Volume 5, Issue 10, October-2015 ISSN: 2277 128X
International Journal of Advanced Research in Computer Science and Software Engineering
Research Paper
Available online at: www.ijarcsse.com

Program Analysis Using Code Instrumentation Techniques

Dnyaneshwar S. Panchal*, Raju Maher, S. N. Deshmukh
Dept of CS and IT, Dr. B.A.M.U. Aurangabad,
Maharashtra, India

Abstract*— In C or C++ program Memory is allocated but not freed, this allocated memory consuming memory and acquires valuable space of memory. Other one is Memory deallocated more than once i.e. double free, deleting unallocated memory, these are the examples which decrease the performance of executable program. So that we use a technique depends on dynamic program analysis and results of the pin tool. To find memory leakages, using source as well as binary level, pin tool is help full to find memory leakages created with the help of pin framework developed by Intel.

Keywords*— Instruction, Static, Dynamic, Framework, Profiling, Instrumentation, Memory Leakages, pin tool, analysis.

I. INTRODUCTION

Software application errors can be classified based on occurrence time of error. Most errors are captured after completion of software, and can be removed quickly. But it is not possible that all errors are detected during compilation and linking. Remaining errors are found at program execution time. Unluckily, it is depend on input data and can be very hard to detect. Even they can be reproduced the reason of the error and the exact location in the source code is usually difficult to find. One possibility is to detect and remove these errors by using runtime detection application. These applications are able to detect errors when they occur and to report the programmer where they are located in the source code, so that they can be fixed easily.

C or C++ is very strong and expressive language: It supports low-level control of the hardware, which makes it adjustable for performance critical applications and application programming. However, this expressiveness is obtained by allowing the programmer direct access to the hardware, and therefore to overcome many built-in limitations of the language, like the type system. In C/C++ it is, for example, allowed to access any memory location by using any kind of pointer, and each type can be casted to each other kind. Memory management is clearly handled by the programmer as compare to automatic memory management, like other languages implemented i.e. JAVA and C# [1].

EXAMPLE OF INSTRUCTION COUNT BY USING CODE INSTRUMENTATION TECHNIQUE

```
#include <stdio.h>
#include "pin.H“
UINT64 icount = 0;
VOID IncCounter(INT32 c) { icount += c; }
VOID Trace(TRACE trace, VOID *v) {
for(BBL b=TRACE_BblHead(trace); BBL_Valid(b); b=BBL_Next(b)){
BBL_InsertCall(b, IPOINT_BEFORE, (AFUNPTR)IncCounter,
IARG_UINT32, BBL_NumIns(b), IARG_END);
}}
VOID Fini(INT32 code, VOID *v) {
fprintf(stderr, "Count %lld\n", icount);
}
int main(int argc, char * argv[])
{
PIN_Init(argc, argv);
TRACE_AddInstrumentFunction(Trace, 0);
PIN_AddFiniFunction(Fini, 0);
PIN_StartProgram();
return 0;
}
```

Fig. 1: Example of Instruction count using code instrumentation

A. Memory Leakages

Maximum C/C++ program memory is allocated at runtime, if allocated memory is not freed; then this free memory acquiring the executables and decreases the performance of an application. Therefore a written program will go on consuming runtime memory and not freed the acquired memory. To solve or avoid this, the program is needed to be checked for memory leakages. To detect the memory leaks, we keep a track of memory allocations and deallocations occurring while program execution time and report the detected memory leakages and there type to the developer [5].

B. Instrumentation Technique for Program Analysis

Code Instrumentation is a technique used to analyse the program at static and dynamic level. We use this technique for dynamic program analysis. "Code Instrumentation means inserting extra code into a program instruction for the purpose of modifying the program instruction to change the programs behaviour or to analyse the program to improve the whole performance of application". Here we use this technique for program analysis and modifying the code for profiling the application program, determine memory leakages occurred during program execution time. To collect all these profiling information we use tools called as pin tool created and developed by using pin framework developed by Intel [2].

