# 内核源代码差异分析与展示

------阅读报告

**论文作者：颜世勋 时间：2015年5月**

论文提出函数差异图这一概念，并开发了生成函数差异图的工具Diffe ,又提出语法差异分析。

DIFFE的基础是源代码文本差异分析，源代码差异提取是基于diff，然后对提取的差异结果进行分析归类，使用由内核符号表以及反汇编，生成的函数定义列表将差异定位到函数中。再通过函数调用图生成工具，生成函数调用矩阵graph。合并两个版本的函数调用矩阵，在节点与边上加入差异内容。最后通过SVG图将差异可视化。

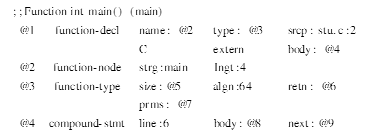
Diffe的语法差异比较则是工具的一个创新点。为实现将宏的值与条件编译内容的改变体现在函数差异内，通过对GCC编译中间语法树的分析，把语法树以一个容易利用的形式，即控制流图（CFG）的形式进行比较，分析出源代码函数体中的语法差异。再将这部分内容与源代码文本差异分析结合，得到更全面的差异内容。

DIFFE依赖函数分析平台DCG-RTL。差异的提取和定位需要利用DCG-RTL所提供的源代码支持和函数定义与调用的数据库表支持，而最后的展示部分也需要利用平台提供的web页面进行版本的选择以及函数差异图的显示。

项目平台所依赖的内核分析平台是以源码阅读工具Lxr与函数调用图在线展示工具DCG-RTL结合开发的。这个平台通过Lxr对内核源码进行管理，设置不同的选项，可以生成不同版本、不同平台的模块/函数的调用关系图。

以函数为粒度的差异比较工具

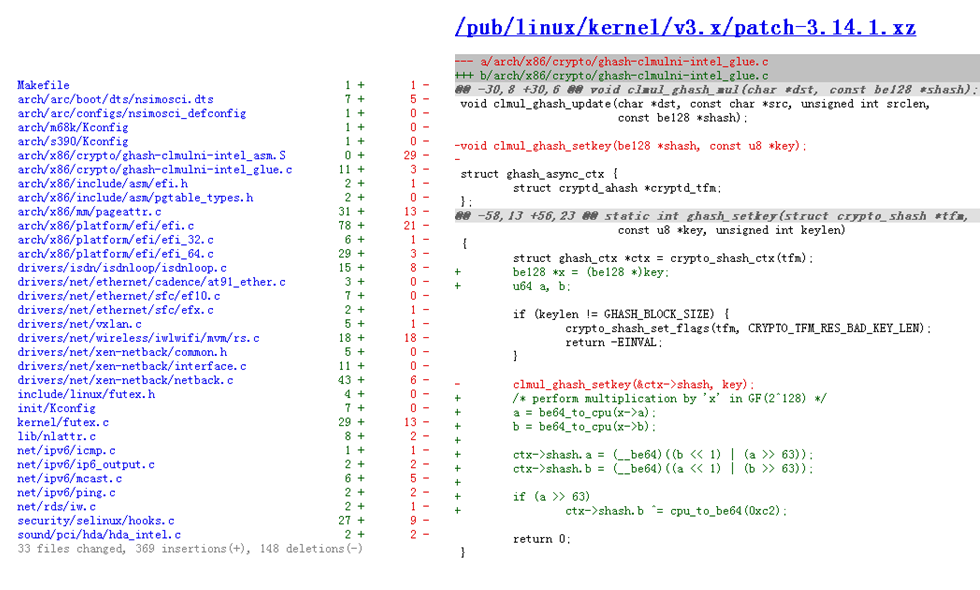
作者开发的工具diffe基于Linux所使用的diff工具，可以先生成DIFF文件作为中间结果，再用DIFF2HTML来转化成HTML。diffe使用GCC（通过和Clang作对比，选择了GCC）产生抽象语法树（抽象语法树，是用树状结构来对代码的语法信息进行表示，又叫AST。），GCC的其中一种抽象语法树结构如下图。



从上图可以看出，每个节点是以“@+数字”标出，左边显示的是节点的名称，也叫节点标识符，右边的是节点包含的信息及其中包含的其他节点。当然，这并不是唯一的展现结果，在编译之前可以加入控制参数，以需要的显示形式生成结果。

diffe工具的目标是扩展跟踪与分析平台的内核更新差异分析和展现能力，结合多版本静态函数调用关系分析和diff工具的差异信息，给出内核更新的源代码差异，展现多层次的可视化的内核更新差异视图，从直观上让人更容易获得有用的差异信息。

