Technical Whitepaper

Runtime and Memory Error Detection and Visualization with Parasoft Insure++

INTRODUCTION

Parasoft Insure++ has been around since 1993 and run through billions of lines of code. Because Insure++ has seen so much code, it is extremely valuable for detecting and preventing sets of errors. This paper is an important read before you begin using Insure++. It provides valuable insight into how Insure++ discovers and notifies you about:

- Errors during compilation
- Linking problems
- Errors during exercise of instrumented executables

AUTOMATING C/C++ RUNTIME ERROR DETECTION WITH INSURE++

C and C++ developers have a unique problem: many errors in their code don't manifest themselves during testing. Software with subtle problems such as memory corruption may run flawlessly on one machine, but crash on another. To help developers find and fix such problems prior to release, Parasoft designed Parasoft Insure++. Parasoft Insure++ is an automated runtime application testing tool that detects elusive errors such as memory corruption, memory leaks, memory allocation errors, variable initialization errors, variable definition conflicts, pointer errors, library errors, I/O errors, and logic errors.

With the click of a button or a simple command, Insure++ automatically uncovers the defects in your code— helping you to identify the source of that strange problem you've been trying to diagnose for weeks, as well as alerting you to problems that you were previously unaware of. Insure++ detects more errors than any other tool because its patented technologies achieve the deepest possible understanding of the code under test and expose even the most elusive problems.

WHAT PROBLEMS CAN INSURE++ FIND?

Insure++ automatically detects errors that might otherwise go unnoticed in normal testing. Subtle memory corruption errors and dynamic memory problems often don't crash the program or cause it to give incorrect answers until the program is delivered to customers and they run it on their systems... and then the problems start. Even if Insure++ doesn't find any problems in your programs, running it gives you the confidence that your program doesn't contain any errors. Of course, Insure++ can't possibly check everything that your program does. However, its checking is extensive and covers every class of programming error, including:

PARASOFT®
Automated Software Testing

- Memory corruption due to reading or writing beyond the valid areas of global, local, shared, and dynamically allocated objects.
- Operations on uninitialized, NULL, or “wild” pointers.
- Memory leaks.
- Errors allocating and freeing dynamic memory.
- String manipulation errors.
- Operations on pointers to unrelated data blocks.
- Invalid pointer operations.
- Incompatible variable declarations.
- Mismatched variable types in printf and scanf argument lists.

DISCOVERING MEMORY USAGE PROBLEMS

Many modern algorithms make heavy use of dynamic memory, but few take any great precautions to ensure that they achieve the best possible use of the memory system. As a result, many applications can benefit from streamlining their memory usage or modifying the order of their allocation requests to reduce the fragmentation that takes place when memory is allocated and freed.

Many algorithms also contain subtle memory leaks in which the program consumes growing amounts of memory as it runs until the resources of the host are finally exhausted and the program crashes—sometimes days or weeks after starting.

Parasoft Insure++ offers a graphical memory display/animation tool. It is designed to help developers avoid memory problems by displaying and animating the memory allocations performed by an application. By watching your program allocate and free dynamic memory blocks, you gain a better understanding of the memory usage patterns of your algorithms and also an idea of how to optimize their behavior. Parasoft Insure++ allows you to:

- Look for memory leaks.
- See how much memory your application uses in response to particular user events.
- Check on the overall memory usage of your application to see if it matches your expectations.
- Look for memory fragmentation to see if different allocation strategies might improve performance.
- Analyze memory usage by function, call stack, and block size.

AVOIDING MEMORY PROBLEMS

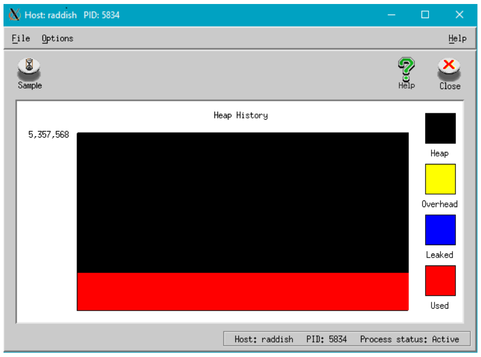
Parasoft Insure++ can help you detect and avoid more than just memory leaks. It can show you most common memory problems, including memory blowout, memory fragmentation, memory overuse, and memory bottlenecks.

Memory Blowout

Many programmers are not aware of the danger memory blowout poses for commercial and industrial applications. Memory blowout is very common because the operating system allocates pages of memory to the program and never releases them. Other programs running on this machine will then be starved for memory and crash.

During memory blowout, a program allocates a large chunk of memory for use and frees it after it is finished with the memory. At that moment, the freed memory can be reused by the program, but it is not released to the operating system. The reason for this is that when memory was allocated by the program, the operating system allocated all the needed memory pages. The operating system will not release the memory pages, even after they are freed by the program, until the program is exited. As a result, other programs running on the machine needing memory will ultimately crash.

