Zen and the Art of Software Debugging

Read About

Approaches to debugging

Three categories of software bugs

Tracepoints, sparklines, and watchpoints

Memory overwrites

The Age Old Debate: Art versus Science

There has always been a debate if software debugging is an art or a science.

Some engineers take the romantic approach to software integration: they write the code and debug by inspiration and intuition. Living in the moment, they forego rational analysis or repeatable processes. When confronted with an issue, they follow gut instincts and change their code often without even making a backup in case the assumption was wrong. They doggedly stack patches on top of patches, or introduce more problems when trying to back out the changes at a later date.

On the other side, some software developers follow the traditional approach: they try to diagnose and solve the problem by rigidly following a step-by-step debugging methodology. They become frustrated when the real world does not function the same as the world described in the programming books. Because of their dedication to ritual, technology transforms into magic and it becomes unpredictable and time consuming. They try the same techniques over and over again vainly hoping for a different result.

Perhaps the best programmer is the one that embraces debugging as both an art and a science. When debugging, bursts of creativity and intuition must work in harmony with rational problem-solving skills along with the best available tools.

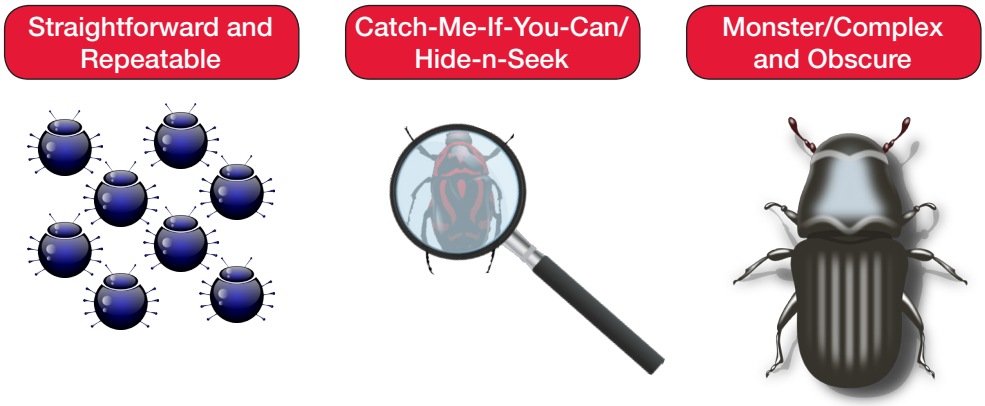

Figure 1: Three categories of software bugs

Info

curtisswrightds.com

Email

ds@curtisswright.com

The 3 Categories of Software Bugs

Most software bugs fall into three broad categories: The first is straightforward and repeatable. Due to its nature, it is the easiest to find and to fix. The next one plays catch-me-if-you-can, and hides when you try to trap it. For example, every developer has tried to debug a problem by putting in “just a couple of ‘print’ statements”, and amazingly the code starts working. Then, a few more ‘print’ statements are added and the problem re-emerges but dressed in totally different symptoms. The final bug is the monster that haunts our nightmares, the one of such complexity and obscurity that it appears to be truly random and not following any discernable pattern. It might only happen once a week after running mode A and then mode B for a thousand times in a complex sequence, under a full moon, when you go to get a drink and a cold piece of pizza. All these bugs create problems, whether they end in dismal failure, incorrect results, a hard crash, or the program lost in the weeds. To make matters worse, the error is usually not where the failure is observed; a large part of the detective work is tracing the error back to the root cause. Adding to the complexity, parallel programs’ bugs can and will propagate across multiple threads as well as multiple processors. If all of that is not enough, the bugs can also be timing dependent. As American writer and philosopher Robert M. Pirsig observed, “Some things you miss because they’re so tiny you overlook them. But some things you don’t see because they’re so huge.”

When debugging, some basic scientific tenants must be observed. One must always use a log book to track what has been tried. Nothing is more frustrating than seeing a similar problem to one that has already been fixed, but not remembering how it was solved. Keeping a paper logbook is a great first step, but then you have to keep track of the logbook itself!

