

广东中兴新支点

KCov 用户手册

嵌入式操作平台

# 声 明

本资料著作权属广东中兴新支点技术服务有限公司所有。未经著作权人书面许可，任何单位或个人不得以任何方式摘录、复制或翻译。

侵权必究。



是广东中兴新支点技术服务有限公司的注册商标。广东中兴新支点产品的名称和标志是广东中兴新支点的专有标志或注册商标。在本手册中提及的其他产品或公司的名称可能是其各自所有者的商标或商名。在未经广东中兴新支点或第三方商标或商名所有者事先书面同意的情况下，本手册不以任何方式授予阅读者任何使用本手册上出现的任何标记的许可或权利。

由于产品和技术的不断更新、完善，本资料中的内容可能与实际产品不完全相符，敬请谅解。如需查询产品的更新情况，请联系广东中兴新支点技术服务有限公司。

若需了解最新的资料信息，请访问网站 <http://www.gd-linux.com/>。

# 手册说明

本手册是广东中兴新支点 KCov 用户手册，向用户介绍如何进行 KCov 的安装及使用。全文图文并茂，易于用户阅读使用。

## 内容介绍

章 名	概 要
第 1 章 概述	介绍何为 KCov，及其使用流程
第 2 章 安装	介绍 KCov 安装方式
第 3 章 用户态生成覆盖率数据	介绍用户态代码覆盖率分析时所需的覆盖率数据生成方式
第 4 章 内核态生成覆盖率数据	介绍内核态代码覆盖率分析时所需的覆盖率数据的生成方式
第 5 章 使用 KCov 分析代码覆盖率	详细描述了如何使用 KCov 进行代码覆盖率分析
第 6 章 开源 LICENSE 说明	简单介绍 GPL

## 术语说明

术语	描述
KCov	一种代码覆盖率分析工具


## 版本更新说明

文档版本	发布日期	备 注
KCov_V3.02.20_P1B3	2013-3-20	2013 年第一次版本发布

## 本书约定

介绍符号的约定、键盘操作约定、鼠标操作约定以及四类标志。

## 1) 符号约定

- 样式  表示按钮名；带方括号 “【属性栏】” 表示人机界面、菜单条、数据表和字段名等；
- 多级菜单用 “->” 隔开，如 【文件】->【新建】->【工程】 表示【文件】菜单下的【新建】子菜单下的【工程】菜单项；
- 尖括号<路径>表示当前目录中 include 目录下的.h 头文件,如<asm-m68k/io.h>表示 `/include/asm-m68k/io.h` 文件。

## 2) 标志

本书采用二个醒目标志来表示在操作过程中应该特别注意的地方：



提示：介绍提高效率的一些方法；



警告：提醒操作中的一些注意事项。

# 联系方式

电 话： 021-87048510

电子信箱： cgel@gd-linux.com

公司地址： 广州市天河区高唐软件园基地高普路 1021 号 E 栋 601 室

邮 编： 510663

# 目 录

<b>1. 概述.....</b>	<b>1</b>
1.1. 功能说明.....	1
1.2. 使用流程.....	2
1.3. 获取.....	错误！未定义书签。
<b>2. 安装.....</b>	<b>2</b>
<b>3. 用户态生成覆盖率数据 .....</b>	<b>3</b>
3.1. 编译生成.gcno 日志文件 .....	3
3.1.1. 使用 KIDE 编译.....	3
3.1.2. 用户管理 Makefile 方式 .....	8
3.2. 运行生成.gcda 数据文件 .....	9
3.3. 关于无限循环应用程序的覆盖率数据生成方法 .....	10
3.4. 关于程序运行中覆盖率数据实时生成方法 .....	10
<b>4. 内核态生成覆盖率数据 .....</b>	<b>11</b>
4.1. 单板对内核映像的大小限制 .....	11
4.2. 生成.gcno 日志文件 .....	12
4.2.1. 内核配置 .....	12
4.2.2. 开启或屏蔽内核目录或文件代码覆盖率开关的方法.....	15
4.2.3. 编译 .....	17
4.3. 生成.gcda 数据文件 .....	18
4.3.1. 动态加载 gcov-proc 模块 .....	18
4.3.2. 数据生成、重置和目录 .....	19
<b>5. 使用 KCOV 分析代码覆盖率 .....</b>	<b>22</b>

5.1. 使用图形界面 KCov.....	22
5.1.1. 启动 KCov.....	22
5.1.2. 添加待分析文件或目录 .....	23
5.1.3. 修改待分析文件或目录 .....	32
5.1.4. 增量添加分析文件或目录 .....	32
5.1.5. 覆盖率数据累加 .....	34
5.1.6. 解析代码覆盖率分析结果 .....	38
5.1.7. 保存覆盖率数据 .....	61
5.2. 使用命令行 KCov.....	65
<b>6. FAQ .....</b>	<b>71</b>
6.1. 点击 KCov 分析工具图标无响应.....	71
<b>7. 开源 LICENSE 说明 .....</b>	<b>72</b>
7.1. GPL 简介.....	72
7.2. 开发指导.....	73
7.3. 源码获取方式.....	74

# 1. 概述

## 1.1. 功能说明

在代码发布之前，我们需对其进行一系列的测试来协调软件的性能和功能，使其达到预期的设想。一般借助一些测试工具来模拟或重建执行脚本，对程序的各个功能进行测试以证明其是可以工作的。

代码覆盖率分析即是定位没用的或者不执行的代码的过程，从而增强程序的可读性和执行效率，降低不执行代码中存在 bug 的风险。代码覆盖率分析工具——KCov，能够检验出程序中的每一条语句是否都被执行过，从而帮助您剔除软件中的 dead code（无用的、不会被执行的、以及腐朽的代码）。

KCov 对于代码的覆盖率分析分为以下两类：

- 内核态代码覆盖率分析
- 用户态代码覆盖率分析

对于一个复杂的项目来讲，不可能一次性启动所有的功能，因而生成的覆盖率数据也仅仅是针对其中某一项功能而言的。即是要求使用者必须在清楚某项功能的代码分布的前提下，才可能有目的性的去观测哪些文件、哪些函数的覆盖率情况。

KCov 与开源代码覆盖率分析工具的功能对比如下：

**1-1 KCov 与开源代码覆盖率分析工具功能对比**

特点	gcov	ggcov	KCov
图形界面、命令行支持情况	有命令行，无图形	有命令行，有图形	有命令行，有图形
保存输入数据源路径	支持	不支持	支持
无源码解析	不支持	不支持	支持
elf 文件解析	不支持	部分支持，无法交叉解析	支持
汉化	不支持	不支持	支持

覆盖率数据解析效率	不支持多文件方式解析，效率较低	支持多文件解析，解析算法有缺陷，文件规模较大时解析效率很低	支持多文件、单文件的解析，文件规模较大时解析效率高
动态编译程序覆盖率数据生成	只生成动态库或程序的覆盖率数据	只生成动态库或程序的覆盖率数据	可同时生成动态库和程序的覆盖率数据
覆盖率数据累加	不支持	不支持	支持
多文件覆盖率显示	不支持	支持	支持
windows 系统支持	支持	不支持	支持
交叉解析	有缺陷	支持	支持
输入数据源方式	命令行	命令行	命令行、图形、配置文件
html 文件保存覆盖率数据	不支持	不支持	支持

## 1.2. 使用流程

使用 KCov 的代码覆盖率统计功能，大致分以下几步：

1. 编译生成.gcno 日志文件；
2. 运行生成.gcda 数据文件；
3. 得到覆盖率数据后，使用 KCov，结合源代码，进行代码覆盖率分析。

## 2. 安装


若是下载 KIDE 集成开发环境，完成 KIDE 的安装后，则自动安装好了 KCov 代码覆盖率分析工具。

若是直接下载的 KCov 压缩包，则将其解压到任意目录下，会生成一个 kcov 目录，其下包含了 kcov、cygwin 目录和 startkcov.exe 文件。



## 3. 用户态生成覆盖率数据

在使用 KCov 对代码进行覆盖率分析之前，需保证存在与之对应的覆盖率数据，即编译时生成.gcno 或.bb 或.bbg 日志文件，其包括了关于程序的相关文件，向开发者提供相应信息，可以用这些文件来重组每一个可执行程序的程序流图，用于静态分析哪些代码行需要计算覆盖率。程序执行时生成.gcda 或.da 数据文件，这类文件中包含了每一个指令分支的执行次数，与源码文件一起使用，来标识源码的执行能力，用于记录程序运行时的实际覆盖率数据。因而，首先，对如何生成覆盖率数据进行介绍，此处我们以生成.gcno 和.gcda 为例。