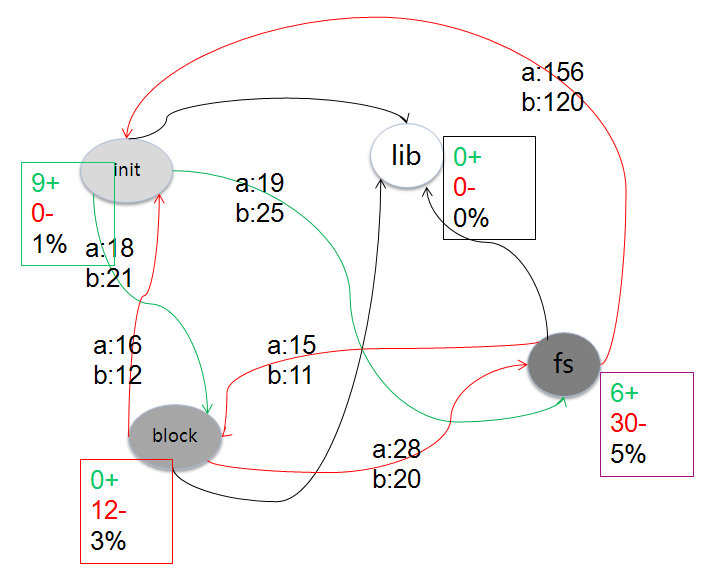
内核版本差异展现方式：更新函数在内核各模块中的分布情况；在函数调用图中模块顶点标记更新函数数目；模块间边上函数调用中的更新函数分析情况；在函数调用列表中标记更新函数。Diff结果示例：



**图3.预期函数对比模式**

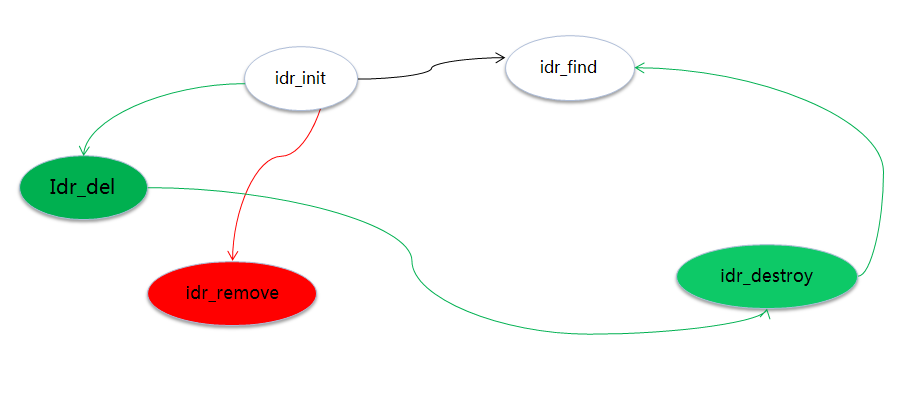
比较模式分为两种：

模块/文件级：版本升级一般是新版本和旧版本比较，按照传统diff使用习惯，红色表示新版本较旧版本删除了内容，绿色表示增加了内容。在模块级别，红色和绿色的顶点分别表示完全删除和增加的模块（这种情况应该比较少出现，更多的是后面以灰度来表示的代码发生变化的模块），以变化的代码行数百分比来设定模块节点的灰度，红色和绿色有向边表示调用次数减少或者增加，图层和数据可以设置开关菜单项。边权表示两个版本下的模块间函数调用次数，A表示旧版本的函数静态调用次数，B表示新版本中的静态函数调用次数。顶点颜色是用灰度表示的模块源代码变化程度。鼠标所放在某个顶点时浮出框展现源代码变化数据，分别是删除源代码行数（红色）、新增源代码行数（绿色）和变化代码行数的比例（黑色）。



**图4.函数差异图设计方案**

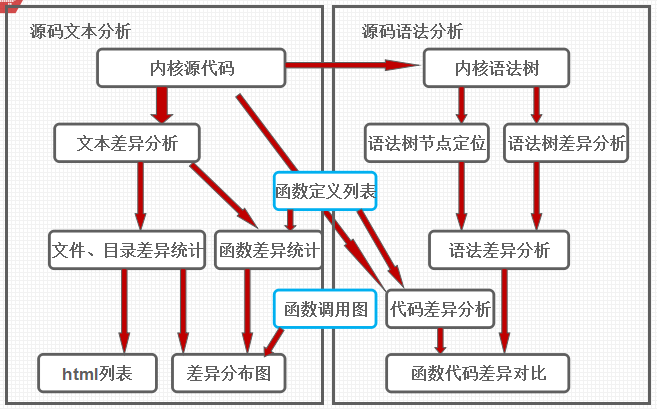
函数级：同上，绿色为新版本中增加的函数，红色为新版本中已经删除的函数，调用也按相应颜色标识。同样需要设置开关菜单项进行按需裁剪边和节点（如隐藏绿色边和节点则为旧版本局部函数调用图，隐藏红色则为新版本局部函数调用图）。



**图5.函数差异图调用设计**

函数调用图之后可附上函数定义列表和函数调用的详细列表，表中字段可指向源码文件、源码diff、两个版本的注释等。最后将生成的结果，与分析平台整合，结果嵌入到网页中，生成路径固定在lxr分析平台目录下。主要的研究内容包括：宏定义的内容的差异，如何结合源码diff内容展示出来；diff出的差异文件的展示：与SVG和DIFF2HTML的展现；模块的依赖函数调用关系的合并计算；旧版本代码进行新版本系统开发时，能够主动获得相关差异的提示；差异展示与分析平台的集成；对系统的实现与测试。

