discrepancy" can be used for more neutral terms, to avoid the words "error" and "defect" or "bug" where there might be an implication that all so-called errors, defects or bugs must be fixed. Instead, an impact assessment can be made to determine if changes to remove an anomaly (or discrepancy) would be cost-effective for the system, or perhaps a scheduled new release might render the change(s) unnecessary. Not all issues are life-critical or mission-critical in a system. Also, it is important to avoid the situation where a change might be more upsetting to users, long-term, than living with the known problem(s) (where the "cure would be worse than the disease"). Basing decisions of the acceptability of some anomalies can avoid a culture of a "zero-defects" mandate, where people might be tempted to deny the existence of problems so that the result would appear as zero defects. Considering the collateral issues, such as the cost-versus-benefit impact assessment, then broader debugging techniques will expand to determine the frequency of anomalies (how often the same "bugs" occur) to help assess their impact to the overall system.

II. TECHNIQUES

- **Print debugging** (or tracing) is the act of watching trace statements, or print statements, that indicate the flow of execution of a process. This is sometimes called printf debugging**,** due to the use of the printf function in C. This kind of debugging was turned on by the command TRON in the original versions of the novice-oriented basic programming language. TRON stood for, "Trace On." TRON caused the line numbers of each BASIC command line to print as the program ran.
- **Remote debugging** is the process of debugging a program running on a system different from the debugger. To start remote debugging, a debugger connects to a remote system over a network. The debugger can then control the execution of the program on the remote system and retrieve information about its state.
- **Post-mortem debugging** is debugging of the program after it has already crashed. Related techniques often include various tracing techniques and/or analysis of memory dump of the crashed process. The dump of the process could be obtained automatically by the system (for example, when process has terminated due to an unhandled exception), or by a programmer-inserted instruction, or manually by the interactive user.
- **"Wolf fence" algorithm:** Edward Gauss described this simple but very useful and now famous algorithm in a 1982 article for communications of the ACM as follows: "There's one wolf in Alaska; how do you find it? First build a fence down the middle of the state, wait for the wolf to howl, determine which side of the fence it is on. Repeat process on that side only, until you get to the point where you can see the wolf."This is

implemented e.g. in the Git version control system as the command git bisect, which uses the above algorithm to determine which commit introduced a particular bug.

- **Delta Debugging** - technique of automating test case simplification.
- **Saff Squeeze** - technique of isolating failure within the test using progressive inlining of parts of the failing test.

III. DEBBUGING FOR EMBEDDED SYSTEM

In contrast to the general purpose computer software design environment, a primary characteristic of embedded environments is the sheer number of different platforms available to the developers (CPU architectures, vendors, operating systems and their variants). Embedded systems are, by definition, not general-purpose designs: they are typically developed for a single task (or small range of tasks), and the platform is chosen specifically to optimize that application. Not only does this fact make life tough for embedded system developers, it also makes debugging and testing of these systems harder as well, since different debugging tools are needed in different platforms.

- to identify and fix bugs in the system (e.g. logical or synchronization problems in the code, or a design error in the hardware);
- to collect information about the operating states of the system that may then be used to analyze the system: to find ways to boost its performance or to optimize other important characteristics (e.g. energy consumption, reliability, real-time response etc.).

IV. ANTI-DEBBUGING

Anti-debugging is "the implementation of one or more techniques within computer code that hinders attempts at reverse engineering or debugging a target process".It is actively used by recognized publishers in copy-protection schemas, but is also used by malware to complicate its detection and elimination. Techniques used in anti-debugging include:

- API-based: check for the existence of a debugger using system information
- Exception-based: check to see if exceptions are interfered with
- Process and thread blocks: check whether process and thread blocks have been manipulated
- Modified code: check for code modifications made by a debugger handling software breakpoints
- Hardware- and register-based: check for hardware breakpoints and CPU registers
- Timing and latency: check the time taken for the execution of instructions
- Detecting and penalizing debugger

An early example of anti-debugging existed in early versions of Microsoft Word which, if a debugger was detected, produced a message that said: "The tree of evil bears bitter fruit. Now trashing program disk.", after which it began erasing the Word floppy disk with zeros.

REFERENCES

[1] Grace Hopper from FOLDOC

[2] https://en.wikipedia.org/wiki/J._Robert_Oppenheimer

[3] https://en.wikipedia.org/wiki/Ernest_Lawrence

[4] http://bancroft.berkeley.edu/Exhibits/physics/images/bigscience25.jpg

[5] S. Gill, The Diagnosis of Mistakes in Programmes on the EDSAC, Proceedings of the Royal Society of London. Series A, Mathematical and Physical Sciences, Vol. 206, No. 1087 (May 22, 1951), pp. 538-554

[6] Robert V. D. Campbell, Evolution of automatic computation, Proceedings of the 1952 ACM national meeting (Pittsburgh), p 29-32, 1952.

[7] Alex Orden, Solution of systems of linear inequalities on a digital computer, Proceedings of the 1952 ACM national meeting (Pittsburgh), p. 91-95, 1952.

[8] Howard B. Demuth, John B. Jackson, Edmund Klein, N. Metropolis, Walter Orvedahl, James H. Richardson, MANIAC, Proceedings of the 1952 ACM national meeting (Toronto), p. 13-16

[9] The Compatible Time-Sharing System, M.I.T. Press, 1963

[10] Peggy Aldrich Kidwell, Stalking the Elusive Computer Bug, IEEE Annals of the History of Computing, 1998.

[11] Postmortem Debugging, Stephen Wormuller, Dr. Dobbs Journal, 2006

[12] E. J. Gauss (1982). ""Pracniques: The "Wolf Fence" Algorithm for Debugging",".

[13] Andreas Zeller: *Why Programs Fail: A Guide to Systematic Debugging*, Morgan Kaufmann, 2005. ISBN 1-55860-866-4

[14] Kent Beck, Hit 'em High, Hit 'em Low: Regression Testing and the Saff Squeeze
[15] Shields, Tyler (2008-12-02). "Anti-Debugging Series - Part I". *Veracode*. Retrieved 2009-03-17.
[16] Software Protection through Anti-Debugging Michael N Gagnon, Stephen Taylor, Anup Ghosh
[17] Ross J. Anderson. *Security Engineering*. p. 684. ISBN 0-471-38922-6.
[18] "Microsoft Word for DOS 1.15".