POCS OP6 : Self-recoverable software for zero-administration systems

Rémi Coudert

February 7, 2020

In a zero administration environment, software needs to be able to self-recover. This means that on software failure, it can reboot itself or recover from a previous state.

To this end we identify two important properties : self-recoverable software needs to run on top of a trusted kernel with an unfailable core and self-recoverable software should have a reboot recovery protocol using a state stored in the kernel. In the following, we assume that hardware is self-recoverable and that the kernel compiler and assembler are correct.

For a software to be able to self-recover, it first needs to trust the stack its based on to be self-recoverable as well or unfailable. With self-recoverable hardware, this means the software needs to run on top of an unfailable kernel. To make a kernel unfailable we need to prove its correctness. Klein et. al.[1] proposed a fully formally proven kernel, that is, we can predict the kernel behaviour in every possible situation, it will not perform unsafe operations and it will not crash.

The unfailable property of the kernel completes the trusted base stack on top of which the software will run. The kernel needs to maintain a table of running applications and, on software failure, the kernel should recover it by starting the application again. When an application terminates correctly, it can be removed from the table.

For the software to be self-recoverable, it needs to have a reboot protocol with state saving. On failure, the application will be rebooted by the kernel using this specific protocol defined by the application. The application should save a state within a specific defined kernel space periodically. The state should contain useful information for the application to reboot and recover from the crash, and hence depends on the type of application. This way, the application will use its saved state to recover important properties, such as ongoing TCP connections with remote nodes.

Since this state could potentially grow big, it should be restricted in size by the kernel depending on the application and carefully used by the software designer. In theory, the state should be the smallest possible set of values that permits to recover the most crucial parts of the software. Most of the recovery should be done by computation based on this state. For example, a web server recovering from a crash can self recover by fetching the list of its ongoing connections with TCP clients (their IPs) and the current sliding window states and buffer from the state. This is enough, as TCP packets that failed to reach the application during its down time will count as dropped and will be resent.

References

[1] Gerwin Klein et al. “seL4: Formal verification of an OS kernel”. In: *Proceedings of the ACM SIGOPS 22nd symposium on Operating systems principles*. ACM. 2009, pp. 207–220.