Fixing Software with Software

Enabling the preservation of records of the scientific inquiry, Allinea DDT provides a digital logbook that automatically records the entire debugging session. For each stop in the program’s execution, the reason and location is recoded along with the parallel stacks, variables, and tracepoints which is a scalable ‘print’ alternative. The only exercise left for the user is recording the hypothesis, resulting observations, and concluding using the annotation option. The formation of the hypothesis is part of the art of debugging, because as Pirsig opined: “For every fact, there is an infinity of hypothesis.” Using proper tools like Allinea’s debugger, the theory can be checked by a rational process without changing the code.

Combining art and science for segment faults and more

Let’s examine the straight forward problems that happen repeatedly, such as segment faults, aborts, or an exit without an error code and the tool features that can facilitate solving these problems. These common bugs are easy to fix with a debugger, but much harder and time consuming without one. When first opening the code with Allinea DDT, common mistakes will be flagged by the static analysis tool. An example of the type of errors flagged is shown in Figure 2.

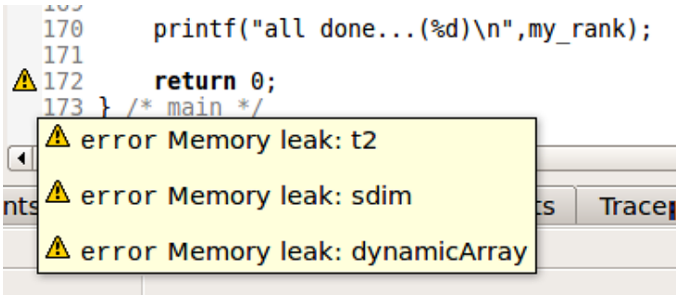

Figure 2: Potential errors are flagged with the static analysis tool

To become familiar with the tool, let's look at a simple program that aborts when it runs with input arguments. As seen in Figure 3, the debugger stopped on the line where the error was detected. The "Stacks" view pane shows the function call tree to the point where the code was stopped. Looking at the "Variable" pane reveals 'arg = 0', which is the problem. Most times, the answer is still not obvious and one might be tempted to sprinkle some 'print' statements, which would require the code be recompiled.

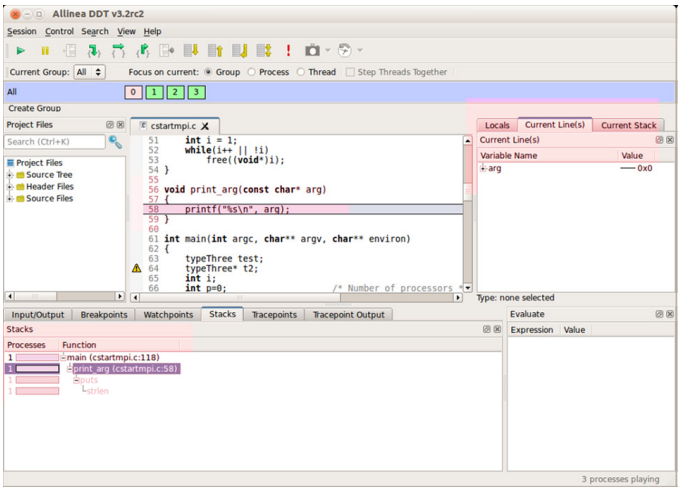

Figure 3: Debugger quickly identifies error in code

Tracepoints are a superior option because they track the lines of code executing and the variables of interest without stopping execution across multi-processes. Visual line indicators, called sparklines, quickly show the variation of the values across the processes and the range of the values as shown in Figure 4. For even more detail, the "Cross-Process and Cross-Thread Comparisons View" will present the data as a raw comparison, statistically or graphically.

One could explore arrays using the previously described tools, but that could be painful, slow and confusing for large arrays. Luckily, Allinea also has a tool for analysis of arrays complete with graphing and export capabilities.

