Abstract of the article: “Tolerating Hardware Device Failures in Software” written by A. Kadav, M. J. Renzelmann and M. M. Swift, published in the Symposium on Operating Systems Principles 2009

Louisa Bessad

24 juin 2014

1. The problem, its importance

Systems need to use drivers to be able to access devices. But when a device fails, this can cause driver to crash or hang. It has been demonstrated that most system failures are due to hardware device failures. Most of the time these failures are transient. By tolerating device failures drivers can prevent the system from crashing or hanging. In this way the reliability of the system can be improved.

Most approaches for research on drivers failures fall into two categories. The first is the static bug finding ; it analyzes the interface between the driver and the kernel to find incoherences. The second is the runtime fault tolerance. Most of the time these approaches need new operating sytem or new driver models ; it is not the case for static analysis. These approaches can also use memory detection to find hardware failures. But they do not check inputs from the driver. In addition, they find the failure but do not fix it. While they try to reduce faults on the interface between driver and device, these approaches need specific interfaces and languages.

2. The solution

Carburizer was created to insure ; input checking from the driver, a failure detection before corruption appears and a fix of device failures. Carburizer is an automatic code manipulation-tool combined with a runtime service. Its goal is to improve the reliability of the system. So it takes a driver, lists bugs that can occur in it and suggests fixes for these bugs according to their type.

To find bugs Carburizer scans the code to locate fragments that depend on input from the device. First, it consults tables of functions that are known to perform I/O and check the return value. Second, it searches for variables used as pointers or indexies of an array. In this way, it can check that a variable does not reference a forbidden memory area. Carburizer also looks for structures dependant on input from the device. With this research Carburizer avoids to have corrupted system structures. Third, it searches loops that wait for a specific hardware device state without timeout. Carburizer considers as timeout a variable incremented or decremented and used as exit condition on the loop. Then it searches for calls to panic and replaces such calls by a recovery function.

To fix the bugs that it finds, Carburizer provides several possible approaches. According to an kind of bug, it adds code to validate the unchecked data received from the device or a timeout to avoid infinite loop. If the timeout expires or the data received is not valid, a failure can occurs. To avoid a failure Carburizer creates a recovery function, and adds code to call this function where the bug can appear. Also, Carburizer verifies uses a runtime service to verify device responsiveness and make a report of all device failures. When Carburizer is not able to correct a bug, this report allows correction by programmer. Furthermore, Carburizer is able to find if a report for a specific bug is already existing on the system. This report could have been created by an other control mechanism. If the report already exists Carburizer does not create another report for the bug, it adds its own comments about a device failure to this report.

Carburizer provides an automatic recovery system : the runtime service. This service restores drivers and devices to a functioning state by using shadow drivers. They permit to save the state of the driver when a function is called between the driver and the kernel. In this way, if the driver needs to recover, the last state back-up will be used by Carburizer. In addition, the runtime service allows to manage the generation of interrupts. By generating too many interrups devices can hang because when the handler interrupt is running there is no useful work. On the other hand, devices that do not generate interrupts can became unstable or useless. Moreover the runtime service make it possible to find stuck interrupts. In this case the runtime service does not call the recovery function on the driver ; it will only disable the interrupt request line.

3. Results

Carburizer found 992 bugs in Linux drivers caused by hardware failures. Among these 992, it corrects 903 bugs by inserting code and using the runtime recovery service. Nevertheless 58 bugs are false positives ; the rate of false positive is 7.4%. The 89 others bugs require intervention from a programmer because they are due to dynamic arrays. Moreover Carburizer finds 2383 locations where drivers detect failures by timeouts, comparisons or range tests. In only 781 of the cases do the driver reports the error ; for the rest Carburizer adds error reporting code.

4. Point of view

We can say that Carburizer brings many improvements to handle device failures. This tool increases the number of failure logged, which makes easier the administrators' work. But this point shows that Carburizer is still not fully automatic. Moreover the number of machines that can be used for testing with Carburizer is limited because the device is needed. Thus, few platforms have been tested, only network drivers, sound drivers and drivers in the Linux 2.6.18.8 kernel distribution. More platforms should be tested.

Another negative point is that when a hardware failure is detected by the driver, Carburizer assumes that the driver will correctly respond to that failure. The system can also crash when a device receives data from the driver when the device is not ready. Carburyzer does not manage this bug.

In addition, Carburizer is limited in its handling of variables. When a variable is a pointer, an index of an array that is used several times, Carburizer checks the value of the pointer only the first time. Dynamic arrays bounds are unknown, so Carburizer can only report a bug. Moreover, when a variable depends on inputs from device it is notified as a possible bug, but if the variable is never used this notification is useless, amounting to a false positive.

About false positives, sometimes Carburizer does not detect a validity check whereas the code contains one. It occurs with infinite loop detection. Carburizer may not recognize a timeout inserted by the programmer if it is not a standard integer, and in this case adds a second timeout. There are also false positives when the array has exactly the same size that the index's range. But it does not create a false failure detection by calling recovery function for instance. Most of the false positives that are found on dynamic arrays are due to calculating the index by shifting or by operations other than mask or comparison. When the runtime recovery service repairs a false positive it has no impacts on the execution code, it just adds an overhead time execution.

The analysis can also suffer from false negatives. When variables depending on input from device are passed as an argument and not as a return value, Carburizer does not check them.

Finally, the driver may detect failures that are missed by Carburizer. This is due to the fact that reading operations can be wrapped in a single function which is not checked by Carburizer. And to the fact that Carburizer cannot find checking mechanisms implemented by the system when they are in a separate function. This lack of interprocedural analysis must be filled.

It is important to improve the ability of Carburizer to analyze separate function or variables called by the scanning code. The authors also proposed to report device with malfunctiononing, that does not lead to hang or crash system, not only device with failures.