 提示：若 gcc 版本在 gcc-3.3.6 之前，则在编程时生成.bb 或.bbg，在程序执行时生成.da 数据文件。之

后的 gcc 版本，在编程时生成.gcno，在程序执行时生成.gcda。

### 3.1. 编译生成.gcno 日志文件

#### 3.1.1. 使用 KIDE 编译

##### 3.1.1.1. 新建用户态项目

启动 KIDE 集成开发环境后，按照以下步骤创建一个用户态项目：

1. 主菜单/项目视图右键菜单->【文件】->【新建】->【项目】;
2. 选择【User Model 项目】->【User 项目】，点下一步;
3. 为项目取名，点下一步;
4. 选择希望部署的平台和配置：选择“可执行的（默认）”，点下一步;
5. 按自身需求，选择【CPU 体系】、【CPU 型号】、【编译工具链】，点完成。

一个用户态项目创建完毕，用户可自行添加需进行覆盖率测试的代码。若需查看项目的具体创建方法请参照《广东中兴新支点 KIDE 2.8 用户使用手册》。

 提示：静态库、共享库的用户态程序没有通过 KIDE 管理 Makefile 的方式提供代码覆盖率的功能，

但可以通过用户自己管理 Makefile，即在编译和链接时添加相应参数来进行代码覆盖率的测试，具

体方式请参考 3.1.2。

针对用户态项目的代码覆盖率分析，即可以对整个项目开启代码覆盖率开关，也可以仅对单个文件开启代码覆盖率开关，针对前者，需要修改项目属性中编译和链接的参数，而后者则需要针对单个文件修改其编译参数，再在项目属性中修改其链接参数。

### 3.1.1.2. 开启项目代码覆盖率开关

若是对整个项目进行代码覆盖率测试，则可以通过修改项目属性的方式，开启针对整个项目的代码覆盖率开关。在 C/C++项目管理视图中，对新建的用户态项目名称点击**右键**，在打开的右键菜单中，选择“属性”选项。打开“属性视图”后，添加“代码覆盖率”编译选项，C++程序，如图 3-1，C 程序，如图 3-2，添加“代码覆盖率”链接选项，如图 3-3。必须同时为编译选项和链接选项添加“代码覆盖率”参数，才能保证代码编译后在与.o 文件相同目录下生成\*.gcno 日志文件以及运行后生成\*.gcda 数据文件。