C. Pin Tool

Pin tool is a set of program instruction which is developed with the help of pin framework. This framework is multiplatform, can work on Windows, Linux, Mac OS and Android and executables for the IA-32, Intel(R) 64 and Intel(R) Many Integrated Core architectures. Pin allows a tool to insert any code in any places in executable of C or C++. The code is added dynamically while executable is running. Pin provides a rich API and also includes lot of tools those are executing with respect to binary i.e. executable. By performing the instrumentation on the binary at runtime, Pin eliminates the need to modify or recompile the application's source and supports the instrumentation of programs that dynamically generate code. It allows register content to be passed to the code as parameter. This tool is easy to use, portable and robust and it uses Just in Time (JIT) in order to insert and optimize code. Its ability to inlining, register allocation and instruction scheduling makes this instrumentation tool different from others. In order to instrument, profile, or perform other analysis, the developer simply attaches to the already running application. When necessary data has been collected after that PIN tool can detach from an application. The important point about PIN is analysis can performed at specific points in application run time.

Pin's Software Architecture

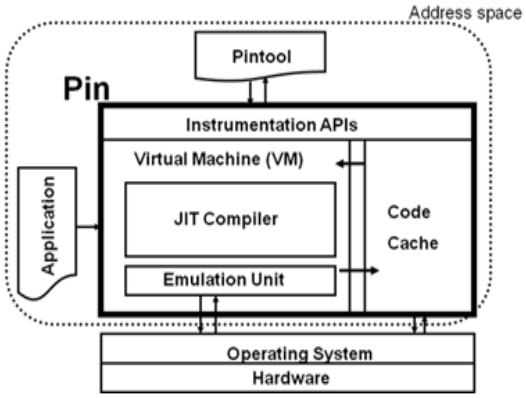

Fig. 2 Pin's Software Architecture

Instrumentation consists of two components shown below.

a. Mechanism that decide what and where code is inserted.
b. The code to execute at insertion points. These two components are instrumentation code and analysis code.

These components live in an executable called Pin tool, which can be act as plug-in that modifies the code generation process inside Pin. The Pin tool registers instrumentation callback routines with Pin that are called from Pin whenever new code needs to be generated. This instrumentation callback routine represents the instrumentation component. It inspects the code to be generated, investigates its static properties, and decides if and where to inject calls to analysis functions. Pin and the Pin tool control a program starting with the very first instruction. For executable compiled with shared libraries this implies that the execution of the dynamic loader and all shared libraries will be visible to the Pin tool. It's more important to adjust analysis code than the instrumentation code because instrumentation is executed once whereas the analysis code called many times. With the help of pin tools many multithreaded applications can be analysed and executable can be instrumented easily by using various tools like simple examples, memory trace, Cache analysis (All cache), Manual examples to cover procedure count, instruction tracing counting jump instructions and branch counted for taken and executable as well as to identify misses and other parameters. It is possible to know static properties to use pin to examine binaries without instrumenting them. Probe mode is a method of using Pin to insert probes at the start of specified routines. A probe is a jump instruction that is placed at the start of the specified routine.

The probe redirects the flow of control to the replacement function. Before the probe is inserted, the first few instructions of the specified routine are relocated. It is not uncommon for the replacement function to call the replaced routine. In probe mode, pin overwrites the entry point of procedures with jumps to dynamically generated instrumentation. When pin tool instrument a parallel program application threads execute the call to analysis functions. Pin allows instrumentation of parallel programs that use multiple threads and multiple cooperating processes. The new thread executes the same instrumented code as the other threads and accesses the same data. There are variety of pin based tool to analyse parallel program performance and correctness. These tools analyse the cache parameters and provide accuracy for application [6].

Pin includes the source code for a large number of example instrumentation tools like basic block profilers, cache simulators, instruction trace generators, etc. It is easy to derive new tools using the examples as a template.

