调试器的断点机制

齐尧
qiyaoltc@gmail.com

HelloGCC 2010
2010-11-06

1. 目录及流程
2. 关于我和我的话题
3. 最简单的调试器
   - 最简单的调试器是什么样子的
   - 和标准调试器的比较
4. 断点机制细节
   - 断点
   - 断点在其它部分的应用
5. *The End*

关于我和我的话题

我

- 齐尧 qiyaoltc@gmail.com
- 曾经在frysk项目为PowerPC64维护断点和底层事件循环
- 中间中断了三年的toolchain开发，现在又开始继续toolchain的工作

我的话题

- 它不可能让你在三十分钟内，让你明白调试器的内部，
- 甚至不可能让你明白调式器的一个组件或者模块，
- 它能让你从最简单的角度，理解调试器的必要功能，
- 掌握学习调试器的方法

调试器

- 父进程（调试器） 与 子进程（被调试程序）
- 通过操作系统提供的接口，

功能	操作	实现
控制被调试程序	`contiue`,	ptrace, signal
监视被调试程序	`break, watch`	breakpoint epilogue detection
检查被调试程序	`bt`	stack unwind parse prologue
源代码级别调试	associate with source	dwarf调试信息

最简单的调试器是什么样子的

```
1   pid_t child = fork();
2
3   if(child == 0){
4     ptrace(PTRACE_TRACEME, 0, NULL, NULL);
5     execl("./hellodebugger", NULL);
6   }else{
7     int status; long pc;
8     while(1){
9       wait(&status);
10
11      if (WIFEXITED(status)){
12        printf("Exit status:%d\n",WEXITSTATUS(status));
13        return 0;
14      }
15      if (WIFSTOPPED(status)){
16        printf("STOPPED by %d, ", WSTOPSIG(status));
17      }
18      if (WIFSIGNALED(status)){
19        printf("SIGNALED by %d, ", WTERMSIG(status));
20      }
21
22      pc = ptrace(PTRACE_PEEKUSER, child, 4 * EIP, NULL);
23      printf ("at 0x%lx\n", pc);
24      ptrace(PTRACE_CONT,child, NULL, NULL);
25    }
26
27  }
```

Listing 1: 最简单的调试器

和标准调试器的比较

断点

- ptrace：父进程可以通过这个接口对子进程的内存空间和寄存器进行修改
- signal：父进程可以知道子进程受到的信号

和标准的调试器比较

- breakpoint
- ELF parsing
- Frame stack unwind
- Shared object
- Thread

断点到底是什么?

断点

- 与指令集密切相关的特殊指令或者非法指令
- 当这个指令被执行的时候，导致操作系统 trap，调试器就被通知
- 调试器通过分析，知道程序执行到了一个断点

更多的问题

- 编译器一般不会产生这样的指令，谁加入的? 怎么加入的? 原来的指令怎么办?
- 当程序执行到某个断点指令，产生trap时候，调试器怎么知道，这个是个断点? 到底是执行到了哪个断点，
- 调试器把断点指令加入的程序中，原来的程序还执行吗? 怎么执行?

断点

```
1  int
2  main()
3  {
4    asm ("int3");
5    return 0;
6  }
```

Listing 2: 一个带断点的程序

```
1  STOPPED by 5, at 0x40000850
2  STOPPED by 5, at 0x80483b8
3  Exit status:0
```

Listing 3: 简单调试器的输出

理解结果

- 为什么收到SIGTRAP两次?
- 0x40000850 和 0x80483b8 分别是什么地址?

断点

0x40000850 和 *0x80483b8* 分别是什么地址?

```
1  080483b4 <main>:
2   80483b4:        55                          push   %ebp
3   80483b5:        89 e5                       mov    %esp,%ebp
4   80483b7:        cc                          int3
5   80483b8:        b8 00 00 00 00              mov    $0x0,%eax
```

Listing 4: main()函数的汇编

- 0x80483b8是断点后的那条指令 （pc指向要执行的那条指令）
- int3 指令是一个字节，为什么?