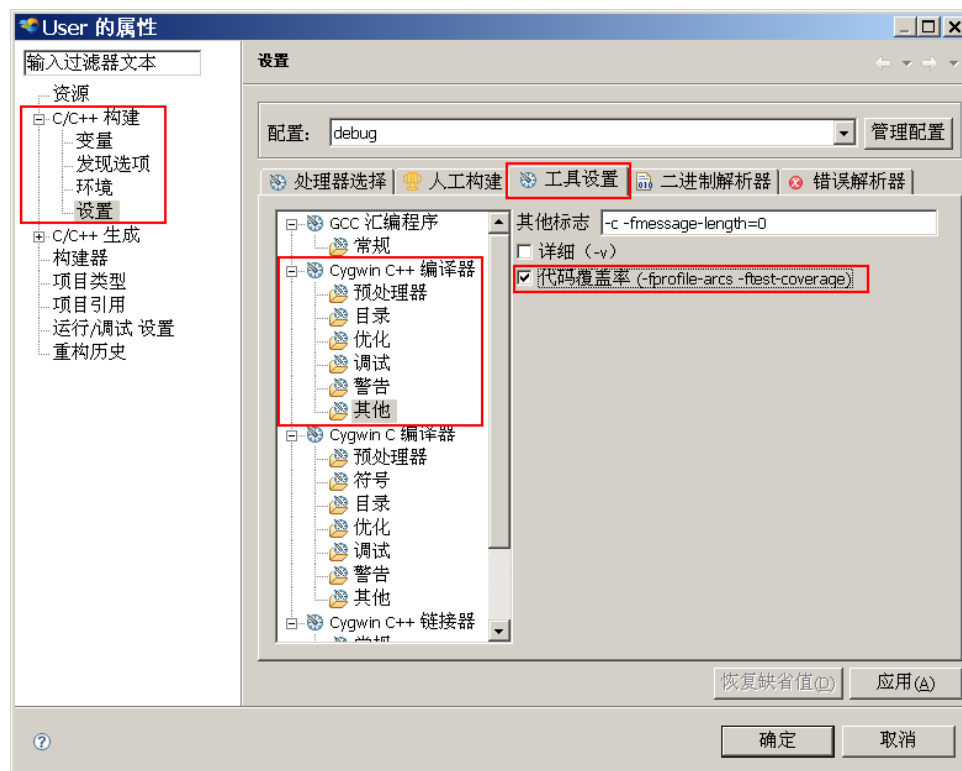


图 3-1 C++程序添加“代码覆盖率”编译选项

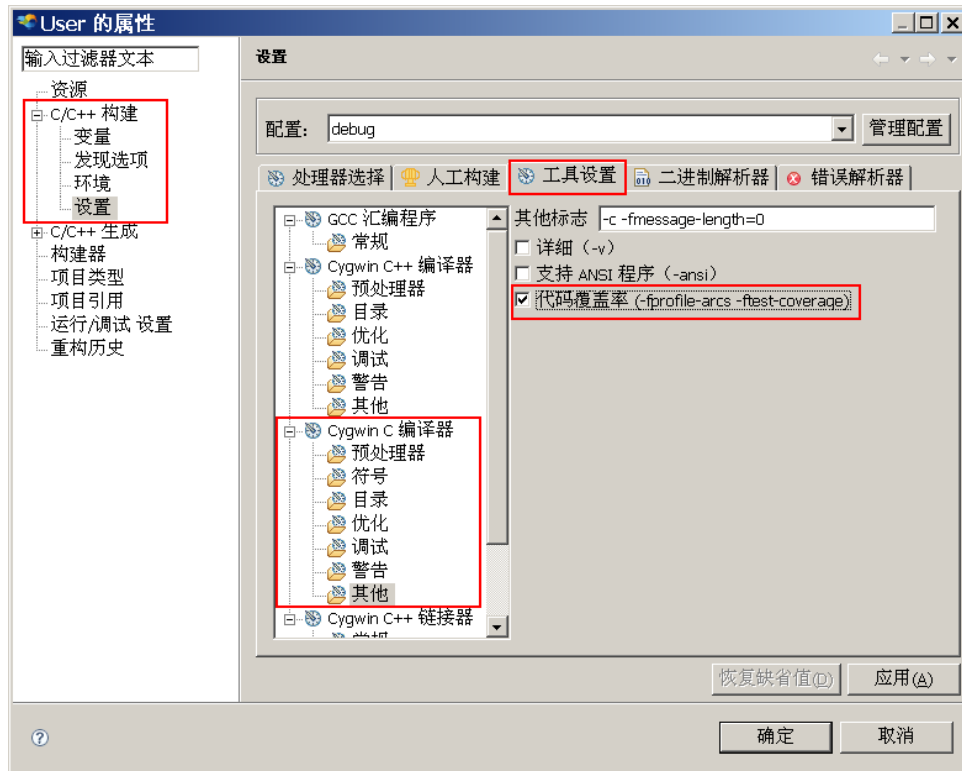


图 3-2 C 程序添加“代码覆盖率”编译选项

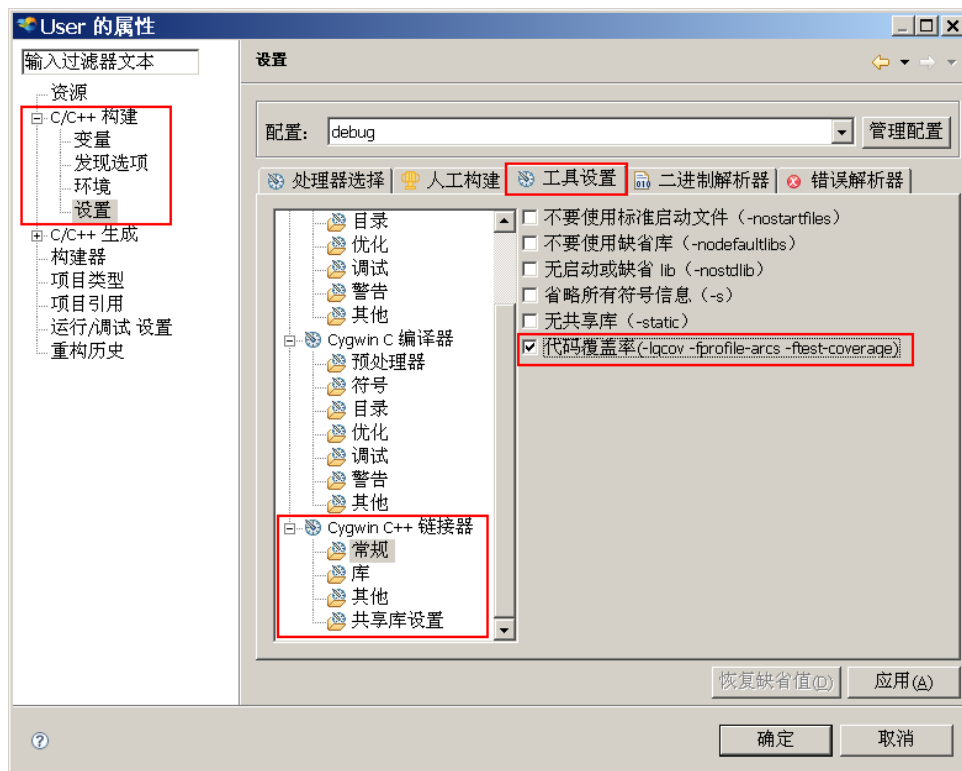


图 3-3 添加“代码覆盖率”链接选项

修改项目属性后，对用户态项目进行构建，会发现.gcno 日志文件已经生成。

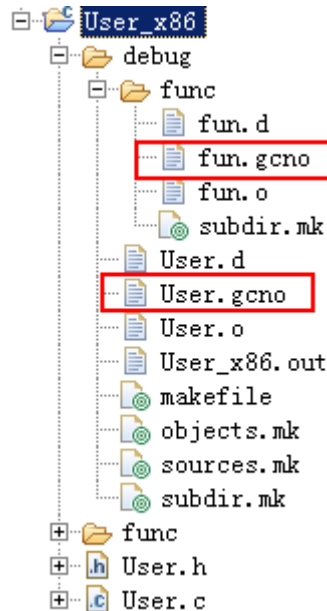



图 3-4 生成.gcno 日志文件

 提示：调试器的 kerlibc 内核模块不能使用 kcov 编译，否则内核无法正常运行。

### 3.1.1.3. 开启单个文件代码覆盖率开关

若只希望对单个文件进行代码覆盖率测试，则可以通过修改单个文件属性的方式开启对单个文件的代码覆盖率开关。对需要进行代码覆盖率测试的文件点击**右键**，在右键菜单中，选择“属性”选项。开启文件属性设置窗口，参照如下提示进行勾选，添加“代码覆盖率”编译选项，如图 3-5，之后再对文件所在**项目的属性**设置里面加链接标志，如图 3-6。必须同时为编译选项和链接选项添加“代码覆盖率”参数，才能保证代码编译后在与.o 文件相同目录下生成\*.gcno 日志文件以及运行后生成\*.gcda 数据文件。

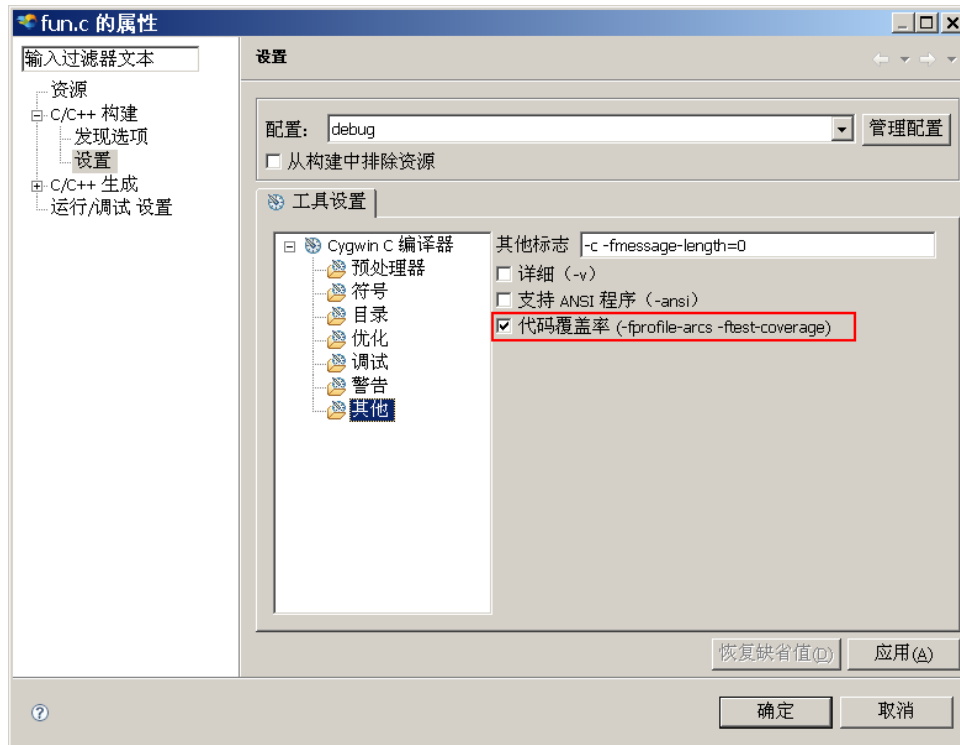


图 3-5 对单个文件添加代码覆盖率编译选项

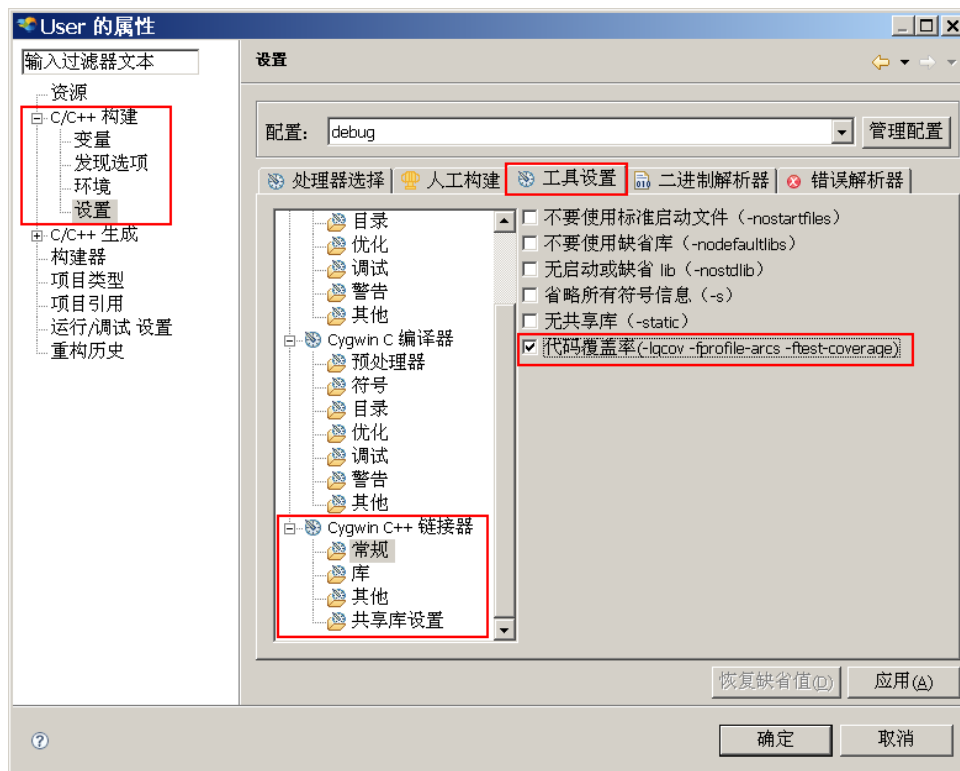


图 3-6 对项目添加“代码覆盖率”链接选项

修改完文件属性后，对用户态项目进行构建，则可生成针对此文件的.gcno 日志文件。

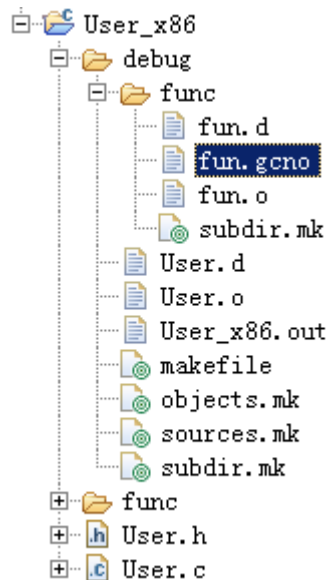


图 3-7 生成针对 fun.c 单个文件的覆盖率数据

### 3.1.2. 用户管理 Makefile 方式

若是用户自行在 Linux 环境下编译程序，需使用 KIDE 提供的工具链才能生成日志文件.gcno。请按如下方式修改 Makefile 文件：

- 编译时添加选项： `-fprofile-arcs -ftest-coverage`;
- 链接时添加选项： `-lgcov`。

➡ 提示：使用标准 gcc 也能生成 gcno、gcda，但不能为 KCov 所用。

➡ 提示：由于库 libgcov 属于工具链自身的库，而非 C 库，因而当使用 ld 进行链接时，报错“can't find -lgcov”，此时应当增加-L 参数，用于指定编译工具链自身的库路径，例如若使用 x86\_64 工具链，其库路径为“当前 x86\_64 工具链路径/lib/gcc/x86\_64-pc-linux-gnu/4.1.2”，且需将-L 参数置于-lgcov 参数

