Robustness Measurement in OS Forecast and Selection

Xiaoen Ju and Hengming Zou
School of Software and Department of Computer Science
Shanghai Jiao Tong University
Shanghai 200240, China
{xiaoen.ju, zou}@sjtu.edu.cn

Abstract

Commercial Off-the-Shelf (COTS) operating systems have long been widely used. However, the problem concerning the robustness of such OSs is far from being solved. Although many efforts have been made in this research domain, people still find it difficult to make choices among various OSs for robustness concerns. This fast abstract presents a brief summary of our OS robustness measurement in our work of constructing a reference model for OS forecast and selection. We measure both the robustness of OS APIs and their usage frequency in typical operational profiles. Our research has obtained some interesting preliminary results and is still ongoing.

1. Introduction

Software robustness has long been the concern of software researchers, engineers, and practitioners. Software failures, especially those in mission-critical areas, keep reminding people that the issue of software robustness can never be underestimated. In particular, due to the impact operating systems have on software running on them, choosing a robust OS is vital for concerned users.

This abstract presents our OS robustness measurement in our work of constructing a reference model for OS robustness forecast and selection. The reference model aims to predict the robustness of OSs in different typical operational profiles (such as Vista's robustness under compute-intensive usage) and select appropriate OSs as the development/operating platform based on robustness requirements. This research is part of a bigger project called Platform-based Software Reliability Forecast and Selection [10], which aims to predict software's maximum achievable reliability under various development/operating platforms.

2. Approach

Due to the important role played by OS APIs in upper level software's robustness, we decide to focus our study on the robustness of the APIs. Our work is divided into two parts: the first part measures the robustness of the APIs; the second part relates API usage to typical operational profiles.

We define: a robustness vector R_{API} with each component R_{API_i} indicating the robustness of the ith API, and an operational profile vector P with each component P_i indicating the usage frequency of the ith API in the profile. Then the overall robustness $R_{overall}$ of the OS in this operational profile can be calculated (predicated) as follows:

$$R_{overall} = R_{API}^T \cdot P$$

Next we describe how to obtain R_{API} and P.

3. Robustness Test

Our testing is derived from Ballista [5]. The methodology is a combination of random testing and fault injection. Well-defined exceptional values are used to inject faults into the OS APIs. Those values are created according to parameter data types (instead of API functions). In other words, we create exceptional values (and also normal values) for each data type such as integer and C-type character, and put them into separate data type pools. When launching a test for one particular API, the test manager randomly picks up values in the pools corresponding to the data types of the API parameters. Like Ballista, we use the CRASH scale [1] to measure the test responses and to report results.

4. Operational Profile Measurement

Since APIs are OS-dependent, the tools used for the measurement vary accordingly. On Windows, we use Detours [2]; on Linux, we use the "strace" system call. Typical operational profiles can be generated by using benchmark programs. For example, SPEC CPU 2006 benchmark is used to simulate a compute-intensive operational profile.

5. Recent Research Result

We have created a Windows Ballista test harness from the POSIX-oriented Ballista project and measured the robustness of 87 C-Library functions and over 100 Win32

APIs on XP and Vista. We have also measured the usage of Win32 APIs in a typical compute-intensive environment with SPEC CPU 2006 INT. Table 1 shows the top ten most frequently called Win32 APIs when running the benchmark on Vista, as well as their robustness values.

Table 1. WinAPI usage with SPEC CPU06 INT

Win32 API (Vista)	Usage	Robustness
WriteFile	43.17%	96.269%
GetConsoleMode	39.63%	100.000%
FileTimeToLocalFileTime	4.28%	99.000%
FileTimeToSystemTime	4.28%	100.000%
ReadFile	2.13%	96.259%
CreateFileA	1.53%	N/A
FindFirstFileA	1.42%	95.948%
GetCurrentDirectoryA	1.42%	55.854%
FindClose	1.41%	92.537%
GetSystemTimeAsFileTime	0.29%	90.000%

The robustness is calculated as $1 - FailureRate$ which is measured by our robustness test program. The robustness for "CreateFileA" API is "N/A" because its total number of test cases is too large and we decide to omit such test for now ("CreateFileA" has 25,903,957,500,432 test cases).

