November 2020

Heuristic Approach to Detect Software Application Crashes

Joseph Turner

Christopher Burns

Bob Liu

Ashwin Bellur

Heuristic Approach to Detect Software Application Crashes

ABSTRACT

It is valuable for a provider of a software library to gain insight into crashes of an application that may be caused by or otherwise related to the software library. Since the provider does not have access to the application codebase, it is not possible for the provider to obtain direct signals that an application crash occurred, or whether a crash was related to a library from the provider. This disclosure describes the use of a temporary dirty file to detect crashes that may be related to the use of an embedded library in an application. The dirty file is written to disk when the embedded code (library code) is first accessed or when the application is first launched. Upon successful completion of execution of the embedded code, the dirty file is automatically deleted. At a subsequent launch of the application or execution of the embedded code, if the dirty file still exists, it is determined that a crash of the application likely occurred since no application exit signal was received which would otherwise cause deletion of the dirty file.

KEYWORDS

- Crash reporting
- Dirty file
- Embedded code
- Software Development Kit (SDK)
- Code library
- Third-party library

BACKGROUND

Modern software development involves building applications by leveraging third-party libraries that are embedded within an application. For example, such libraries may be provided in the form of a software development kit (SDK) provided by the third party and accessed via an application programming interface (API). Providers of such libraries include operating system vendors, programming language vendors, etc. For providers that provide SDKs, determining that the performance of applications that are built using their SDKs is satisfactory is important.

One important aspect of application performance relates to crashes. It is valuable for a provider of a software library to gain insight into crashes of an application that may be caused by or otherwise related to the software library. Since the provider does not have access to the application codebase, it is not possible for the provider to obtain direct signals that an application crash occurred, or whether such a crash was related to a library from the provider. There are tools (e.g., [1]) available for application-native (direct) crash monitoring. However, such tools do not allow indirect monitoring since the application that crashes would need to call into third-party code to report the crash.

DESCRIPTION

This disclosure describes the use of a temporary dirty file to detect crashes that may be related to the use of an embedded library in an application. The dirty file is written to disk when the embedded code (library code) is first accessed or when the application is first launched. Upon successful completion of execution of the embedded code, the dirty file is automatically deleted. At a subsequent launch of the application or execution of the embedded code, if the dirty file still exists, it is determined that a crash of the application likely occurred since no application exit signal was received which would otherwise cause deletion of the dirty file.

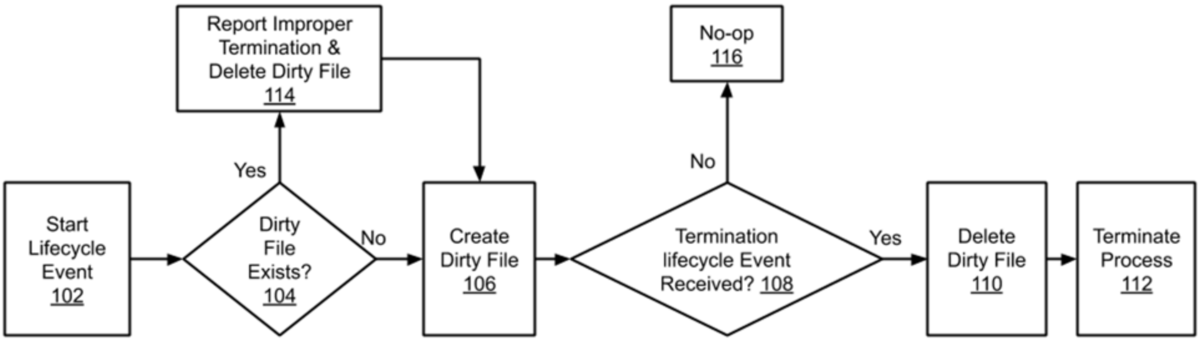
Report Improper Termination & Delete Dirty File 114
No-op 116
Start Lifecycle Event 102
Yes
Dirty File Exists? 104
No
Create Dirty File 106
No
Termination lifecycle Event Received? 108
Yes
Delete Dirty File 110
Terminate Process 112

Fig. 1: Use of a dirty file to detect crashes of third-party applications

Fig. 1 illustrates an example process for a provider of a code library to detect crashes of a third-party application that incorporates the code library by the use of a dirty file, e.g., a temporary file that is created on the device that executes the third-party application. At the start lifecycle event (102), e.g., a launch of a third-party application (or initial call to the library), code included in the library executes to determine whether a dirty file exists on the device that is running the application. Non-existence of the dirty file (“no” branch at 104) is an indicator of no prior crash of the application and a dirty file is created (106).

When a termination lifecycle event is detected (108), e.g., end of execution of the library code or termination of the third-party application, the dirty file is deleted (110) from the device prior to process termination (112). Until such time as termination is detected, no operations are performed related to the dirty file (116).

If a dirty file is detected at the start lifecycle event (“yes” branch at 104), it is an indication that a prior run of the application crashed while the library code was in use. Upon such detection, improper termination of the application is reported to the library provider and the dirty file is deleted (114). Note that the reporting of the crash to the library provider occurs at a subsequent execution of the code, e.g., the next time the application launches or executes the library code. Execution of the application then continues by creation of a new dirty file (106).

Third-party libraries are commonly used in many applications, e.g., mobile applications may use libraries for maps, database access, advertising, etc. It is important for the library provider to determine if a library causes crashes of an application that the library is compiled into. The described techniques provide a heuristic approach that allows detection of crash trends that are specific to the use of the library, given that other reasons for which an application may crash are not associated with the dirty file. For example, multiple concurrent (or near in time) reports of crashes of the same application, generated based on detection of the dirty file, sent to the library provider can indicate that there may be a problem.

CONCLUSION

This disclosure describes the use of a temporary dirty file to detect crashes that may be related to the use of an embedded library in an application. The dirty file is written to disk when the embedded code (library code) is first accessed or when the application is first launched. Upon successful completion of execution of the embedded code, the dirty file is automatically deleted. At a subsequent launch of the application or execution of the embedded code, if the dirty file still exists, it is determined that a crash of the application likely occurred since no application exit signal was received which would otherwise cause deletion of the dirty file.

REFERENCES

1. "A set of client and server components which implement a crash-reporting system" https://chromium.googlesource.com/breakpad/breakpad accessed 30 Oct 2020.