之前。

## 3.2. 运行生成.gcda 数据文件

将编译生成的可执行文件放置到目标机上运行，将生成与编译源码文件时生成的.o 文件同名的.gcda 数据文件，此文件中收集了代码覆盖率的测试数据，且其在目标机上的存放目录结构也与.o 文件相同。

```
# ls
User_x86_2.out  etc          linuxrc      switchToKsh
bin            home         proc         sys
cygdrive       init         root         tmp
dev            lib          sbin         usr
# cd cygdrive/d/target/workspace_KCov/User_x86_2/debug/
# ls
User.gcda
```

图 3-8 生成.gcda 文件

若是不通过下载，也可选择共享的方式，使得主机与目标机上的数据能够互访，有以下两种方式供选择：

1. 通过 samba 方式，将目标机目录映射到本机。在目标机上生成的.gcda 文件可直接被主机访问，若源文件和.gcno 在目标机上，也可通过 samba 直接被主机访问。
2. 源文件在主机，编译生成的可执行文件运行在目标机，则可以通过 mount 的方式，将主机目录映射到目标机。

如在 Windows 端 “D:\target\workspace\_KCov\User\_x86\_2\debug” 目录下生成目标文件 User.o，此刻将 “D:\target\workspace\_KCov\User\_x86\_2\debug” 这一目录挂载到目标端的 mnt 目录下：

```
# mkdir mnt
# mount 192.168.10.1:/cygdrive/d/target/workspace_KCov/User_x86_2/debug /mnt
# ls
bin            home          linuxrc      root         sys
dev            init          mnt          sbin         tmp
etc            lib           proc         switchToKsh  usr
# cd mnt
# ls
User.d          User.o        makefile     sources.mk
User.gcno       User_x86_2.out objects.mk    subdir.mk
```

图 3-9 挂载目录

运行可执行文件（User\_x86\_2.out），默认情况，会自动在目标端 “cygdrive/d/target/workspace\_KCov/User\_x86\_2/debug” 目录下生成 User.gcda 数据文件。如果这一放置目录不存在，且无法创建，那.gcda 文件也会创建失败。另外，可以通过修改环境变量 GCOV\_PREFIX 和

GCOV\_PREFIX\_STRIP 的方式来动态设置.gcda 的输出目录。我们将输出目录设置为挂载的目录路径，则 gcda 数据生成后，可在主机端直接被访问，非常方便。

具体设置方法如下：

```
export GCOV_PREFIX=/mnt
export GCOV_PREFIX_STRIP=6
```

数字代表目录的层数，此处挂载目录“cygdrive/d/target/workspace\_KCov/User\_x86\_2/debug”有 6 层，因而这样就会生成数据文件/mnt/User.gcda。

```
# cd /mnt
# ls
User.d      User.o      objects.mk
User.gcda   User_x86_2.out sources.mk
User.gcno   makefile    subdir.mk
```

图 3-10 生成.gcda 文件

### 3.3. 关于无限循环应用程序的覆盖率数据生成方法

若是希望对无限循环的应用程序进行代码覆盖率分析，需按照如下步骤进行，才能生成覆盖率数据：

1. 修改应用程序，添加一个信号处理函数：

```
static void sighandler( int sig_no )
{
    exit(0);
}
```

2. 在程序进行初始化时注册 `signal(SIGUSR1,sighandler);`
3. 程序运行后，使用如下命令，程序即可正常退出并生成覆盖率数据：

```
kill -USR1 pid
```

### 3.4. 关于程序运行中覆盖率数据实时生成方法

覆盖率数据被收集并保存在内存中，默认情况下，仅在程序正常退出，或程序中调用 `fork`、`exec` 创建子进程的时候才会将这些覆盖率数据保存在数据文件中。这也就意味着数据文件并未对这些覆盖率数据进行实时保存。针对此种情况，用户可通过在源代码中新增一个信号的方式，在期望获得数据文件时，向程序发送该信号，一旦程序收到此信号，即刻保存数据文件并清空内存数据。此数据文件可不断累加、



更新，具体步骤如下：

1. 修改应用程序源码，添加信号处理函数：

```
static void sighandler( int sig_no )
{
    __gcov_flush();
}
```

2. 在调用\_\_gcov\_flush 时加上声明：**extern void \_\_gcov\_flush (void);**
3. 在程序进行初始化时注册 **signal(SIGUSR1,sighandler);**
4. 程序运行后，使用如下命令发送信号，程序即可生成覆盖率数据：

```
kill -USR1 pid
killall -USR1 program_name
```

## 4. 内核态生成覆盖率数据

### 4.1. 单板对内核映像的大小限制

boot 对原始内核映像的大小有限制，且限制不同。多数单板 kboot 对内核映像解压后的大小限制在 8M，uboot 对内核映像解压后的大小限制在 4M。具体情况视单板烧录的 boot 而定。需要注意的是可以通过对编译后生成的 vmlinux 进行 stirp（工具链对应 bin 目录里）操作，去掉调试信息，得到的 vmlinux 大小与 uImage 在单板上解压后的大小基本上一致。该 vmlinux 为 elf 格式文件，在 object/obj/lsp/make\_image/image 下也有对应的内核映像，如 vmlinux.bin，该文件为 bin 格式，比 elf 稍小一些，也可以通过查看该文件估计原始内核映像大致的大小。

PPC、PowerPC 架构，多数单板在内核配置时开启了 **Profile entire Kernel** 时，原始内核映像超过 8M。在使用代码覆盖率功能时，注意原始内核映像的大小。可以通过上述方式，对 vmlinux 使用 stirp，去掉调试信息，查看内核映像的大小是不是在 kboot 或 uboot 规定的大小之内。其他架构，也与此类似。

## 4.2. 生成.gcno 日志文件

### 4.2.1. 内核配置

要使用 KCov 工具，首先在内核配置时必须配置 KCov 支持。无论是针对 KTH 内核还是 NO-KTH 内核，其内核配置方法完全一样。打开内核配置窗口，选择【Kernel hacking】这一项，如图 4-1：

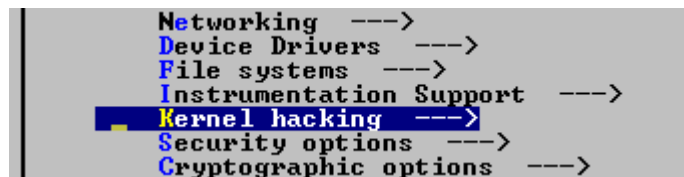


图 4-1 选中 Kernel hacking 选项

选中【GCOV coverage profiling】配置选项，图 4-2：

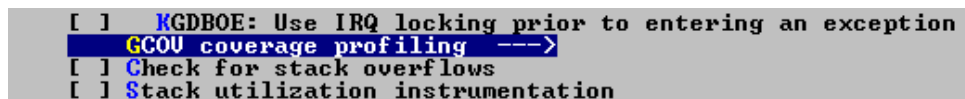


图 4-2 GCOV coverage profiling 配置项

进到 GCOV 配置页，按图 4-3 进行配置：

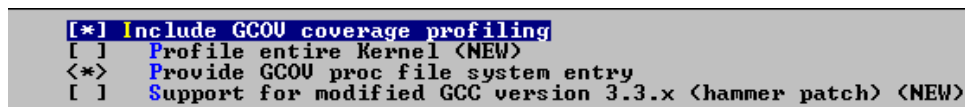



图 4-3 配置 GCOV

其中，Include GCOV coverage profiling 和 Provide GCOV proc file system entry 这两项为必选的，Provide GCOV proc file system entry 可设置为手动或静态编译。

各项含义如下：

- Include GCOV coverage profiling：包含 gcov 基本支持。
- Profile entire Kernel：评估所有内核源码。默认为不选。当用户需要对测试的目录加覆盖率开关时，则在编译规则所在文件，如 Makefile 中的编译标志里添加“-fprofile-arcs -ftest-coverage”选项。如：EXTRA\_CFLAGS +=-fprofile-arcs -ftest-coverage。

用户也可选择<\*> Profile entire Kernel，但需要注意编译后生成的内核映像原始大小。

 提示：“-fprofile-arcs -ftest-coverage”选项添加的文件一定要是编译规则所处的文件，并非全部为 Makefile 文件，如 kuClibc 模块的编译规则所在文件就为 Rules.mak，而非 Makefile。用户需准确判断后，做出正确的添加。