Complimenting all the options the tool provides to analyze the data, flow control features such as breakpoints, watchpoints and step features add to the arsenal for proving hypotheses without ever having to change a line code. A watchpoint is a variable or expression monitored by the debugger, such that when it is changed or accessed, the debugger pauses the application. As demonstrated by just this quick look at some of the capabilities of Allinea DDT, the science is the tools while the art is the wielding of the tools.

"An experiment is never a failure solely because it fails to achieve predicted results. An experiment is a failure only when it also fails adequately to test the hypothesis in question, when the data it produces don't prove anything one way or another." – Pirsig

Remembering the tricks for memory errors

The next family of errors to be addressed are memory errors which can be very painful to track down. Depending on the input data, the problem might not be triggered or the results are incorrect with subtle, easy-to-miss differences. Overwriting or reading past memory boundaries can cause crashes or errors that don't occur until much later in the execution. Add to this the potential problems of deallocating memory multiple times, pointers to the wrong address and the all too common null pointers. The Allinea tool provides options to dial in the just the right amount of memory help. The more in-depth levels require more memory and time.

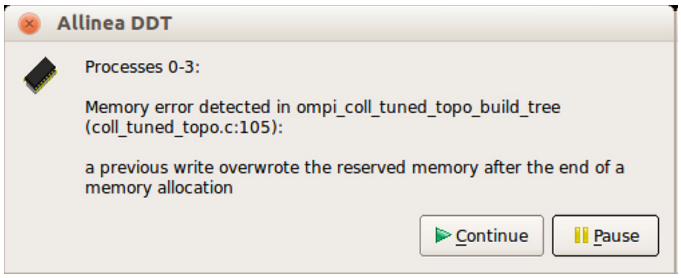

Figure 5: Allinea DTT program traps a memory overwrite

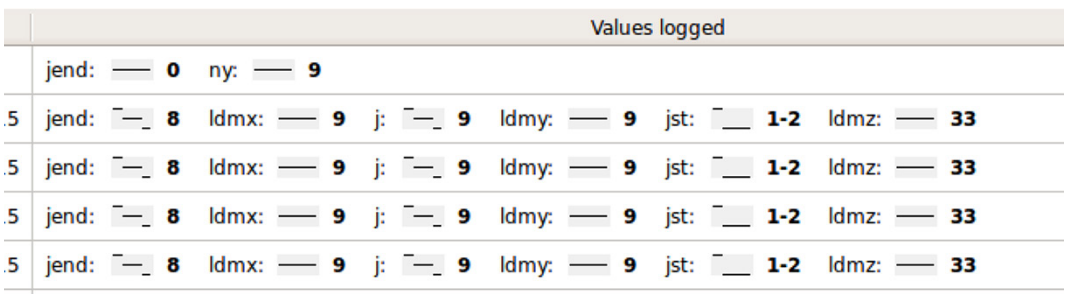

Figure 4: Sparklines show the variation of value

Figure 5 shows an example when a basic memory check traps a memory overwrite. By selecting the pause button, the debugger will stop with the offending line of code highlighted, then variables, pointer details, and expressions can all be evaluated as previously described. Given the art of the developer, these insights might be enough to reach a logical conclusion. If not, guard pages ride to the rescue to check for read and writes beyond an allocated block. The pages can be added before or after the block, and the default of a single page catches most errors. When memory usage is growing faster than expected due to a dreaded memory leak, the currently allocated memory for selected processes can be tracked with the "Current Memory Usage" View as shown in Figure 6.