**工具框架结构：**



**图6.Diffe工具设计框架**

Diffe工具集成于DCG-RTL工具之中，以一个web页面体现。在该页面中填入与生成函数差异图或函数代码/目录差异表相关的参数。同其他功能一样，Diffe工具将这些参数组成url连接，传递给位于服务器的处理脚本，进行解析调用。Diffe的两个子功能，通过脚本实现生成函数差异图功能，以SVG图（可缩放矢量图形）的形式展现；以及直接生成函数代码/目录差异表，以html的形式展现。通过函数差异图上的节点连接，能跳转查看函数调用列表、函数代码/目录差异表以及下层目录的函数调用图。最终生成的图和页面文件，将被Diffe调用，嵌入到网页中下方的空白区域，用以展示。

**Diffe工具实现源代码差异分析**

**1、函数差异分析（文本上的差异分析）**

（1）函数定义列表。

包含函数的信息有：函数所处的文件路径、函数名列表、函数起始行号和函数结束行号。获取到改动的位置时，去与数据库中的函数起始行号和函数结束行号比较，如果改动位置在函数范围之内，则可以确定该函数是被改动的，改动的程度就是获取到这个函数改动的多少。

函数定义列表存储在数据库中。利用之前工作（？）db-rtl-callgraph的自动部署函数调用图相关脚本，将编译好的内核函数信息按特定格式导入到数据库中

（2）利用diff获取差异信息

使用diff的“-u”参数，选定上下文行数为3的差异格式，获取最初的差异文件。

（3）分析差异文件，得到差异信息

【Diff文件的格式分三部分，第一部分是两个相互比较的文件路径，用diff及其参数标识；第二部分是改动的起始行号和范围，包括之前设定的上下文，用@@符号标识；第三部分是具体的改动。为了方便的获取查询数据库的关键字，我们需要对得到的结果进行处理，增减的行数分别用“+”、“-”来标识出来，没有这两种标识的语句就是没有发生变化。下列代码是一个两个版本同一函数的diff文件内容。

--- kmem\_cache\_alloc\_linux-3.8.13 2015-03-10 14:09:57.834411155 +0800

+++ kmem\_cache\_alloc\_linux-3.5.4 2015-03-10 14:09:57.851411083 +0800

@@ -1,8 +1,8 @@

void \*kmem\_cache\_alloc(struct kmem\_cache \*s, gfp\_t gfpflags)

{

- void \*ret = slab\_alloc(s, gfpflags, \_RET\_IP\_);

+ void \*ret = slab\_alloc(s, gfpflags, NUMA\_NO\_NODE, \_RET\_IP\_);

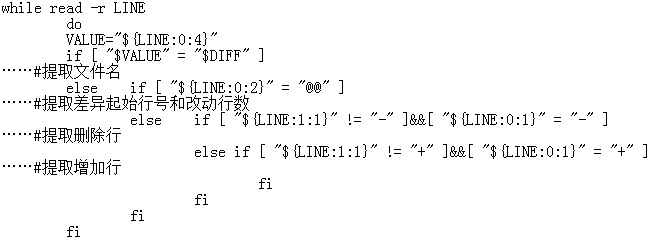
- trace\_kmem\_cache\_alloc(\_RET\_IP\_, ret, s->object\_size, s->size, gfpflags);

+ trace\_kmem\_cache\_alloc(\_RET\_IP\_, ret, s->objsize, s->size, gfpflags);

return ret;

}

然后就是根据每行的标志来提取出这些变化。路径、行号和行数无需改动，直接提取，而差异的行数要分成增加了多少行、减少了多少行记录下来，最后一起保存到文件中。减少的行是原始版本有、而比较版本没有的语句，增加的行是原始版本没有、比较版本有的语句。为了方便分别查询两个版本的改动函数，我们需要将两个版本的差异信息分开，也就是增的部分和减的部分。使用cut命令将两个版本相关的差异信息的切分保存到两个文件中，以方便后续处理，增加的部分与比较版本的数据库函数定义列表进行对照，减少的部分与原始版本的数据库函数定义列表对照。



然后就是要查找差异函数。diff结果的起始行号加上下文行数，就是改动的起始行数；diff结果的起始行号加改动行数减1，再减去上下文行数，等于改动的结束行号。连接数据库，找到减少的部分对应的函数定义列表，然后根据函数文件路径进行搜索，根据改动起始结束位置和函数起始结束位置进行对比，找到改动函数。这里的查找有两种情况：一种是改动范围小，改动全在一个函数之内，这时可以根据改动起始行号大于函数定义行号，改动结束行号小于函数结束行号；另一种情况是改动范围很大，覆盖几个函数，这时可以查找改动范围所包含的多个函数，这些函数都是被改动的。

将第一个差异文件的查询结果保存到数据库函数差异表中。将函数路径和函数名作为主键，另外的项为函数增加行数、减少行数和函数总行数，存入时，若一个文件中有某个函数、而另一个文件却没有这个函数，则认为是纯增加（或减少）的函数，设没有的那个版本增加（或减少）的值为0。