若内核映像原始大小超过 boot 中规定的值，可以使用下面的方法对某些目录的代码覆盖率开关进行屏蔽。（详见 4.2.2）

在其 Makefile 中初始行增加如下代码：

```
ifdef CONFIG_GCOV_PROFILE
ifdef CONFIG_GCOV_ALL
CFLAGS :=$(subst -fprofile-arcs,,$(CFLAGS))
CFLAGS :=$(subst -ftest-coverage,,$(CFLAGS))
endif
endif
```

需要注意的是，“**Profile entire Kernel**”开关仅针对内核代码，即：

✧ CGEL 3.0 KTH 内核

“**Profile entire Kernel**”开关是针对 X:\KIDE\targetkernel-version\cgel3.0-kth\linux 内的代码。

若用户只对内核代码外的模块或自行添加的模块进行代码覆盖率分析，那么这种情况不需要配置“**profile entire kernel**”，只需要对模块的编译规则所在文件，如 Makefile 中编译标志里面添加“-fprofile-arcs -ftest-coverage”，即可对该模块进行评估。

如对 kcplus 模块的 X:\KIDE\target\kernel-version\cgel3.0-kth\kcplus\Makefile，修改其中两行编译参数，C++FLAGS\_PARA 和 CFLAGS\_PARA：

```
C++FLAGS_PARA += -fno-common -fprofile-arcs -ftest-coverage
CFLAGS_PARA += -fno-common -DIN_LIBGCC2 -fprofile-arcs -ftest-coverage
```

则可以进行代码覆盖率分析。

✧ CGEL 3.0 NO KTH 内核

“**Profile entire Kernel**”开关是针对 X:\KIDE\target\kernel-version\cgel3.0\linux\内的代码。

若用户只对内核代码外的模块或自行添加的模块进行代码覆盖率分析，那么这种情况不需要配置

“**profile entire kernel**”，只需要对模块的编译文件，如 Makefile 中编译标志里面添加 “**-fprofile-arcs -ftest-coverage**”，即可对该模块进行评估。

如对 kerLibc 模块的 X:\KIDE\target\kernel-version\cge13.0\LinuxkerTools\kerLibcModCreate\Makefile，修改其中一行编译参数，CFLAGS：

```
CFLAGS      := -Wall -Wstrict-prototypes -Wno-trigraphs \
              -fno-strict-aliasing -fno-common -pipe -msoft-float -Os -fprofile-arcs -ftest-coverage
```

当然，如果用户除了对模块进行分析外，还希望对内核代码也进行分析，那么 “**profile entire kernel**” 也是需要配置的。



提示：若是对模块进行代码覆盖率分析，除修改编译文件外，一定要保证内核配置了该模块，才能使得编译生效，获得代码覆盖率文件。

➤ Provide GCOV proc file system entry: 提供 proc 文件系统方式访问代码覆盖率数据，即提供目录 /proc/gcov，用于访问操作代码覆盖率数据文件。

✧ 当选择为 M 时，则编译为 gcov-proc 模块，存放路径为：

\bspConfig\object\images\modules\linux\kernel\gcov\gcov-proc.ko

✧ 当选择为 Y 时，gcov 的运行配置使用默认参数 gcov\_link=1，gcov\_persist=0。

当选择为 M 时，可使用 modprobe 命令来设置 gcov 的运行参数。gcov 运行参数见后面描述。

➤ Support for modified GCC version 3.3.x (hammer patch) (NEW): 仅在使用 gcc 版本为 3.3.x 时选择该项。该选项可能会导致编译时错误或/proc/gcov 中文件格式错误。

也可直接修改配置文件：

```
#
# GCOV coverage profiling
#
CONFIG_GCOV_PROFILE=y
CONFIG_GCOV_ALL=y
CONFIG_GCOV_PROC=y
CONFIG_GCOV_HAMMER=n
```

➡ 提示：内核的调试选项可能会影响 gcov 分析结果，如果开启了 oprofile 或者 kernel hacking 选项，必需重新进行代码覆盖率支持的内核配置后才能进行代码覆盖率分析。

➤ 不能勾选 Loadable module support->Module versioning support。

➡ 提示：开启 GCOV coverage profiling 开关后，若需要建立在此 LSP 项目上的 kernel 或 module 项目，则需要开启【Loadable module support】->【Enable loadable module support】和【module unloading】。

## 4.2.2. 开启或屏蔽内核目录或文件代码覆盖率开关的方法

### 4.2.2.1. 开启内核目录或文件代码覆盖率开关的方法

1. 若在内核配置时，未开启内核代码覆盖率开关，用户可以使用下面的方法开启特定目录的开关，即在目录下的编译规则所在文件中，如 Makefile 中相应位置处添加“-fprofile-arcs -ftest-coverage”编译选项，如下例所示：

```
ifdef CONFIG_GCOV_PROFILE
EXTRA_CFLAGS = -fprofile-arcs -ftest-coverage
endif
```

➡ 提示：由于每个 Makefile 文件的编写方式不同，添加“EXTRA\_CFLAGS = -fprofile-arcs -ftest-coverage”并非对所有编译规则所在的文件都适用。重点是需要将编译选项“-fprofile-arcs -ftest-coverage”加入，但其具体位置和加入方式都应视具体情况而定。若编译结果并无覆盖率数据文件生成，请检查编译规则，是否将“-fprofile-arcs -ftest-coverage”参数传入，使得编译生效。

如：

只针对 linux/net/atm 目录下的所有文件进行代码覆盖率分析，可以在 linux/net/atm 目录下的 Makefile 中合适位置做如下添加：

```
ifdef CONFIG_GCOV_PROFILE
EXTRA_CFLAGS = -fprofile-arcs -ftest-coverage
endif
```

此时编译出来的内核文件中只有该目录下的所有文件配置了 kcov 参数，会生成.gcno 文件。


2. 用户可用下面的方法开启**特定文件**的开关，即在文件所在目录下的编译文件中的合适位置添加：

```
ifdef CONFIG_GCOV_PROFILE
CFLAGS_filename.o = -fprofile-arcs -ftest-coverage
endif
```

如只针对 linux/kernel/cpu.c 文件，可以在该文件所在目录 linux/ kernel 的 Makefile 中做如下添加：

```
ifdef CONFIG_GCOV_PROFILE
CFLAGS_cpu.o = -fprofile-arcs -ftest-coverage
endif
```

此时编译出来的内核文件中只有该文件配置了 kcov 参数，会生成.gcno 文件。

 **提示：**内核启动的一些关键文件，如 arch/ppc/kernel/setup.c、arch/powerpc/kernel/setup\_32.c、arch/powerpc/kernel/cputable.c 不能对它们添加 kcov 参数。

#### 4.2.2.2. 屏蔽内核目录代码覆盖率开关的方法

若是在内核配置时，已开启了对整个内核代码覆盖率的开关，而由于内核映像太大等诸多原因，又期望屏蔽某些目录的代码覆盖率开关，则可以通过如下方式屏蔽某些目录的代码覆盖率开关。即是在这些目录的编译规则所在文件中的合适位置添加如下代码：

```
ifdef CONFIG_GCOV_PROFILE
ifdef CONFIG_GCOV_ALL
CFLAGS := $(subst -fprofile-arcs, $(CFLAGS))
CFLAGS := $(subst -ftest-coverage, $(CFLAGS))
endif
```

```
endif
```

如对 fs 目录去掉覆盖率开关:

CGEL 3.0 KTH 内核: 在 X:\KIDE\target\kernel-version\ cgel3.0-kth\linux\fs\Makefile 第一行加上上面的代码即可;

CGEL 3.0 NO KTH 内核: 在 X:\KIDE\target\kernel-version\cgel3.0\linux\fs\Makefile 第一行加上上面的代码即可。

### 4.2.3. 编译

内核配置完毕, 编译后, 将在目标文件相同的目录下生成\*.gcno 日志文件, 例如 LSP 项目, 则可以看到在/bspconfig/object/obj 目录下生成.o 以及.gcno 文件。



