An automatic test for the software reliability: the evaluation of the overflow due to memory leaks as failure cause

Marcantonio Catelani[(1)], Lorenzo Ciani[(1)], Valeria L. Scarano[(1)], Alessandro Bacioccola

[(1)]University of Florence, Department of Electronics and Telecommunications,
via S. Marta 3, 50139, Florence (Italy)
marcantonio.catelani@unifi.it lorenzo.ciani@unifi.it valeria.scarano@unifi.it

***Abstract*:** In order to guarantee a software with high quality level vs a reasonable cost, the testing planning phase has to be study in detail, also to reduce the test time [1]. This paper introduces the result of the application of software automatic test to one of the most common software failure causes: the overflow due to memory leaks. The analysis carried out on a single software failure cause wants to show the total generality of the technique and to define some parameters of life of the software. The general value of the method is shown applying the technique to different software and for everyone of them defining the mean time to failure, where with failure we mean only the failure due to overflow; even if the overflow it isn't the only analyzable failure cause through the application of such software automatic test.

I. Introduction

The most important applications of the software automatic test are regression test and test to locate the memory leaks.
Regression test is the classic software automatic testing application and it is the main part of the acceptance test. This kind of software testing is suitable for automatic tests with continuous repetitions of the same operations or test sequences, carried out in order to verify that the new functionalities implemented doesn't penalize the old ones [1].
The continuous repetitions of an operations set permit to evaluate also the memory occupation by the software modules; so monitoring the occupation of the total memory of the computer and that one engaged by the single process, it is possible to underline the presence of memory leaks by one or more software modules.
Memory leaks and memory corruption are the two major software bugs that severely threaten system availability and security. Memory leaks, caused when some allocated memory isn't accessible again, can cumulatively degrade overall system performances increasing memory paging. Or worse, they may cause programs to exhaust system resources, eventually leading to program crash. The system memory, allocating by the processes, is managed by the developer trough rules set. All programming languages have default rules necessary to manage the memory. If the developer doesn't specify any rule to manage memory, the compiler sets the default settings. These rules are optimized for many applications but not for all.

II. Proposed Approach

The software automatic test, shown in this paper, can be classified as dynamic, which means that it is able to stimulate the software under test with a stress level comparable to the real use in order to evaluate the memory leaks.
In order to estimate the occupation of the total memory of the machine on which the SW under test turns and that one occupied from its single modules we use the Performance Monitor of Microsoft, distributed from the software house with the Windows XP operating system. Such SW is chosen because all the tested tasks are accommodated by the Microsoft operating system, moreover this analysis tool turns out intuitive and light (doesn't slow down the normal execution of the monitored SW), offering all the instruments necessary to this kind of survey.
The Performance Monitor allows to define a list of processes that have to be observed; for every process we can choose the parameters to analyze, such to be observed or in real time or successively (our choice). For each process we estimate:

- Private bytes: displays the number of bytes reserved exclusively for to specific process. If a memory leak is occurring, this value will tend to steadily rise.
- Working set: it is the current size of the memory area that the process is utilizing for tails, threads, and date. The size of the working set will grow and shrink as the VMM (virtual memory manager) can permit. When memory is becoming scarce the working sets of the applications will be trimmed. When memory is plentiful the working sets are allowed to grow. More larger working sets mean tails and date in memory making the overall performance of the applications increase. However, to large working set that doesn't shrink appropriately is usually an indication of a memory leak.

Such parameters will be estimated on single operating task and on total occupied memory of the computer. During the testing the SW has to be opportunely stimulated, repeating a series of normal use operations. The stimulation on the

tested products has been executed by macro realized in Macro Magic of the Iolo software. The choice of the Iolo software is due to its litheness, in fact, it doesn't induce the fall of the machine performances on which it is active.
We apply the described procedure to two wide use trade SW, a task of management for stations of fuel distribution and a data base controller, the obtained results are shown in this paper.

III. Application

A. Application to the complete petrol station management

The first analyzed software is spread task of management of fuel stations, in its complete version is made up by approximately 40.000 modules and 5.000.000 of code lines. The software allows the complete management of the petrol station, from the fuel sale to its supplying, from the warehouse management to the accounts department. It has been characterized a typical sequence that will frequently repeat on the service stations.
The implemented sequence involves the following operations, that are all real operations with connection to test Terminal Management (TM) :

1. Login cashier;
2. Sale of an article, yet in warehouse, paying it cash with points credit on the fidelity card;
3. Sale of a distribution, paying it cash with points credit on the fidelity card;
4. Redemption of a fidelity prize;
5. Logout cashier and cash closing.