处理第二个差异文件的函数比上述过程要多个步骤，首先要判断函数差异表中是否已经在上述处理第一个差异文件时添加了该函数的部分差异项，如果已经存在了这个函数，则直接对这个函数的对应项进行修改，填入剩余的差异信息；若没有添加，则直接插入这个函数及差异行数，同样，作为纯增加（或减少）的差异，设另一部分差异行数为0。至此，函数差异已经存入数据库表。】

2、文件/目录差异分析

【继续使用上面得到的diff文件，用另一种方式分析其中内容。根据diff加参数标识的文件名，统计出该文件中标识“+”、“-”，也就是与行数增减相关的数目，并使用wc -l指令去该文件所在位置统计出该文件的总数目，输出的结果即为文件的差异以及该文件总行数。

然后分析路径，将路径以“/”依次拆开，代表每个上级目录，然后将每一个文件的差异值向上级每层目录的差异值进行累加。对所有文件分析完毕时，所有层次目录的差异值即为目录下所有文件差异值之和。

while line = file.gets

temp = line.split

#将改行各个元素插入到数据库中

length = temp[0].rindex("/")#每一次循环向上减少一级目录

while length != nil#直到目录分解完毕

temp[0] = temp[0].slice(0..length-1)

flag = 0

res = 数据库中查询到的内容

……#若数据库已经有该目录项，将差异数以及目录行数累加至当前目录的差异行数和目录总行数上

end

if flag == 0#目录项没有在数据库中，则建立目录项

……

end

length = temp[0].rindex("/")

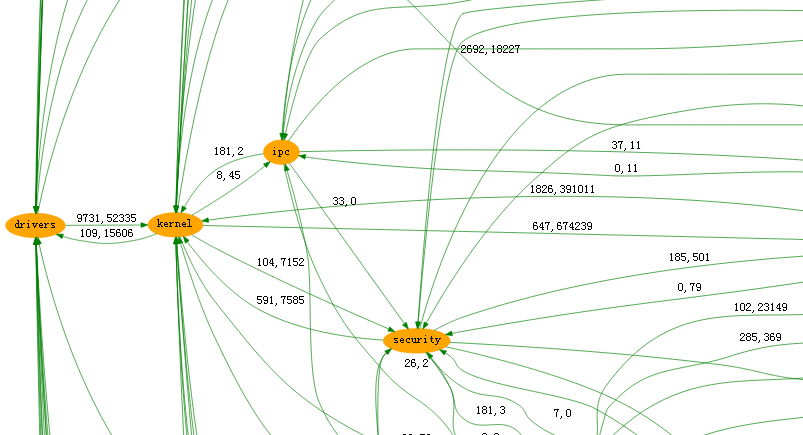
end

end

然后按照文件路径、增加行数、减少行数和总行数，将文件/目录差异分析结果存入到数据库表中。】

**Diffe工具生成函数调用关系图**

函数调用关系图，将函数调用关系转化为图形的形式，节点代表函数、目录或模块，边代表节点之间函数的调用及次数。画出函数调用图Diffe工具采用graphviz[25]的dot功能来实现，但如何生成画图需要的矩阵之前已经有同学完成，作者重点介绍了生成包含动态和静态函数调用图依赖数据矩阵的算法。



**图7.函数调用关系图展示**

算法步骤：

（1）生成函数调用图的节点。当输入为目录时，则节点为该目录下的目录和文件，逐级返回并生成节点，直到根目录为止；当输入为文件时，则该文件里的函数为节点，并按上述规则继续生成节点，直到根目录为止。

（2）确定顶点编号。为了统计方便，设定调用图中模块内部节点和外部节点颜色为不同的。按模块内、外顺序把节点按顺序编号，统计模块内、外节点的数目。所谓模块内节点为输入目录下的所有目录和文件节点，其余的节点则为模块外节点。

（3）统计边的个数。函数调用图是有向图，在边计数时要计算双向，即A−> B和B−> A全要统计。函数调用图要求静态和动态调用分别进行输出，因此对静态调用和动态调用要分别统计。具体作法是读取一遍边数据，查找主调、被调函数属于哪两个的节点，同时该边计数加1。

（4）计算每个节点内函数的总执行时间。对于函数而言需计算其本身的执行时间，而对于模块和文件而言则是该模块或文件内所有函数执行时间之和。

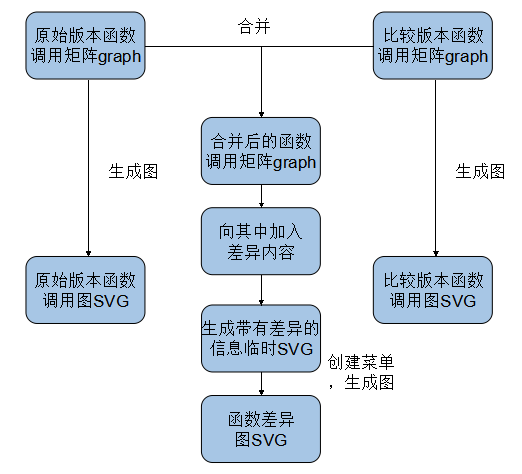
（5）设定节点和边的颜色。按模块内外分别设定节点颜色，边的颜色则分为模块内函数之间调用、模块外函数之间调用、模块内函数调用模块外函数以及模块外函数调用模块内函数，设定四种颜色。