Parasoft Insure++ can help you prevent memory blowouts. Figure 1 shows the Heap History report generated for a program showing memory blowout. This problem can be easily identified by the large size of the heap and the small amount of allocated memory. This shows up on the display as a low ratio of allocated memory to heap space. Without Parasoft Insure++, it would be difficult--if not impossible--for most developers to detect this impending catastrophe.



Host: raddish PID: 5834 Process status: Active

Figure 1. *Memory Blowout*

MEMORY FRAGMENTATION

Most programs that use a lot of dynamic memory are at risk of memory fragmentation. Memory fragmentation can be caused by the overuse of memory, which slows down memory allocation. For example, when a program allocates and frees small and large memory blocks interchangeably, the programmer will expect the total amount of allocated memory to remain constant. However, memory allocation routines cannot fit blocks of memory in freed spaces that are the wrong size. This will lead the program to request new pages for memory from the operating system. As a result, the program will consume more memory and ultimately run out of dynamic memory without leaking a single byte.

Parasoft Insure++ Heap Layout report can help you monitor memory fragmentation (See Figure 2), which immediately presents the layout of dynamically allocated blocks and the free spaces in between them. By simply clicking on a block, the status of the block, its memory address, its size, and the stack trace where it was allocated are displayed.

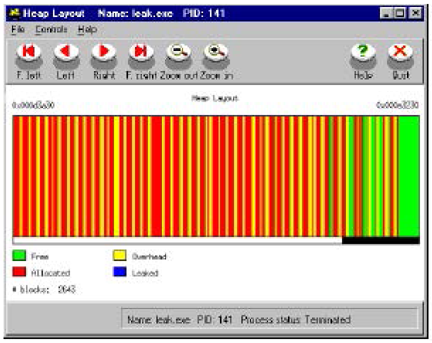


Figure 2. *Monitoring Fragmented Memory*

MEMORY OVERUSE (HOGGING)

Memory overuse or hogging occurs when memory is allocated by a program and never freed. Memory is not leaked during overuse, though, because pointers remain in the program—the memory can be freed, the program just doesn't free it. As a result, the program uses more and more memory until it runs out and crashes.

Memory overuse can be observed in the Heap History graph. If the amount of leaked memory is negligible and stays constant, or Insure++ does not report any leaks but the Heap History exhibits a stairway pattern (as in Figure 3), the program is hogging memory and eventually will run out. In these cases, the programmer should use the Query report to look for totals of memory allocated for different stack traces. If a stack trace shows the total amount of allocated memory growing, it likely contains the offending line of the source code (see Figure 4). At that moment, the programmer should analyze the code and verify that the program needs to allocate memory at that stage and needs to keep it. If it doesn't, this is an algorithmic problem and needs to be fixed.

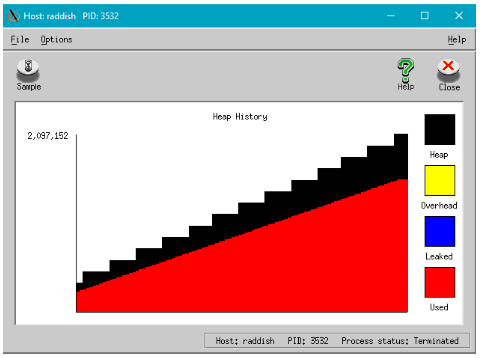
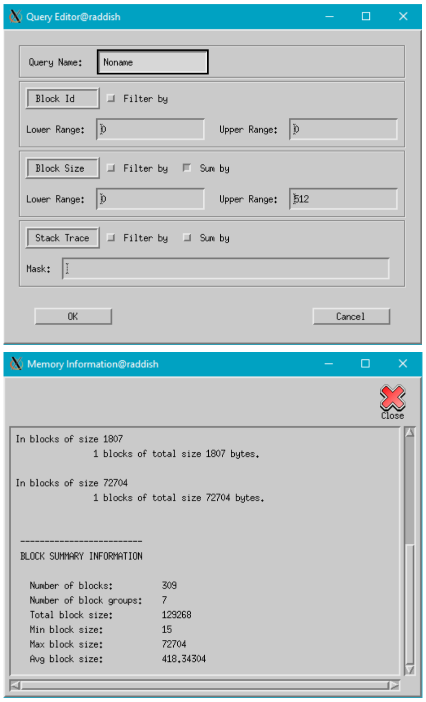


Figure 3. *Memory Hogging*



Query Name: Noname

Block Id □ Filter by
Lower Range: 0 Upper Range: 0

Block Size □ Filter by ☑ Sum by
Lower Range: 0 Upper Range: 512

Stack Trace □ Filter by □ Sum by
Mask:

```
In blocks of size 1807
            1 blocks of total size 1807 bytes.

In blocks of size 72704
            1 blocks of total size 72704 bytes.


 ------------------------
 BLOCK SUMMARY INFORMATION

   Number of blocks:         309
   Number of block groups:   7
   Total block size:         129268
   Min block size:           15
   Max block size:           72704
   Avg block size:           418.34304
```

Figure 4. *Analyzing Memory with Query*

MEMORY BOTTLENECKS

Bottlenecks occur when an operating system spends more time paging memory than running the program. Memory bottlenecks frequently arise when a program uses large amounts of dynamic memory or calls large amounts from different parts of the program. They are particularly problematic because a program may perform well in-house but stop dead in the field.