图 4-4 LSP-KTH 生成.gcno 日志文件

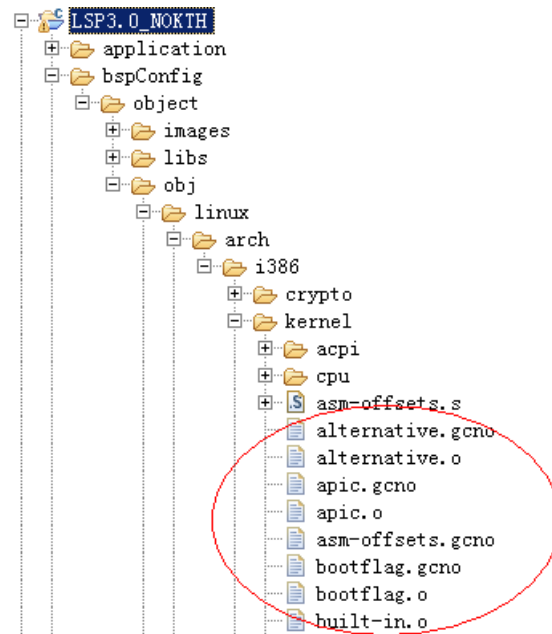


图 4-5 LSP-NOKTH 生成.gcno 日志文件

\*.gcno 是代码覆盖率日志文件，该文件记录了代码行的信息，需要妥善保存，以便 KCov 分析工具使用。

## 4.3. 生成.gcda 数据文件

### 4.3.1. 动态加载 gcov-proc 模块

内核配置时，若将【Provide GCOV proc file system entry】编译为模块，则必须在内核运行时动态加载 **gcov-proc** 模块，之后才可生成代码覆盖率数据文件\*.gcda，否则将不会生成代码覆盖率数据。

若将该选项编译进内核，则可对覆盖率数据的记录功能进行开启和关闭控制。

#### ■ 运行参数

关于 gcov 代码覆盖率模块在内核运行时动态加载，有两个参数对其运行时的行为进行控制。

1) gcov\_link: (0/1) (默认设置为 1)

如果设置为 1，在 /proc/gcov 中，源码符号链接和 gcov 图形文件随同数据文件一起创建。

符号链接对运行时没有影响，因此该选项的设置不关键。

如图 4-6: 设置了 gcov\_link 为 1 时，目录里面包含的.gcno 和.c 文件为链接文件。随同\*.gcda 生成的。



```
lrwxrwxrwx 1 root root 82 Mar 5 15:54 wait.c -> /cygdrive/d/KI
DE-2.6.20-p1b3/target/kernel-version/cgel3.0-kth/linux/kernel/wait.c
-rw-r--r-- 1 root root 2132 Mar 5 15:54 wait.gcda
lrwxrwxrwx 1 root root 85 Mar 5 15:54 wait.gcno -> /cygdrive/d
/target/workspace/lsp-kth-gcov/bspConfig/object/obj/linux/kernel/wait.gcno
lrwxrwxrwx 1 root root 87 Mar 5 15:54 workqueue.c -> /cygdrive
/d/KIDE-2.6.20-p1b3/target/kernel-version/cgel3.0-kth/linux/kernel/workqueue.c
-rw-r--r-- 1 root root 4356 Mar 5 15:54 workqueue.gcda
lrwxrwxrwx 1 root root 90 Mar 5 15:54 workqueue.gcno -> /cygdr
ive/d/target/workspace/lsp-kth-gcov/bspConfig/object/obj/linux/kernel/workqueue.
gcno
#
```

图 4-6 设置 gcov\_link=1

## 2) gcov\_persist: (0/1) (默认设置为 0)

如果设置为 1，在当动态加载的内核模块被卸载后，gcov 覆盖率数据仍能保持。若要清除数据，则可使用如下命令：

```
echo 0 > /proc/gcov/vmlinux
```

若【Provide GCOV proc file system entry】配置项选择为 Y 时，gcov 的运行配置默认使用参数“gcov\_link=1，gcov\_persist=0”。

另外，可以使用 **dmesg | grep gcov** 命令来查看当前 gcov 模块的运行配置情况。

```
# dmesg | grep gcov
gcov-core: initializing core module: format=gcc_3.4
gcov-core: init done
gcov-proc: initializing proc module: persist=0 link=1 format=gcc 3.4
gcov-proc: init done
#
```

图 4-7 查看 gcov 模块运行配置情况

### 4.3.2. 数据生成、重置和目录

内核启动后，即生成代码覆盖率数据文件.gcda，默认保存在/proc/gcov 目录下，其中除了 modules 目录外，gcov 的目录结构与目标文件的目录结构一致，如下。



图 4-8 gcov 目录结构

而 module 目录下，存放的为内核模块的代码覆盖率数据，其目录结构如同内核源码，结构如下（除 linux 目录）：

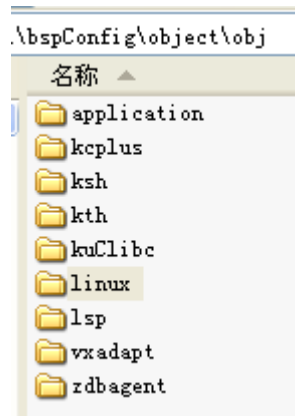


图 4-9 module 目录结构

用户可从/proc/gcov 目录中直接通过 copy 的方式将数据导出到磁盘，之后使用 KCov 工具进行数据分析，具体方法参见 5。

对内核执行操作后，相关文件的代码覆盖率数据将被更新。

#### 4.3.2.1. 重置覆盖率数据

若希望对指定文件的覆盖率数据进行重新收集，只需要使用如下命令即可实现：

```
echo 0 > 文件
```

例如：

```
echo 0 > /proc/gcov/kernel/signal.da
```

若希望重置所有的覆盖率数据，则使用如下命令：

```
echo 0 > /proc/gcov/vmlinux
```

注意此处的重置覆盖率数据，如果 gcov-proc 模块是动态加载的，且使用的默认启动方式（即 gcov-persist=0），只需要卸载 gcov-proc 模块，数据就可以清空。但如果使用的 gcov-persist=1，卸载 gcov-proc 后，数据会保持，即是说再次加载 gcov-proc 模块时，数据会叠加。如果要清空数据，使用 **echo 0 > /proc/gcov/vmlinux**。

如果 gcov-proc 模块是内核启动时加载的，默认使用的 gcov-persist=0，当然无法卸载 gcov-proc 模块，所有若要清空数据，使用 **echo 0 > /proc/gcov/vmlinux**。

#### 4.3.2.2. 数据生成目录

代码覆盖率模块的数据保存使用到了 proc 文件系统，对生成数据进行管理。根据 proc 文件系统的特性，只存储在内存当中，并不占用外存空间，所以一旦内核重启后，数据会重新生成。

在 gcov-proc.c 文件中，定义了生成的代码覆盖率数据在 proc 文件系统的 root 目录路径，


```
#define GCOV_PROC_ROOT "gcov"
```

这个目录路径是可以由用户指定的，通过修改代码在内核配置中指定该目录。但仅限于在 proc 文件系统中指定该目录。

## 5. 使用 KCov 分析代码覆盖率

### 5.1. 使用图形界面 KCov

#### 5.1.1. 启动 KCov

若是集成在 KIDE 中的 KCov，可以通过启动 KIDE，在主菜单，选中【项目】->【KCov】，或者是在主工具条，点击，也可以通过在 Windows 开始菜单中 KIDE 目录下启动 KCov，或是进入 KIDE 安装目录点击 startkcov.exe 开启 KCov 分析工具；若是独立发布的 KCov，则可以通过点击 startkcov.exe 启动 KCov 代码覆盖率分析工具。

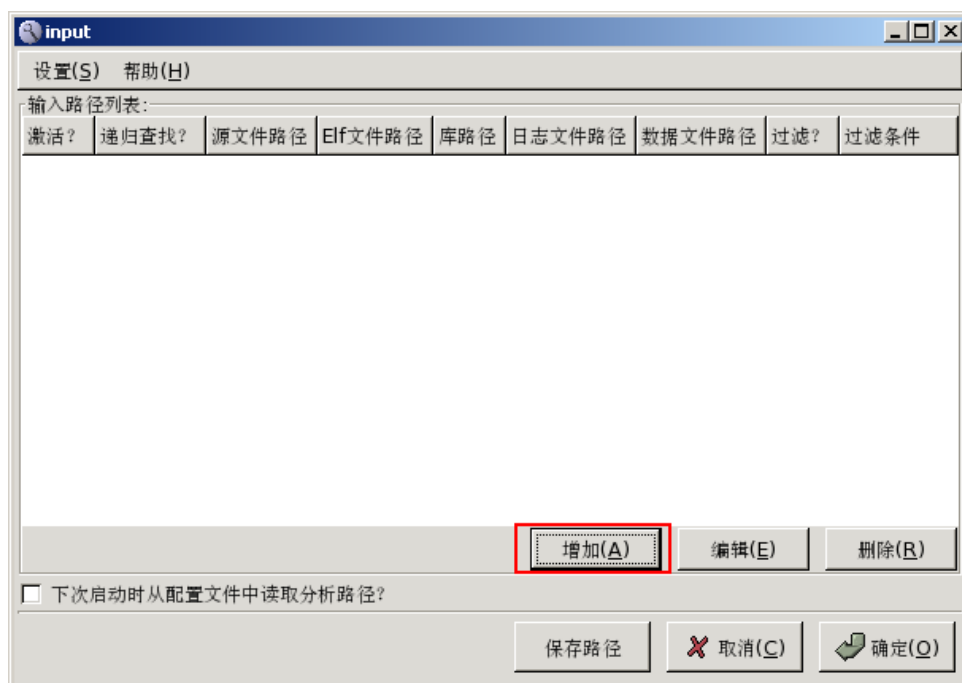


图 5-1 KCov 分析工具启动界面

在开启的 input 窗口中，可以进行有关设置代码覆盖率文件的任何操作，包括对待分析文件或目录的添加、修改和移除操作。该操作不仅可以通过界面按钮实现，同时也有一套相应的右键菜单功能。一旦文件或目录设置完毕，会在 input 窗口中呈现一张分析文件列表，如图 5-2，点击**确定**，即可使用 KCov 工具进行代码覆盖率分析。