（6）给节点和边加链接。使函数调用图能够生成所选模块下一级的函数调用关系表和函数调用图。

（7）画出SVG图。使用dot工具画出包含函数执行时间、静态、动态函数调用以及链接的函数调用图（SVG文件是纯粹的XML），具体命令为：

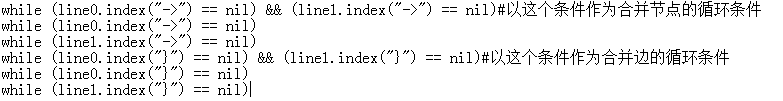
dot -Tsvg xxxx.graph -o xxxx-name.svg

在上述graph矩阵图中添加内容，把之前获得的函数差异信息以及所用到的链接加入其中即通过脚本对图像解析，向其中加入差异内容，在其他目录生成新的带有差异信息的graph矩阵图（使用callgraph工具画图）。为了保证全面的显示两个版本的差异，最保险的办法是使用两个版本的函数调用图，将两个版本的节点求并集，对边进行合并，静态与动态调用次数取差值，合成一个新的矩阵图。



**图8.Diffe工具画图流程**

使用有序字符串合并的思路，首先要对两个graph矩阵的节点与边分别按照一定规则进行排序，保证合并的正确性，通过脚本带的sort功能，将节点名称按照字典序排列。然后顺序进行合并，如果节点相同则保留一个，遍历位置同时向下一步；如果节点不同则先选择保留值较小的，然后向下一步，值大的不动。同理，对边的处理也是要先排序再合并。

节点合并以后，根据节点的目录或函数属性，从数据库得到之前计算的差异值，分别为减少行数、增加行数，然后用增减行数之和比总行数，得到作为差异程度的百分数，把它们作为节点的属性按照顺序依次填入到graph图中tooltip属性中，方便后期拆分处理做显示。节点的颜色则根据差异程度的多少进行染色，函数或目录差异的程度越大，则节点上面的灰度越深，反之灰度越浅。作者设定了一个差异区间对应的颜色规则。

$percent = (增加行数 + 减少行数)/总行数 #函数差异程度

……

if $percent < (最大的改动程度 \* 0.2)#改变节点颜色

设置颜色为“DDDDDD”

else

分别按百分比把节点设置不同颜色

end

if 一个文件有节点，而另一个没有

设置为红或绿

$percent = 1.0#百分比设置为100%

end

对于边的合并，只需要对边上的数值做差值，然后根据静态差值的正负设置边的颜色，正值为绿色，负值为红色。差值也作为边的一个属性写入到边属性的位置上。

if snum > 0

调用次数之差为正，设置为绿色

elsif snum < 0

调用次数之差为负，设置为红色

else

否则为黑色

end

合并的同时将节点url信息更新。在合并中，需要将链接内容附加上带有第二个版本的信息，链接字段如depth=linux-3.5.4，插入到节点的url的平台号与路径之间，通过“&”分隔开。

def redoURL(str)#处理url，加入比较版本信息

将原url在指定位置分割为前缀和后缀

newurl = 前缀 + 比较的版本 + 后缀

return newurl

end

**函数代码差异**

函数代码差异部分主要分成三步：

1. 从两个版本的源代码中找到函数体并提取出来保存成文件

res = 函数定义列表中是否能找到函数

fdline1 = 函数定义行号

frline1 = 函数返回行号

……

while 遍历函数所在文件

count1 = count1+1

if count1 == fdline1 #找到函数

flag = true

end

if flag == true#输出函数

write1.syswrite(line)

end

if count1 == frline1#结束输出

break

end

end

（2）对两个函数文件进行diff处理，加入足够大的上下文参数

（3）若第（2）步有生成差异文件，则对生成的diff文件做可视化显示

用到了diff转化为html的开源脚本工具diff2html.py。

**目录/模块具体差异**

根据输入的目录/文件名路径，到数据库中寻找目录文件，得到该目录下面所有目录和文件。对其中数据进行筛选，只保留目录下一级的路径信息，包括子文件夹的差异数目和文件的差异数目也在其中显示。