II. RELATED WORK

A. Memory Leak

Dynamic Binary Instrumentation analysis framework developed until now, maximum used application is for developing memory supervisor tools. Dynamic Binary Instrumentation is applicable as compare to common memory detection tool in the support of underlying system and accuracy detection. Up till a lot of Dynamic Memory Supervising tools from DBI technologies emerged. Maximum tools detect the memory leaks using situation of a program as well as memory errors available in program, illegal use of memory and also detect buffer overflow accurately. DBI dependent memory supervising tools and related things are as bellow.

1) Memcheck: Memcheck is a memory error detector. It can detect many common problems found in C and C++ programs, such as: Accessing unwanted memory, using undefined values, faulty freeing of heap memory, memory leaks etc.

2) Dr. Memory: Dr. Memory is created on the open-source dynamic instrumentation platform DynamoRIO. It supports Windows and Linux and it is perfect memory checker. Dr. Memory uses Memory shadowing to track properties of a target applications data during execution. So that it can detect error more precisely. Other advance is, Dr. Memory provide two instrumentation paths: the fast-path and the slow-path. The fast path is implemented as a set of carefully hand-crafted machine-code sequences or kernels covering the most performance-critical actions. Fast-path kernels covering the most performance-critical actions. Fast-path kernels are either directly inlined or use shared code with a fast subroutine switch. Rarer operations are not worth the effort and extra maintenance costs of using hand-coded kernels and are handled in the slow-path in C code with a full context switch used to call out to the C function. Through using different path in different situation, the efficiency of detection is increased greatly [9].

III. OBJECTIVE

Objective of this technique is to build a tool for the analysis of programs. For this, static analysis could be coupled with dynamic analysis, Depending on the problem statement. The problem should be analyzed to the best making use of both static and dynamic analysis methods. To begin with the problem of memory leakages in C or C++ programs has been relate with dynamic analysis and used to generate a report that produces an output about detailed analysis of the program with respect to memory leaks. It produces information about the memory allocations that have been allocated but not freed. It also gives information of the invalid frees. The other problem solved with this way is of the uninitialized variables. Many tools are available to detect the uninitialized variables using static analysis methods. This process gives large results consisting of many false positives. Source code instrumentation and binary instrumentation together produces a report stating all the uninitialized variables that are used along the path that has been taken during the program execution. This removes all the false positives [9].

IV. METHODOLOGY

A. Source code modification.

By this way we can instrument the code via source code modification and recompilation. The main drawback of this technique is that all controlled methods have to be reparsed and recompiled. Furthermore, another recompilation is needed to reinstall the original methods.

B. Bytecode modification.

Another way to control Program flow is to directly insert byte-code into the byte-code of the compiled methods. However, this technique relies heavily on deep knowledge of the bytecode instructions used by the virtual machines. Another potential danger of this technique is that these codes are not standardized and can change.

C. Instrumenting the Virtual Machine.

A technique for collecting runtime information is instrumenting the runtime environment in which the system runs. This technique instruments the Virtual machine itself to generate the events of interest. But this method is dangerous and can challenge the consistency and integrity of virtual system. Implementing such instrumentation requires a profound and in depth knowledge of virtual machine.

D. Method Wrappers.

Wrappers are mechanisms for introducing new behavior that is executed before and/or after, and perhaps instead of, an existing method. Rather than changing method lookup, they modify the method objects that the processes return.

Wrappers are easy to build for Smalltalk as it was designed with reflective facilities that allow programmers to intervene in the lookup process, while the same is not true for Java which only supports introspection.

E. Debuggers.

It is possible to run a system under the control of the debugger. In this case, breakpoints are set at locations of interest (e.g., entry and exit of a method). This technique has the advantage of not modifying the source code and the environment. However it slows down the execution of a system considerably and the instrumentation itself can be heavy.

F. Logging

Log that records either the events which happen while pin debugger runs, or the messages between different processes of communication software. The act of keeping a log file is called logging. In the simplest case, log messages are written to a single log file. We can display the record like line number, error type etc [8].