The sequence has been repeated carrying out 2233 sales in 40 hours, for an average of approximately 56 sales/hour; number 4 times greater than the conditions on field of 14 sales/hour, that are carried out for every POS (Point of Sale). The sequence through the macro simulates the interaction between the software and the controller, the simulator of the pumps recreates the interaction between the system, the customer and the connection to the services centre of the oil company for the management of the fidelity card, all real situations that occur on field.
During the tests we monitored the occupation of the total memory of the computer and five operations that the field feedbacks signalled like critic in terms of memory requirement. For every observed task we monitored private bytes and working set. The accountable task of the interface to operator has shown the trend plots in figure 1, where the red line represents the private bytes and the black one the working set.

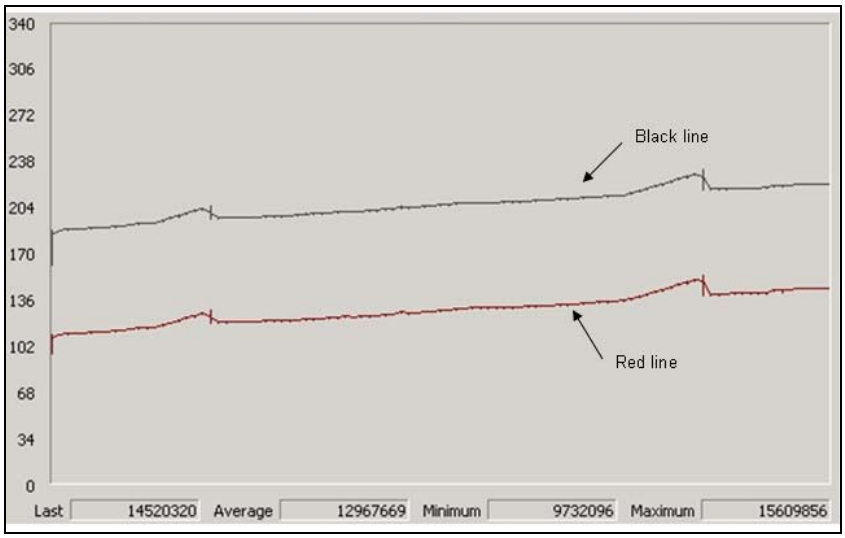

Figure 1: Trend of memory occupation from the customer interface manager.

The trend of the occupied memory grows linearly with time and sales number, we assume that the sales are distributed in uniform way during the considered time interval. Knowing that the computers, on which the software under tests is accommodated, have 256 Mb (268.435.456 byte) of memory and assuming that these ones are only dedicated to the exclusive use of this process, we can carry out the calculation of the hour memory variation:

$$Delta = \frac{(Max - Min)}{t} = \frac{5.877.760}{40} = 146.944 bytes/h \quad \textbf{(1.1)}$$

So we can obtain the Mean Time to Overflow (MTOF):

$$MTOF' = \frac{RAM}{Delta} = 1826{,}787h \cong 76 days \quad \textbf{(1.2)}$$

Considering that the distributor server is never off and that the management software stays on field for a mean time of 3 months before the maintenance; it appears that this process would carry to a crash of the server after 76 days of operativeness that is before the maintenance operation of installation of the new version; instead when the RAM memory value is achieved, the operating system executes a swap on the physical disc.
So we can consider the trend of the total occupied machine memory under investigation and the maximum dimension of the swap partition as 1 Gb (1.073.741.824 byte). From the performance monitor we can observe that at computer start, before executing any solicitation, the memory already occupied by all the processes (named as Mem0) is equal to 327.122.944 bytes; therefore yet to the test start the system is carrying out the swap on disc. The total memory that the process (named as Mem) can allocate is:

$$Mem = swap - Mem0 + RAM = 1.073.741.824 - 327.122.944 + 268.435.456 = 970.054.336 bytes \quad \textbf{(1.3)}$$

After the new considerations we can evaluate the MTOF as:

$$MTOF = \frac{Mem}{Delta} = 6601.524h \cong 275 days \quad \textbf{(1.4)}$$

The swap on the disc permits to the software to carry out nearly three cycles life, remembering the hypothesis that the process under investigation is the only one that occupies the memory. However from the (1.1) appears the necessity to expand the RAM of the machine on field in order to assure a greater reliability.

B. Application to a database controller

The second tested task is a database controller, from a customer interface the employee inserts a series of data that then are recorded in a big database. The database is saved on two different discs (main and secondary), at 4,00 AM the manager of the database carries out its reconstruction. In order to test this task, we realised a macro that it inserts the data in the employ interface. The test has involved the insertion of 1000 records in 36 hours and then we observed the trend of the memory occupied from the database controller, measuring the private bytes and the working set, respectively the blue line and the yellow one, as is shown in figure 2.