if 判断是否为该目录

flag=1

elsif 通过目录级数来判断是否是该目录的下一级文件

flag=1

elsif 通过目录级数来判断是否是该目录的下一级目录

flag=1

end

然后将输出信息组装成html格式，用一个table表格将其表示出来

最后将上述所有功能集合到一起就是diffe模块，然后与内核分析平台整合到一起。

**2、语法差异分析**

为了实现更全面的差异比较，所以进行语法差异分析。

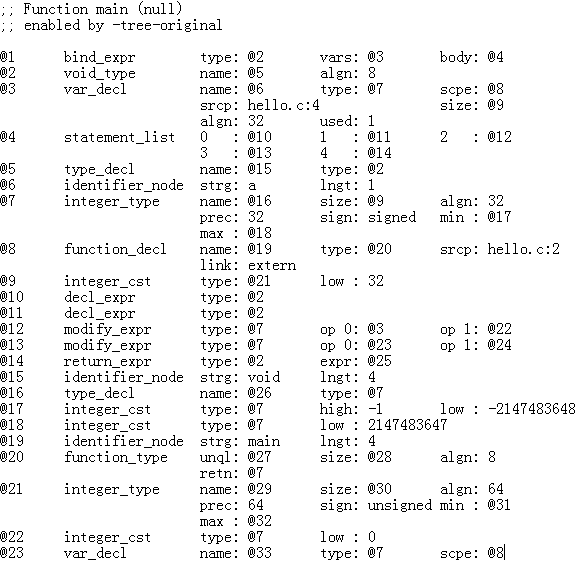
基于GCC语法树。生成语法树的参数有三种，分别是

-fdump-tree-switch

-fdump-tree-switch-options

-fdump-tree-switch-options=filename

【其中switch是为语法树文件附加一个特定后缀，并有着特殊的格式。而options选项是为语法树增加附加的形式，这个形式可能会与switch发生矛盾而没有意义，这时options会被忽略。当有=filename选项作为后缀的时候，生成的是给定文件名的语法树，而不是自动命名。switch参数是“-fdump-tree-original-raw”，生成的语法树如下图。



在这个文件中，抽象语法树以Function连接函数名作为函数的开始，也是一棵语法树的根部。Function下面是抽象语法树的所有节点列表。每个节点都有一个标号，也作为访问该节点的唯一索引，用@数字来表示。后面跟着的是节点标识符，一个字符串，以及带@的节点属性序列。节点标识符描述该节点是何种类型，代表了节点的含义，如常量或运算符。其余部分为节点属性序列，每个标号代表着该类型节点的一个属性，也对应一个节点标号。如果想遍历这棵树，可以使用深度搜索和广度搜索，沿着节点标号到达每一个节点。】

但作者使用的不是这种形式，使用的是另一种语法树 — 控制流图。生成时使用的参数为-fdump-tree-cfg-raw，生成文件后缀为cfg。这棵语法树可以使用参数-lineno，这个参数的意思是在展开的语法树中加入与源代码对应的行号信息，即便文件中的节点破坏了原来的位置，也可以与通过行号找到源代码的位置。

论文中的例子

#include<stdio.h>

int main()

{

int a,b;

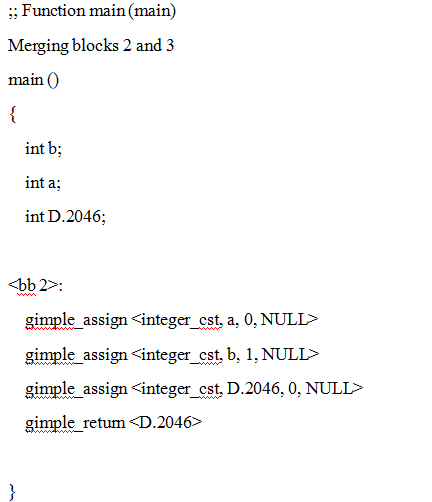
a=0;

b=1;

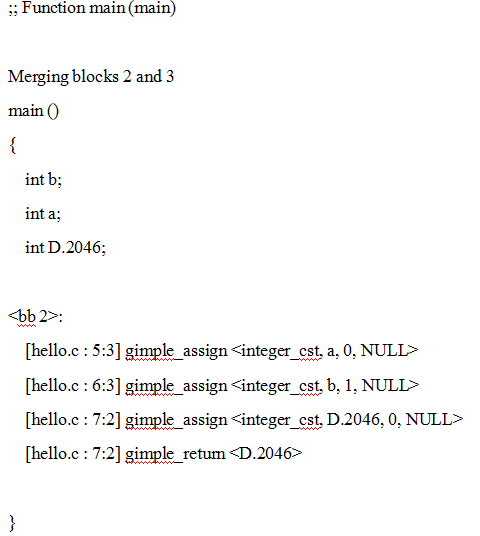
return 0;

}

然后通过gcc -fdump-tree-cfg-raw XXX.c，这个程序生成的语法树名字为XXX.c.013t.cfg格式如下面。

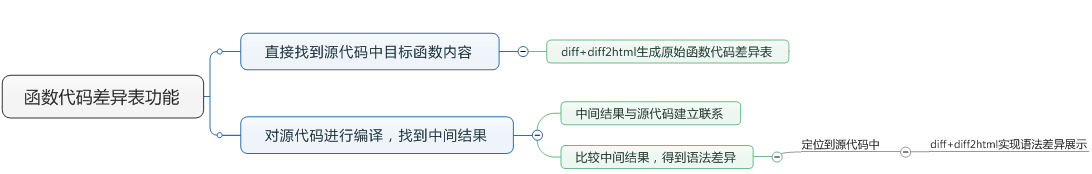


加入lineno参数，使用gcc -fdump-tree-cfg-raw-lineno XXX.c命令，生成的文件名不变，还是XXX.c.013t.cfg，生成的语法树如下图。