断点操作

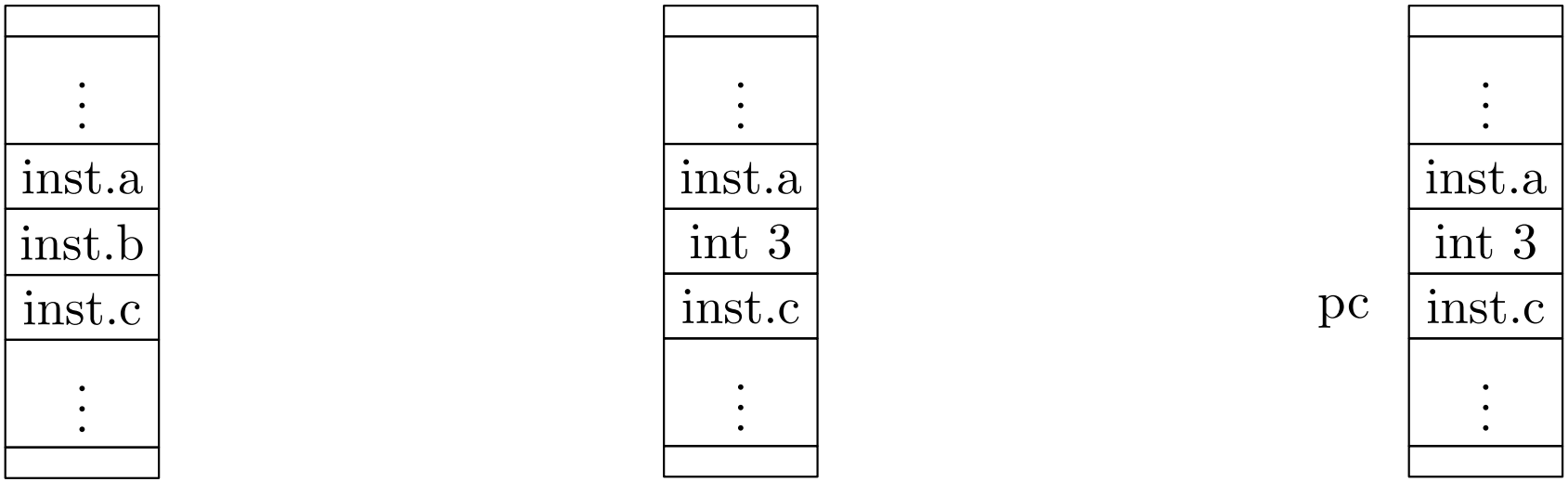

Figure: 调试器设置断点，程序碰到断点

断点操作

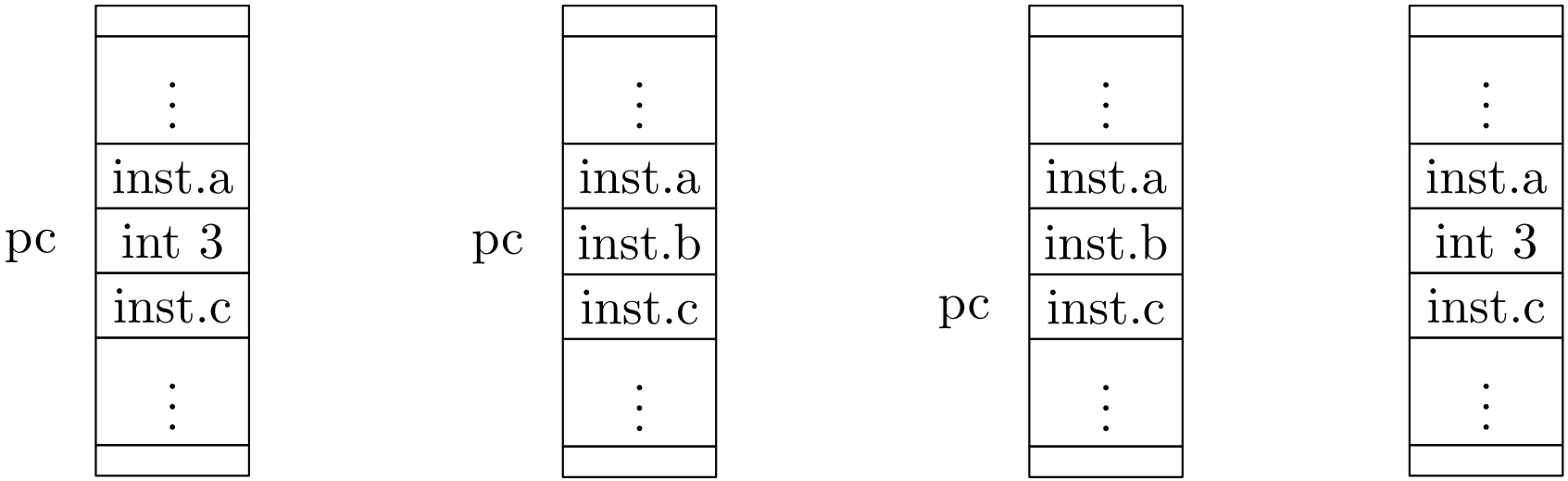

Figure: 调试器让程序继续执行，恢复断点

断点在其它部分的应用

- 软件单步:把断点设置在下一条指令，模拟单步执行。
- 共享对象加载:程序动态加载一个库时，调试器需要知道这个库的信息。
- displaced stepping:上一页的程序resume过程，并不高效。
- .....

The End