V. EXPERIMENTAL WORK

A. Memory Leak

1) Creating instrumenter to find Memory Leak (pintool).

The objective of this instrumenter is to find memory leakages problems which are common to C or C++ programs. The Pintool is analyzing an available program without modifying the source code or instead of recompilation. The reason behind this is Pin performs its work at program execution time. Some memory related problems are as follows.

1) Memory leaks: Memory allocated, but not freed.
2) Double freeing Memory: deallocation is repeated i.e. more than once.
3) Freeing unallocated memory: Trying to deallocate memory that has not allocated.

2) Detection of Memory leakages

To detect those memory troubles, the Pintool required supervising calls to the allocation and deallocation functions. Since the new operator calls malloc internally, and the delete operator calls free internally, we just supervise the calls malloc and free.

Whenever the program calls malloc, tool will record the returned location (address, either null or the address of the allocated memory region). Whenever it calls free, it will match the address of the memory being freed with my records. If it has been allocated but not freed, it will mark it as freed. Though, if it has been freed allocated and freed, that would be an attempt to free it again, which shows the error. At last, if there is no record the memory being freed has been allocated, that would be an attempt to free unallocated memory. When the program terminates, it will again check records for those memory regions that have been allocated but not freed to detect memory leaks [15].

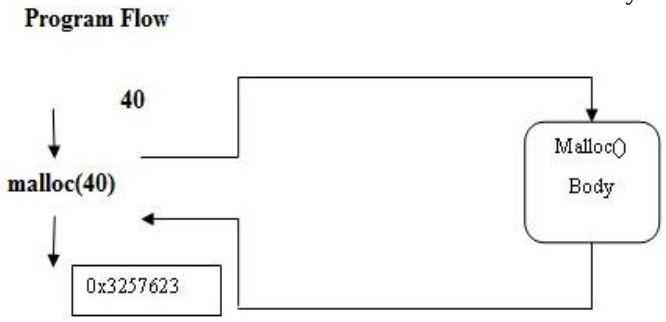

Fig. 3 Program Flow before Instrumentation

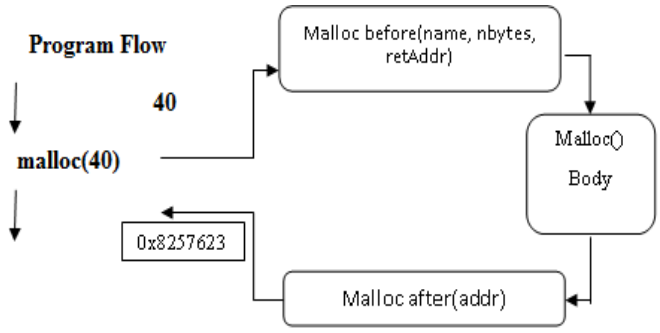

Fig. 4 Program Flow after Instrumentation

3) Source Code Instrumentation

For instrumentation of source file, we need to insert extra code to every function. Because this code will add all the information about local variables of the function (Global variables are already initialized) to the log file. We use lexical analyzer to find to parse the source (.c file) and detect all the variables defined and their locations. For each variable its name, address and size is needed for analysis. For structures and multidimensional arrays additional information is

needed (detail information of member variables or size of all dimensions). Using this information, source code can be instrumented to generate log. After all variables are declared, extra code will be added, also, if the variable is already initialized, it is not logged.

4) Binary Code Instrumentation

After first step, compilation is next step; compile the program code to generate the executable file i.e. binary file. This file is again instrumented at binary level. Every executed instruction scanned for memory reads and writes. If any instruction is reading or writing, all the details including the instruction address, memory address of read or written location and size of data are logged. All this information can be extracted from the instruction in binary form. This instrumentation will also write into the same log file uninit.log [4].