【树中包含着以gimple命名的标签，gimple下划线连接的文字则包含着着语义信息，如gimple\_call是调用其它函数，gimple\_assign是赋值语句，gimple\_return是返回值，所以用gimple可以轻易地识别该语句在源代码中的功能。而在后面的尖括号中，则包含着这个语句用到的变量、常量以及表达式，复杂的语句的功能则可以分解成多个gimple语句表示。

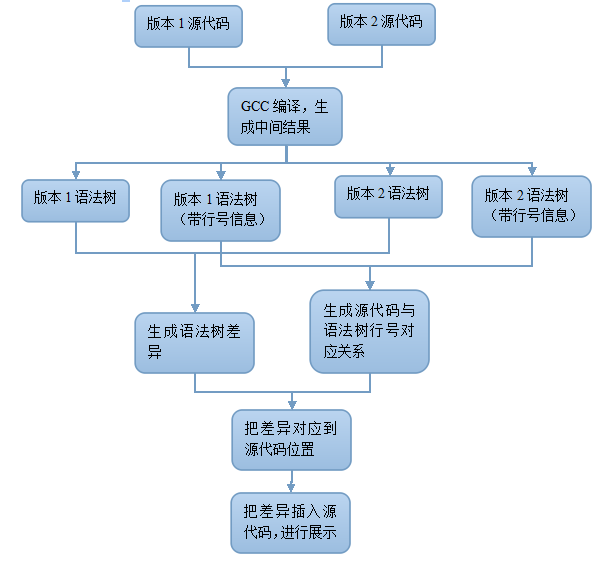
加入lineno参数的语法树在每一个非定义的、有实际意义的语句前，新加入了格式为[文件名:行号:长度]的行号信息，我们需要对比的信息已经都被标注了，我们可以认为，没有标注行号的行是不包含我们需要的函数功能信息的。而在行中出现的D.整数，则是程序中返回表达式计算等等所用到的临时变量，这个由系统临时创造的变量不是源代码中有的，而是作为一个临时存放的容器，生命周期很短。不过由于做差异要多次生成语法树，我们不能保证多次生成的语法树的临时变量名是一样的，所以在做差异比较之前，首先要对这种变量做一个预处理。根据该变量的特点，我们决定保留该变量的基本类型，而去除其编号。】



**图12.语法差异分析功能框架**

【函数代码差异是直接找到函数进行对比，然后显示。而与上一章的函数代码差异的生成方式不同，函数语法差异功能首先要对源代码编译生成中间语法树，然后对这课语法树进行分析，找到并建立语法树和源代码的关系。然后使用两个版本的语法树信息进行比较，得到语法差异。然后通过源代码与语法树的对应关系，将语法差异内容插入到源代码函数中，这样就能完整的显示带有语法信息的函数差异。总之，函数语法差异是在代码差异的基础上完成的补充内容。】

**使用控制流图实现语法差异的设计**



**图13.语法差异分析功能实现流程**

**实现步骤：**

1、使用gcc编译源代码文件，并进行预处理

为了生成中间语法树，在makefile文件中加入make CXXFLAGS=-fdump-tree-cfg-raw all，即可生成对应的语法树文件信息。并且为了避免文件命名重复发生覆盖的显现，在每次编译后利用脚本将生成的语法树统一修改为另一格式的文件名。

gcc -fdump-tree-cfg-raw $1

mv $1.013t.cfg $1.cfg

gcc -fdump-tree-cfg-raw-lineno $1

mv $1.013t.cfg $1.lcfg

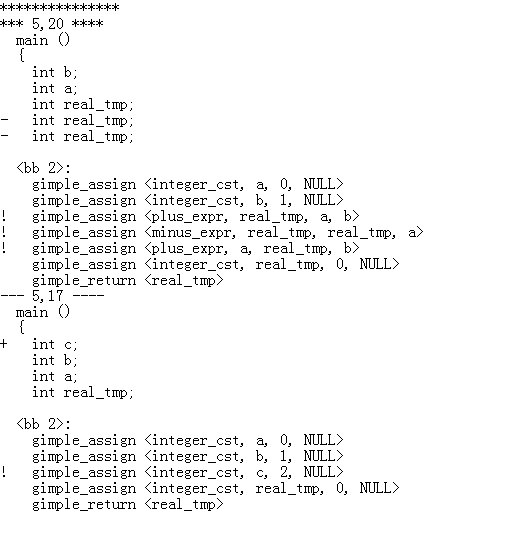
gcc -fdump-tree-cfg-raw $2

mv $2.013t.cfg $2.cfg

gcc -fdump-tree-cfg-raw-lineno $2

mv $2.013t.cfg $2.lcfg

此时已经生成了语法树1、带行号的语法树1、语法树2、带行号的语法树2，然后利用sed命令将文件中带有“D.整数”格式的编号变量替换为容易识别的符号，临时变量类型可以是任何基本类型，所以这里使用“real\_tmp”来替换。然后就可以对语法树1和语法树2进行上下文差异比较，关注差异的位置。使用如下两个文件分开的格式生成比较文件。