路径列表的第一项“激活”按钮，可对“输入路径列表”中添加的路径是否生效进行控制，即是若

勾选“激活”，则此次 KCov 会对其进行代码覆盖率分析，否则将不予以进行分析。

点击**保存路径**，可以将当前 input 窗口中写入的路径列表信息记录保存到配置文件中，若同时又勾选“下次启动时从配置文件中读取分析路径”（该选项也可在“首选项”窗口中设置），则在下次启动 KCov 时，会从配置文件中直接读取路径进行分析，否则，则是启动一个初始的 input 窗口，需要用户再次手动输入待分析路径列表。保存当前分析路径到配置文件中，会对上次保存的信息进行覆盖。

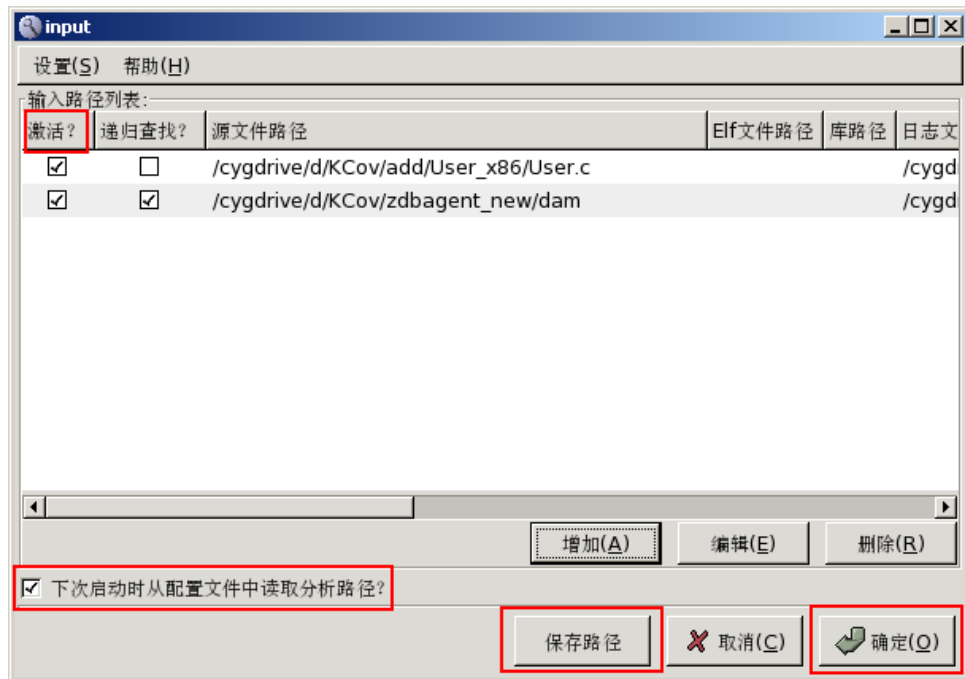


图 5-2 分析文件列表

### 5.1.2. 添加待分析文件或目录

有两种方式可用于添加需要分析的文件或目录，一种方法是在 input 输入窗口点击**增加**，另一种方法则是在 input 窗口内“输入路径列表”显示项任意位置点击右键，选择“增加”选项，如图 5-3，即可开启 Input information 窗口添加需要分析的文件或目录，如图 5-4。

