

Rational PurifyPlus for AIX

AIX平台上的代码动态分析工具, 帮助快速开发高质量C/C++代码, 充分发挥硬件能力。



Rational PurifyPlus产品有三个组件: 内存错误检测的Purify, 代码覆盖分析的PureCoverag以及性能数据分析的Quantify。开发人员在开发过程中使用PurifyPlus, 可尽早发现和定位软件的缺陷。测试人员在测试中是使用PurifyPlus, 可以科学评价软件质量。

内存错误检测工具: Purify

Purify组件是最全面的动态分析工具, 可以检测软件的所有程序, 包括系统库, 第三方库以及自己的C/C++代码, 并支持多进程、多线程以及服务程序的检测。Purify组件通过检测所有的内

存访问操作, 发现内存访问错误的位置, 并提供详细信息帮助定位错误的引入位置。Purify支持的应用内存相关错误包括:

- 数组越界写和读
- 未初始化内存读
- 访问已释放的内存
- 栈指针越界读和写
- 利用空指针进行读和写
- 内存泄露和文件描述符未释放

Click to expand the ABR message

The function call chain indicates an error occurring in _doprint called by printf, in turn called on line 22 of main

The exact location of the error

The details of the access error

The allocation call chain shows that the memory block is allocated in the function main on line 19

```

Purify: a.out
File View Actions Options Help
Finished a.out ( 1 error, 12 leaked bytes)
Purify instrumented a.out (pid 8701 at Fri Aug 8 16:22:57 2003)
ABR: Array bounds read
This is occurring while in:
_doprint [libc.so.1.8]
printf [libc.so.1.8]
main [hello_world.c:22]
#include <stdio.h>
#include <malloc.h>
static char #helloWorld = 'Hello, World';
main()
{
char *ngstr = malloc(strlen(helloWorld));
strcpy(ngstr, helloWorld);
printf("%s\n", ngstr);
}
start [crto.o]
Reading 1 byte from 0x4423c in the heap.
Address 0x4423c is 1 byte past end of a malloc'd block at 0x44230 of 12 bytes
This block was allocated from:
malloc [rtlib.o]
main [hello_world.c:19]
start [crto.o]
Current file descriptors in use: 5
Memory leaked: 12 bytes (100%); potentially leaked: 0 bytes (0%)
Program exited with status code 1.
  
```



通过Purify, 你可以在开发过程中就形成高质量的代码, 而不必在开发后期消耗大量时间通过调试去定位软件缺陷。

代码覆盖分析工具: PureCoverage

在开发过程中, 软件代码经常变化。不幸的是, 测试用例经常不能保持更新。PureCoverage是一个简单易用的工具, 可以帮助明确没被测试的C/C++代码行和函数。

- PureCoverage支持的功能包括:
- 指出未被测试的代码
- 多次执行的测试覆盖数据能被累加
- 和Purify集成, 确保Purify对所有的应用代码进行了内存错误检测
- 自动形成各种已定义格式报告和自定义格式报告

The Calls column shows how many times the program called each function

The FUNCTIONS columns tell at a glance whether each function was used or unused

Adjusted unused lines	Runs	Calls	FUNCTIONS		ADJUSTED LINES		ADJS	total
			unused	used	used%	used%		
1	1	1	1	2	66%	3	6	66%
1	1	1	1	2	66%	3	6	66%
1	1	1	1	2	66%	3	6	66%
0	0	0	used			2	0	0%
1	1	1	used			1	4	80%
1	1	1	used			0	2	100%

Function-level information includes the number of times the program called each function

Number of times each line was executed

Adjustments

Source code

Source code line numbers

Unused code

Unused code

```

16 void display_message()
17
18 main(argc, argv)
19 int argc;
20 char *argv;
21 C
22 1
23 if (argc == 1)
24     display_hello_world();
25 else
26     display_message(argc);
27 exit(0);
28
29
30 void display_hello_world()
31 C
32 1
33     printf("Hello, World\n");
34
35
36 void display_message()
37 char *s;
38 C
39 1
40     printf("In, World\n", s);
  
```



PureCoverage提供的代码覆盖信息可快速发现未被测试的代码, 指导不断完善测试用例, 从而提高测试完备性。

性能数据分析工具: Quantify

软件应用的运行性能是一个关键的质量属性, 并可帮助节省硬件的计算能力。但开发满足客户期望的高性能软件并不容易。软件应用运行时, 涉及自己开发的C/C++代码、第三方库、操作系统、硬件、网络等, 其复杂性导致定位性能瓶颈比较困难。使用

Quantify, 可获得代码的详细性能信息, 帮助定位性能瓶颈, 从而帮助改善应用执行性能。

Quantify具有如下特点:

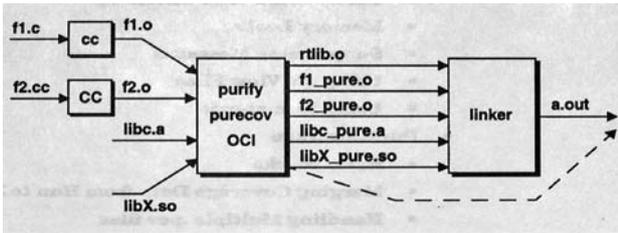
- 获得准确、可靠的性能数据
- 控制性能数据的收集, 收集范围可是一部分应用程序或者所有应用程序。
- 比较多次运行后所得的性能数据之间的差异
- 易于定位和解决最关键的性能问题

The screenshot displays the Quantify application interface with several overlapping windows:

- CONTROL PANEL**: A small window at the top left with buttons for 'Function List', 'Call Graph', 'Help', and 'Exit'. It shows the current target as 'hello_world (pid 29302)'.
- FUNCTION LIST**: A window on the left showing a list of 37 functions. The 'main' function is highlighted, with its execution time (0.39) and other metrics.
- FUNCTION DETAIL**: A window in the center showing detailed performance data for the 'main' function, including file name, call count, and distribution to callers.
- ANNOTATED SOURCE**: A window on the right showing the source code with performance annotations for the 'main' function.
- CALL GRAPH**: A window at the bottom right showing a call graph for the 'main' function, illustrating the flow of control between different parts of the program.

易于使用, 可快速集成到Makefile

PurifyPlus为了获得运行时分析数据, 采取目标代码插针 (Object Code Instrument) 技术, 也就是在目标代码中插入一些特定的代码, 其原理如图:



在上图中, 链接器形成执行程序a.out之前, PurifyPlus对相关的目标文件 (f1.o, f2.o) 以及相关的程序库 (如libc.a以及libX.so) 进行插针。通过执行插针后的可执行程序, 就能获得内存访问数据、代码覆盖数据以及性能数据。

对目标代码进行插针的命令采用如下格式:

```
purify[ <purify-options> xlc -g f1.o f2.o
```

其中purify表示为获得内存分析数据进行插针; 如需获得代码覆盖数据, 对应的命令是purecov, 如需获得性能分析数据, 对应的命令是quantify。

如通过makefile来实现, 其格式为:

```
a.out: f1.o, f2.o
```

```
purify[ <purify-options> xlc -g f1.o f2.o -o a.out
```

PurifyPlus支持的平台信息

PurifyPlus支持如下AIX操作系统和XL C/C++版本。

操作系统	编译器	硬件
AIX 6.1	IBM XL C/C++ 10.1	IBM POWER4
AIX 5L v5.3	IBM XL C/C++ 9.0	IBM POWER5
	IBM XL C/C++ 8.0	IBM POWER6
	IBM XL C/C++ 7.0	IBM POWER6