A comparison between our results and those from CMU, published in DSN'00 [8] shows that our robustness values are much higher. One possible explanation for the difference is that our tests were run on XP and Vista while CMU's were run on W2K and earlier versions. Presumably, XP and Vista have improved on the API robustness over W2K.

Another possible explanation may relate to the selection of input values. While our test inputs are generated based on parameter types, the use of the same data type can be different among various Win32 APIs. Consider the situation in which we employ generally defined unsigned long values to test a DWORD parameter that is used as a flag. The API can tell with a simple check that the input value is invalid, and thus omit the checks for other parameters whose validation may be difficult to verify.

The hierarchy and inheritance of data types may also contribute to test skews. In our test, we follow the data type generation rules of the original Ballista project, which stipulate that every data type (except for root type "bType") must inherit from a more generic parent type. This may lead to a situation in which most of the test cases for one data type are actually generated for its parent type(s). Consequently, those test cases may terminate at the check of the first invalid parameter of the child type, and thus skip the checks on the validity of other parameters, which leads to skews.

For example, a file mapping handle type inherits from a handle type. While there are only 12 test cases for the file mapping handle type, there are 72 test cases for handle type. When testing with the file mapping handle type, in most cases the test process returns system error code 6, indicating that an invalid handle is used. While this is a robust response from the OS, we cannot tell from the tests that whether the other parameters are also checked effectively.

Finally, bugs in the original source code of Ballista may have caused false positive OS failures in CMU's tests, which contribute to the differences between the two results.

6. Related Work

The Ballista project tested fifteen OSs that implement the POSIX standard [4] and Shelton et al. [8] expanded the work to six Windows variants. Süßkraut et al. [9] improved the data type definition of Ballista. Mendonça et al. [6] also followed the Ballista methodology and did robustness tests for device drivers on several Windows OSs, including the newly-appeared Vista. Kanoun et al. [3] created DBench and tested the dependability of several Windows and Linux OSs. Miller and his team used fuzz test [7] to study the robustness of applications on different OSs.

7. Conclusion

This fast abstract presented our current study on OS robustness measurement. We followed the Ballista methodology in testing the robustness of OS APIs. In addition, we measured the usage of APIs in typical operational profiles. Some first-hand research results were presented as well. We anticipate to report in next year's DSN conference the full results of our tests and the construction of the OS robustness forecast and selection model based on those results.

References

[1] R. Biyani and P. Santhanam. Tofu: Test optimizer for functional usage. In *Software Enigneering Technical Brief, IBM T. J. Watson Research Center*, number (2)1, 1997.

[2] Detours. research.microsoft.com/sn/detours/.

[3] K. Kanoun, Y. Crouzet, A. Kalakech, A. Rugina, and P. Rumeau. Benchmarking the dependability of Windows and Linux using postmark^{TM} workloads. In *ISSRE'05*.

[4] P. Koopman and J. DeVale. Comparing the robustness of POSIX operating systems. In *29th Intl. Sym. on Fault-Tolerant Computing*, pages 30–37, 1999.

[5] Carnegie Mellon University. Ballista test harness. In *http://www.cs.cmu.edu/afs/cs/project/edrc-ballista/www/*.

[6] M. Mendonça and N. Neves. Robustness testing of the Windows DDK. In *DSN'07*, pages 554–564, 2007.

[7] B. P. Miller, L. Fredriksen, and B. So. An empirical study of the reliability of UNIX utilities. In *Communications of the ACM*, December 1990.

[8] C. Shelton, P. Koopman, and K. Devale. Robustness testing of the Microsoft Win32 API. In *DSN'00*, pages 261–270.

[9] M. Süßkraut and C. Fetzer. Robustness and security hardening of COTS software libraries. In *DSN'07*, pages 61–71.

[10] H. Zou. Software reference modeling for high reliability. In *Proc. First Asia Working Conference on Verified Software, Macau*, October 31, 2006.