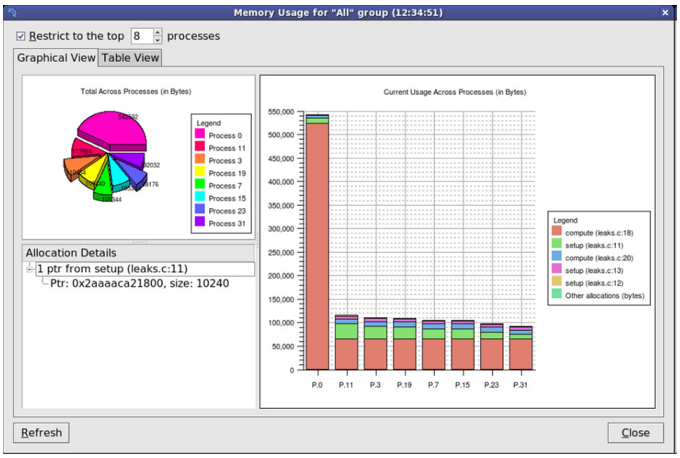

Figure 6: Track allocated memory with the "Current Memory Usage" view

The pie chart presents a quick overview of total memory allocated for each process. In the stacked bar chart, each bar represents a process and that bar is broken down into colored blocks for the contained functions. To drill down even further, clicking on a color block will provide detailed information about the memory allocations inside that function. Digging even deeper, the developer can enter the "Pointer Details View" to see exactly where the pointer was allocated, deallocated, and the amount of memory allocated. One might wonder why complex tools like this debugger are needed. Pirsig supplies the answer: "What makes this world so hard to see clearly is not its strangeness but its usualness. Familiarity can blind you too."

Solving the sleepless night dilemma

Let's shift to the chaotic non-repeatable bugs. These are the ones that lead to lost sleep, long weekends in the lab, and endless cups of cold coffee or warm Mountain Dew. The tracepoint tool can be configured to capture only unexpected values or to search for interesting patterns. Here, the art requires imagination – for example, use this feature to capture variables to determine why a function is called numerous times, but only crashes occasionally (when you walk away for a minute). In addition to the interactive mode described earlier, Allinea DDT also has an offline debugging mode where the debugger directs both the running of the code and the storing of results without user intervention. This allows the system to collect data overnight or however long it takes for the problem to occur. This offline mode supports a full set of tools such as tracepoints, memory debugging aids, and even breakpoints. In this mode, the debugger can also compile snapshots of the program state including the stacks and selected variables. The snapshots can be triggered periodically or by sending a signal from another terminal window to the DDT front-end process. The final report produced by the offline session consists of four sections; messages, tracepoints, memory leak status, and the output complete with timestamps.

Another tool that aids in the hunt for elusive bugs is checkpointing. A program's entire or partial subset can be stored in memory as a checkpoint for the duration of the debugging session. The application's state can then be restored from the saved checkpoint and execution will resume from the restore point. This is useful when you are unsure of what data will be needed to diagnose the problem until it is too late to retrieve it and it is difficult to get back to this point in the execution. For example, the program crashes because a variable has been set to an unexpected value and it is too late to set a watchpoint on that variable. If you had the artful insight to set a checkpoint earlier, the program can be restored to the checkpoint where the watchpoint can be set and the failure re-analyzed. When hunting for those elusive bugs, this quote from Prisig may resonate: "You look at where you're going and where you are and it never makes sense, but then you look back at where you've been and a pattern seems to emerge."

Authors

Tammy Carter, M. S.
Sr Product Manager
Curtiss-Wright Defense Solutions

Taking the wheel

This whirlwind tour of the art and science of debugging is intended to present a sampling of the powerful tools available in the Allinea debugger. These tools are expanding our capability to produce quality and more robust software with far less effort. According to Webster's dictionary, art is defined as "the systematic application of knowledge or skill in effecting a desired result." Webster then goes further by quoting J.F. Genung that "Science is systematized knowledge … Art is knowledge made efficient by skill". Finally, Robert M. Prisig chimes in with "We have artists with no scientific knowledge and scientists with no artistic knowledge and both with no spiritual sense of gravity at all, and the result is not just bad, it is ghastly." Whether debugging is an art or a science or a combination of both will continue to be debated, but all sides can agree the tools can make all the difference between timely success, and riding a motorcycle down the road of failure.

Learn More

White Paper: HPEC: High Availability by Design

Case Study: Avoiding Bottlenecks, Snail Threads, and Pitfalls in RADAR Software

Video: High Performance Embedded Computing (HPEC) and Complex Application Software

Products

- OpenHPEC Accelerator Suite
- Allinea DDT