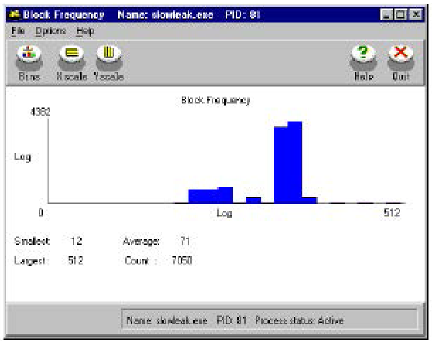
Paging can significantly slow down execution of the program and effectively prevent it from running. A program may perform in-house without having to page, but this can change when different conditions are introduced. A customer might have less memory or be crunching much more data than the program was tested with. As a result, the program uses a larger portion of memory and forces the operating system to page. The problem is compounded if the program accesses a lot of memory from different locations.

To avoid this problem, the program should be analyzed with respect to how much memory different parts of the program need and if that need is warranted. Parasoft Insure++ enables you to calculate how much memory is allocated by a specific path, routine, or block type. This type of analysis is critical for understanding algorithmic problems and can make it much easier to improve memory performance.

REVEAL THE “TRUTH” ABOUT MEMORY USE

When developers write their programs, they usually have some idea how memory should be allocated by the program. This idea is typically far from what the program is actually doing. Parasoft Insure++ lets developers see the “truth” about their programs by visualizing memory allocation at runtime.

Developers can single-step the program by setting breakpoints at calls to malloc, free, new, delete, realloc and then watch allocations as they happen. This works as a confirmation of an algorithm’s performance and shows whether or not the program works correctly. Single-stepping should be done just after a program is first built. By checking memory usage early, you can save a lot of headaches down the road. Also, you can spot significant algorithmic problems. It is particularly useful at this stage to look at the Block Frequency graph of Parasoft Insure++ (see Figure 5). This graph will show the distribution of memory blocks.


Block Frequency Name: slowleak.exe PID: 81

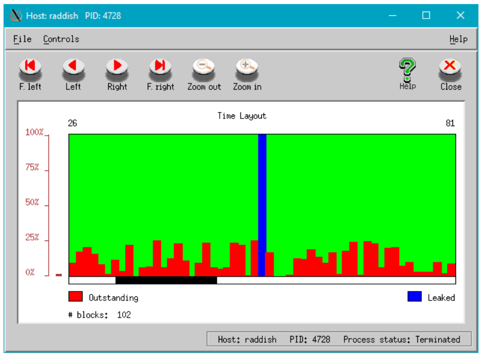
Smallest: 12 Average: 71
Largest: 512 Count : 7050

Name: slowleak.exe PID: 81 Process status: Active

Figure 5. *Viewing Block Frequency*

CLEANING UP LEAKS

Not all leaks in a program have to be cleaned. Leaks are only deadly if they reoccur constantly. Leaks that only occur at the beginning of a program can be ignored in many cases. But, leaks that occur during the execution of a program--especially leaks in parts containing loops--are deadly and need to be fixed. The severity of a leak can be easily determined using the Time Layout display (See Figure 6). This diagram shows blocks as they are allocated in time sequence. All leaks concentrated at the initial stage of program can be safely ignored. On the Time Layout display, clicking on a block will show its size, address, and stack trace where it was allocated.


Host: raddish PID: 4728

# blocks: 102

Host: raddish PID: 4728 Process status: Terminated

Figure 6. *Checking the Timing of a Leak*

ABOUT PARASOFT

Parasoft helps organizations perfect today’s highly-connected applications by automating time-consuming testing tasks and providing management with intelligent analytics necessary to focus on what matters. Parasoft’s technologies reduce the time, effort, and cost of delivering secure, reliable, and compliant software, by integrating static and runtime analysis; unit, functional, and API testing; and service virtualization. With developer testing tools, manager reporting/analytics, and executive dashboarding, Parasoft supports software organizations with the innovative tools they need to successfully develop and deploy applications in the embedded, enterprise, and IoT markets, all while enabling today’s most strategic development initiatives — agile, continuous testing, DevOps, and security.

www.parasoft.com

Parasoft Headquarters:
+1-626-256-3680

Parasoft EMEA:
+31-70-3922000

Parasoft APAC:
+65-6338-3628

PARASOFT®
Automated Software Testing