Tolerating Hardware Device Failures in Software Resume

Louisa Bessad

5 juin 2014

Nowadays systems need to use drivers, which permit access to devices. But when a device fails, some drivers can crash or hang. In this way it was demonstrated that most of the systems failures are due to hardware devices failures, which most of the part are transient. Moreover drivers which tolerte device failures improve the reliability of the systems.

Two approaches have been used to study drivers failures, but they do not check inputs from the driver. The first is the static bug finding, it analyzes the interface between the driver and the kernel to avoid system crash. The second is the runtime fault tolerance.

Most of the time these approaches needed new operating sytem or new driver models. They also used memory detection to find hardware failures. In addition they find the failure but do not fix it. When they try to reduce faults on the interface between driver and device, a specific interface and language are needed. To insure inputs checking from the driver, have a failure detection before corruption appears and fix device failures, Carburizer was created.

Carburizer is an automatic code manipulation-tool in association with a runtime service.

Its goal is to improve the reliability of the system. So it takes a driver, lists bugs that can occur on it and suggests a driver without these bugs. To find bugs Carburizer scans the code to locate region depending on inputs from the device. Firstly it consults table of functions which are known to make I/O and evaluating the return value. Secondly it searches variables (used as pointer or index of an array) and structures dependant on input from the device. Thirstly it looks for loop which wait for a 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 panic calling and replaces it by a recovery function.

Most of the time Carburizer is able to fix these bugs. According to the bug it adds code to validate the unchecked data received from device or a timeout to avoid infinite loop. When it is needed, for instance with expiration timeout, Carburizer creates a recovery function to avoid the failure and adds code to call this function where the bug can occur. Also Carburizer verifies device responsiveness by runtime service and report all device failures. When Carburizer is not able to correct a bug this report allows correction by programmer. Furthermore Carburizer is able to find already existing report systems.

Carburizer provides an automatic recovery system : the runtime service. This service restores drivers and devices to functionning state by using shadow drivers. They permit to save the state of the driver when a fuction is called between the driver and the kernel.

In addition the runtime service allows to handle interrupts. By generating too many interrups devices can hang because when the handler interrupt is running there is no useful work. In another side devices that do not generate interrupts can became instable or useless.

Moreover the runtime service permits to find stuck interrupts. This issue does not lead the runtime service to restore the driver, it will only disable interrupt request line.

Carburizer found 992 bugs on Linux drivers caused by hardware failures. Over 992, it corrects 903 bugs by inserting code and using runtime recovery service. Nevertheless 58 bugs are false positives, the rate of false positive is 7.4%. The others involved an intervention from a programmer because these bugs are due to dynamic array.

About reporting failures, Carburizer finds 2383 locations where drivers detect failures by timeout, comparisons or range test. Only 781 errors are reported by drivers, for the rest Carburizer adds code to report the error.

We can say that Carburizer brings many ameliorations to handle device failures. This tool increases the number of failure logged. By increasing number of failure logged Carburizer allows to facilitate administrators' work. It was suggested to report device with malfunctioning not only device with failures. But this solution has some negative aspects : several points are not managed by this tool and it creates false positives and false negatives.

Firstly to control the hardware failures stress testing is not enough, we need fault-injection testing. Machines that can run these tests are limited because the presence of an instance of the device is needed. In this was few plateforms have been tested, there are only network drivers, sound drivers and drivers in the Linux 2.6.18.8 kernel distribution. More plateforms could be tested, moreover Carburizer is still not full automatic.

Moreover Carburizer assumes that when a hardware failure is detected by the driver, it will correctly respond to that failure. It does not handle crashes that occur when a device received a flag from the driver whereas it is not ready.

In addition Carburizer has its limits about the handle of the variables. When a variable is an index of an array which is used several times, Carburizer checks the value of the pointer only the first time. With dynamic arrays bounds are unknown, so Carburizer can only report the bug and tests if the pointer's value is not NULL. 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 and it creates a false positive.

About false positives, sometimes Carburizer does not detects validity check whereas the code contains one. It appends with infinite loop detection. Carburizer may not recognize a timeout inserted by the programmer if it is not a standard integer and adds one. There is also false positives when the array has exaclty the same size that the index's range. But it does not create a false failure detection by calling recovery function for isntance. Most of the false positives which are found on dynamic arrays are due to calculate index by shifting or others operations which are not mask or comparison. Repairing a false positives has no impacts on the execution code, it just adds an overhead time execution.

We also found 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 it appends that driver detects failures 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 system when the validation is implemented on a separerate function.

It is important to improve the ability of Carburizer to analyze separate function or variables called by the scanning code.