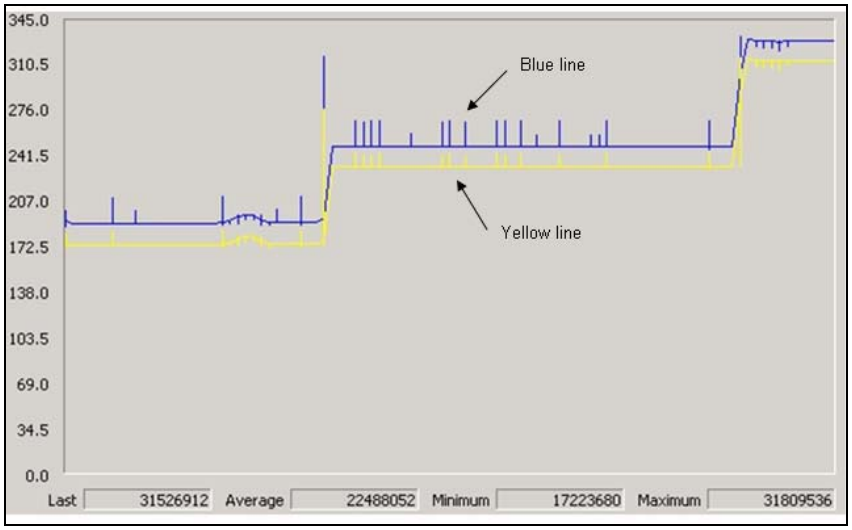

Figure 2: Trend of memory occupation from the database controller

The trend of the memory is constant except for the presence of some instantaneous peaks till to the execution of the database reconstruction. Every time that the reconstruction is executed, the process occupies approximately 10 Mb/day (Delta) of memory, the database server has 512 Mb of RAM and 1.5 Gb (1536 Mb) of swap. Moreover we have to consider that the RAM occupied at the start is approximately 280 Mb (Mem0). According the (1.3) and (1.4) we obtain:

$$Mem = swap - Mem0 + RAM = 1536 - 280 + 512 = 1768 Mb \quad \textbf{(1.5)}$$

$$MTOF = \frac{Mem}{Delta} = 1768 / 35 \cong 51 days \quad \textbf{(1.6)}$$

The server overflow is caught up after 51 days, the server operates every day the reconstruction of database, therefore we will wait 7 crashes due to overflow in one year; as in the previews application it is valid the hypothesis that the process under investigation is the only one that occupies the memory.

IV. Conclusions

The results of the carried out tests on two trade tasks underline the generality of the proposed method besides the effectiveness of SW automatic test in order to characterize memory leaks due to each module of an extensive program.
We obtained that the first task has a linear increase of the occupied memory due to the customer interface; instead the second task has shown the increase of the memory occupied only after the insertion operation in the data base. In order to increase the reliability of the software product, the hardware upgrade of the RAM of the sold machine becomes necessary. The MTOF (Mean Time to Overflow) parameter, calculated for the tasks, allows to estimate the crash of the system due to overflow, it will allow to estimate the availability of the software product in order to plan the right maintenance operations.
Other parameters useful in order to control memory leaks could be handle counts for each process and virtual bytes. If a memory leak is occurring, an application could create additional handles to identify memory resources, so a rise in the handle count might indicate a memory leak. Virtual bytes, that are the current size of the virtual address space that the process is using, doesn't necessarily imply corresponding use of either disk or main memory pages, but virtual space is however finite and, using too much, the process may limit its ability to load libraries

References

[1] M.Catelani, L.Ciani, V.L.Scarano, A.Bacioccola "A novel approach to automated testing to increase software reliability", "2008 IEEE International Instrumentation and Measurement Technology Conference" proceedings, Vancouver, Canada, May 12-15, 2008;
[2] A. Birolini, "Reliability Engineering – Theory and Practice", Springer 2004;
[3] N. Juristo, A. M. Moreno, "Software Testing Practice in industry", IEEE software 2006, p.19-21;
[4] J. D. Musa, "Introduction to Software Reliability Engineering and Testing", Proceedings of the 8th International Symposium on Software Reliability Engineering, 1997;
[5] ANSI / IEEE St. 829,"Standard for software test documentation", 1998;
[6] ANSI / IEEE Std. 1012, "IEEE Standard for Software Verification and Validation Plans", 1986;
[7] ISO/IEC 9126: Information Technology- Software Product Evaluation- Quality Characteristics and Guidelines for their Use, 2001;
[8] Diaz E., Tuya J., Blanco R., "Automated software testing using a metaheuristic technique based on tabu search", Proceedings of 18th IEEE international conference on automated software engineering, 2003, p. 310-313;
[9] ANSI / IEEE Std. 729, "Standard Glossary of Software Engineering Terminology", 1983.