“\*\*\*”符号包含的是语法树1的内容，“---”符号包含的是语法树2的内容。在差异每一行的起始位置上“+”“-”代表增加和减少的差异，而“！”代表有有变动的行。

2、建立语法树行号与源码的行号之间一个多对一的关系

【分析带行号的语法树，把带有源代码行号信息的行号提取出来，并提取出该行的行号信息。因为语法树经常会把源代码中一行的内容展开为多行去解释，这时就会存在一个多对一的关系，所以让语法树行号为索引，源代码行号为存储的内容，这样也方便使差异信息直接定位回源文件。

然后分析生成的差异文件，分别对两个语法树差异内容进行读取，首先读取差异开始位置的起始行号，然后每行顺次读取，每行行号加一以计算差异所在的行号。当读取到带有“+”“-”“！”时，就意味着已经寻找到了差异，根据计算的差异在语法树中文件的行号，可通过之前保存的对应关系对应到源代码位置，这时就可以把记录的语法差异对应回源代码位置上，之后就可以在函数细节差异内容展示上加入语法差异信息。

**脚本设计**

之前的编译及预处理，则使用shell来完成。语法差异生成脚本依然使用ruby来编写，第一步首先是要提取行号关系，两个语法树行号的对应关系通过两个数组来保存。

linearray1 = Array.new(100000) #带行号文件1的行号

linearray2 = Array.new(100000) #带行号文件2的行号

读取行号语法树文件，分析语法树的格式，提出有效行对应的源行号，以现在行号为索引，存储在数组中。

while 遍历行号文件

col = 语法树中源代码行号标识

if col == nil

else

tnum = 提取改行的源代码行号

保存到对应行号关系的数组中

end

行号 += 1

end

第二部是做差异比较，以flag判断目前是在处理语法树文件1还是2，避免混淆，然后提取出差异的起始行号。

if line[0,4] == "\*\*\* "

flag = 1

start = 提取的起始行号

elsif line[0,4] == "--- "

flag = 2

start = 提取的起始行号

end

对差异文件提取，要根据flag的值判断是第几个文件的修改，然后去该文件的行号对应关系中找到差异在源代码中的位置，然后一起输出差异结果。如果数组中没有这个差异的行号，就意味着这一行可能是定义，也可能是标签，这些的含义在文本差异比较也是可以看出来的，所以我们不重复提取。

if line[0,2] == "- "

if 改动在文件1中

输出删减内容到文件1的源文件中

elsif 改动在文件2中

输出删减内容到文件2的源文件中

end

elsif line[0,2] == "+ "

if 改动在文件1中

输出增加内容到文件1的源文件中

elsif 改动在文件2中

输出增加内容到文件2的源文件中

end

elsif line[0,2] == "! "

if 改动在文件1中

输出改动内容到文件1的源文件中

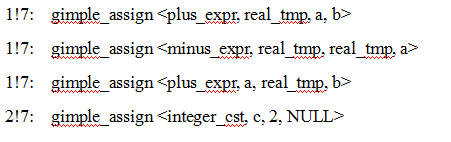
elsif 改动在文件2中

输出改动内容到文件2的源文件中

end

end

输出结果如下所示。



然后读取两个源代码文件，分别将得到的语法差异信息插到源文件相应的行的后面，用以解释该行的语法差异，同一行的语法差异去掉换行符合并到一起，将改完的代码写入到新的文件中，用于后续的差异展示。

while 遍历语法树改动行号与源代码行号的对应关系

if 改行合并完成

换行

end

line2 = 源文件的内容

count2 += 1

if count2 < 有差异区域

原样输出源文件行

end

end

if 合并未完成

去掉源代码修改行的回车

end

将语法差异内容加入到源代码改行末尾（将语法差异内容合并至一行）

**差异显示**

使用函数定义列表，将所选择的带有语法差异信息的函数提取出来。再将上述生成的两个带有语法差异的源代码再次进行对比，得到的就是既带有文本差异又带有语法差异的文件。再次使用diff2html工具，生成的可视化结果。】

**常用版本控制工具**：

Concurrent Version System [1]（CVS）

Subversion Version Control [2]

Marc Rochkind的SCCS（Source Code Control System）[3]和Walter Tichy的RCS（Revision Control System）[4]

Git[5] VSS[7] Rational ClearCase[8] StarTeam[9]

**差异展现工具**

diff

KDiff3[14]是一个用于对文件或目录做比较或合并的工具，他能比较2~3个文件，也能对目录的内容作比较

Kompare[15]同KDiff3一样也是适用于KDE桌面的比较工具。

Vimdiff[16]是一款非常高效的文件的比较合并工具

ColorDiff[17]是一个Perl语言编写的脚本。

Meld[18]是文件及目录比较的图形化工具

Diffuse[19]也是文件及目录比较的图形化工具。

DIFF2HTML[20]没有自身DIFF的功能，需要先将系统自带的diff功能生成文本差异文件，再去对该文件进行处理。

个人见解

因为作者为了提取差异信息方便将生成的语法信息插入到源码文件中，这样修改了源码的原有内容，另外对于函数的参数变化，接口返回类型的比较没有涉及，因此，对于内核更新来说，还尚有一段路要走。