VI. CONCLUSIONS AND FUTURE SCOPE

In this research we use Code Instrumentation Technique to analyze the program Behavior of C or C++ program to find memory leakages and release it back. Analyzing program behavior is difficult and complicated to abstract away from the whole program. Pin tools are used to instrument C or C++ code. Pin Tool are developed using pin framework, are better to use, easy to analyze the program behavior at runtime, can able to solve the problems occurred during program execution time for improving the performance of application program. Memory leakages are found many times in C or C++ which are decrease the performance of C or C++ application program. Pin tool resolve this problem by profiling the executables, collect information in the form of a log using instrumenting the running Program. To change the behavior of application program Pin tool is applicable and user friendly.

Future scope for this technique is we can develop different pin tools with the help of pin framework for profiling the running program, detect faults in c or c++ program, and resolve them. Which are helpful to improve the performance of a running application program? By profiling the program behavior we can track and resolve different executable error and abstract away from the underlying features of whole program also improve the whole performance of C or C++ program.

REFERENCES

[1] Philip J. Guo, Stephen McCamant, "A Scalable Mixed-Level Framework for Dynamic Analysis of C/C++ Programs", MIT Computer Science and Artificial Intelligence laboratory, 32 Vassar Street, Cambridge MA, 02139 USA, pp. 1-2.

[2] https://software.intel.com/sites/landingpage/pintool/docs/62732/Pin/html/

[3] https://groups.yahoo.com/neo/groups/pinheads/info

[4] Chi-Keung Luk, Robert Cohn, Robert Muth, Harish Patil, Artur Klauser, Geoff Lowney, StevenWallace Vijay, Janapa Reddi, Kim Hazelwood, "Pin: Building Customized Program Analysis Tools with Dynamic Instrumentation", Intel Corporation, University of Colorado.

[5] Martin Bauer, Prof. Dr. U. Rude, M.Sc. K. Igberger, Dr. T. Panas, "C/C++ Error Detection Using Source Code Instrumentation", 2010, pp: 1.

[6] Aditya Khamparia, Saira Banu J., "Program Analysis with Dyanmic Instrumentation Pin and Performance Tools", Computer Science and Engineering Institute of Technology, SCSE, Vellore, India, 2013, pp. 1-5.

[7] Valgrind Developers, "Valgrind Quick Start Guide" , 21 October 2010, pp. 1-141.

[8] Markus Denker, Orla Greevy, Michele Lanza, "Higher Abstraction for Dyanamic Analysis", Software Composition Group, University of Berne, Switzerland, Faculty of Informatics University Lugano Switzerland.

[9] Kunping Du, Fei Kang, Hui Shu, Li Dai, "Dynamic Binary Instrumentation Technology", National Digital Switching System Engineering & Technological Research Centre, Zhengzhou, China, 0371-81632000, hnzzdkp@163.com

[10] Walter Binder, "Higher-Level Abstractions for Instrumentation-based Dynamic Program Analysis", Dynamic Analysis Group, Faculty of Informatics University of Lugano.

[11] Anushri Jana, Ravindra Naik, "Precise Detection of Uninitialized variables using Dynamic Analysis – Extending to Aggrigate and Vector Types", TCS Innovation and Design Centre Hadapsar, Pune, India 411 013, anushri.jana@tcs.com, rd.naik@tcs.com, 2012, pp. 197-201.

[12] Milind Chabbi, Xu Liu, John Mellor-Crummey, "Call Paths for Pin Tools", Department of Computer Science, Rice University Houston, TX, USA.

[13] Vasanth Bala, Evelyn Duesterwald, Sanjeev Banerjia, "Dynamo: A Transparent Dynamic Optimization System", Hewlett-Packard Labs, 1 Main Street, Cambridge, MA 02142, pp. 1-11, 2000.

[14] James H. Hill, Dennis C. feiock, " Pin++: A Object-oriented Framework for Writing Pintools", Department of Computer and Information Science, Indian University-Purdue University Indianapolix, IN USA, pp. 1-10.

[15] http://msdn.microsoft.com/en-us/magazine/dn818497.aspx