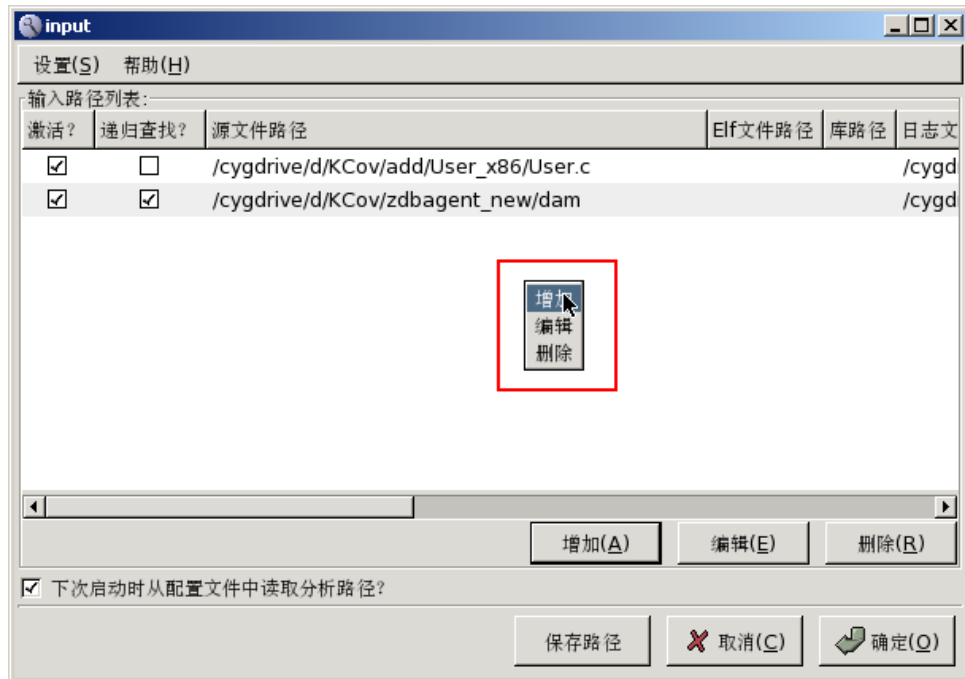


图 5-3 开启添加窗口

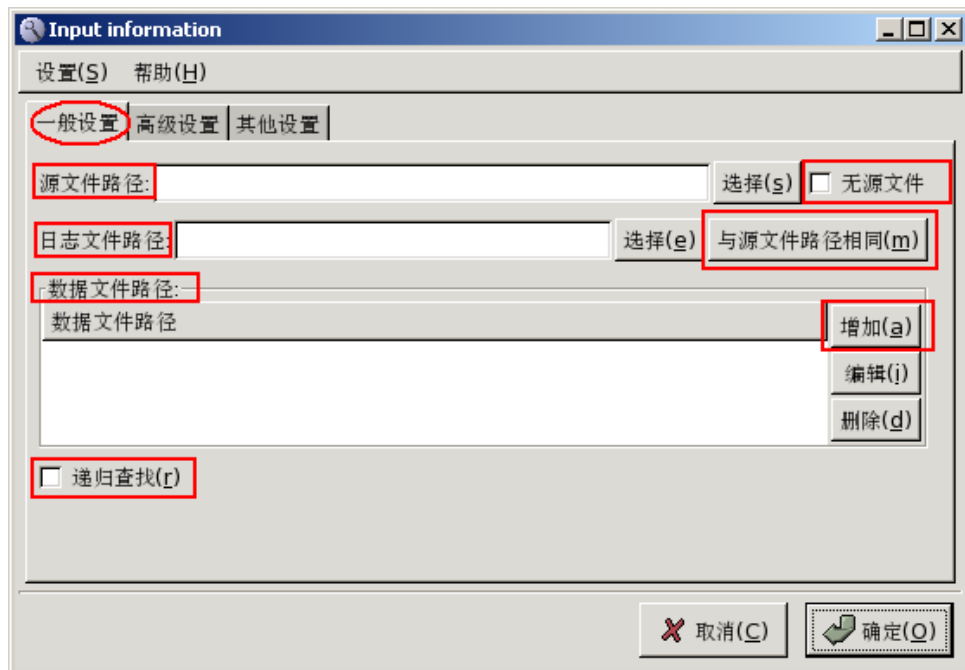


图 5-4 添加需分析文件

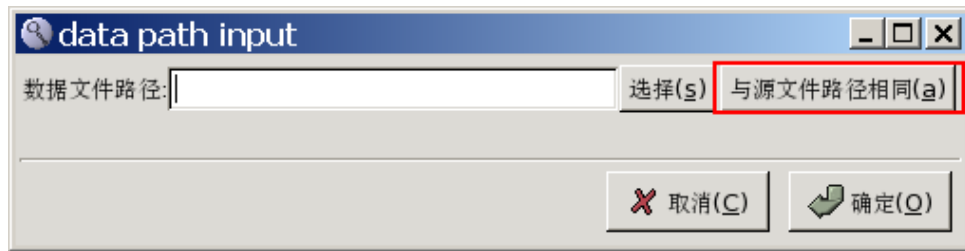


图 5-5 data path 设置

对于添加待分析文件的设置方式，除了图 5-4 中所介绍到的基本设置外，还可以进行高级设置，如图 5-6，以及其他设置，如图 5-7。

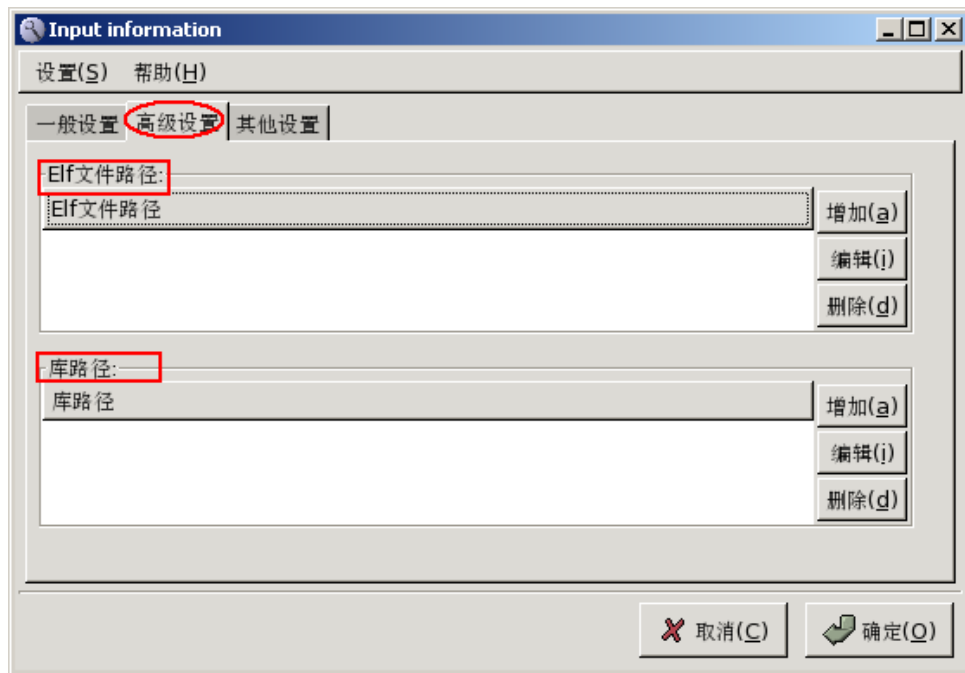


图 5-6 高级设置待分析文件

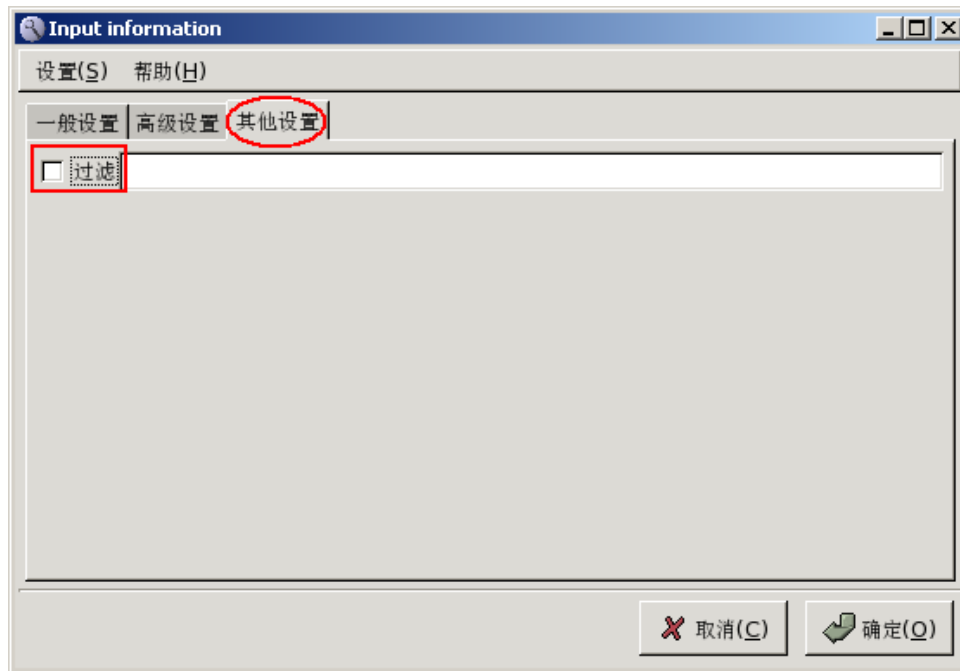


图 5-7 其他设置待分析文件

## ■ 一般设置

- **源文件路径：**导入源码文件或目录，也可以勾选“无源文件”，则由 KCov 直接通过日志文件\*.gcno 中记录的编译路径，查找到所对应的源文件存放路径；同时，若设置的源文件错误，则也将通过编译路径来查找源文件。
- **日志文件路径：**导入.gcno 或 bbg、bb 文件或目录；
- **数据文件路径：**导入 gcda 或 da 文件或目录，可同时导入多个数据文件，实现多份数据文件的累加功能，具体参见 5.1.5，此文件列表可编辑：添加、修改或删除，这三种操作方式可通过数据文件路径窗口的右侧按钮实现，也可以点击右键菜单实现，双击已添加的数据路径，可以打开设置窗口进行修改；
- **递归查找：**递归分析目录，可按用户需求自行勾选，只对多级目录结构有效，默认不选；
- **与源文件路径相同：**当 gcno 或 gcda 文件存放位置同源码，即可点击此按钮，迅速定位路径。

➡ 提示：在目录选择窗口不应选择头文件（如.h）进行覆盖率数据查看，而应该选择与之对应的源文件。

➡ 提示：将源文件、日志文件及目标机上获取的数据文件备份进行分析时，最好拷贝完整的项目路径，



以保持完整的目录结构，便于之后成功完成分析。

若是对目录进行覆盖率分析，则必须进入到目录里层，即是说选中目录后，可对该目录下的各个文件进行查看：

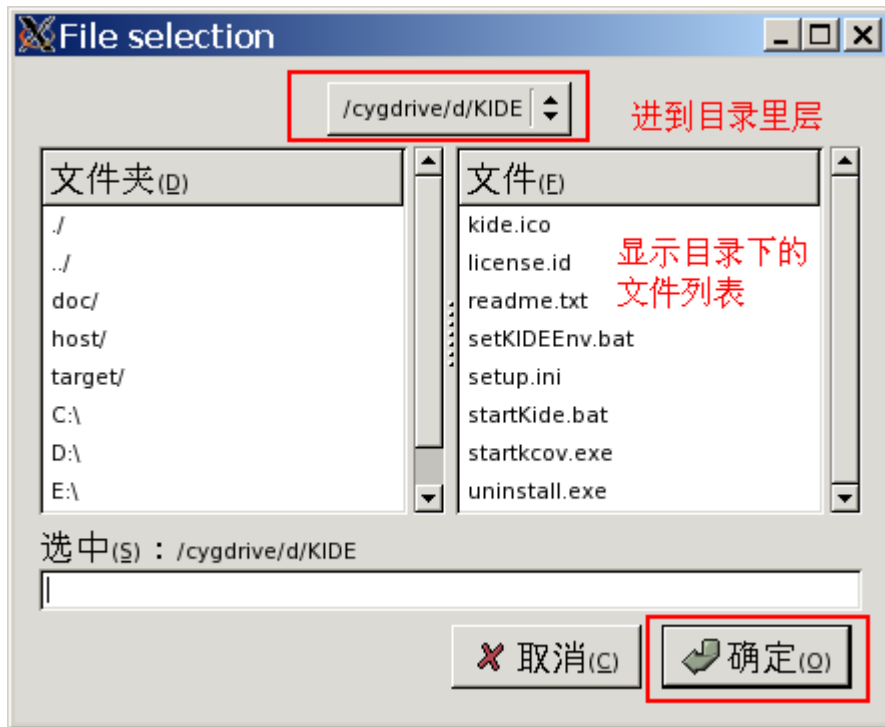


图 5-8 添加目录

在分析目录时，是否在“一般设置”中选择“递归查找”，对目录进行递归分析，所得测试结果完全不一样。尤其是当您的项目中目录结构比较多的情况下，其代码分析结果的区别尤为明显。如在不勾选“递归查找”的情况下，仅对 zdbagent\_new\dam 目录下的文件进行分析，如图 5-9，否则，还会对 zdbagent\_new\dam\ arch 下的目录文件也一并进行分析，如图 5-10

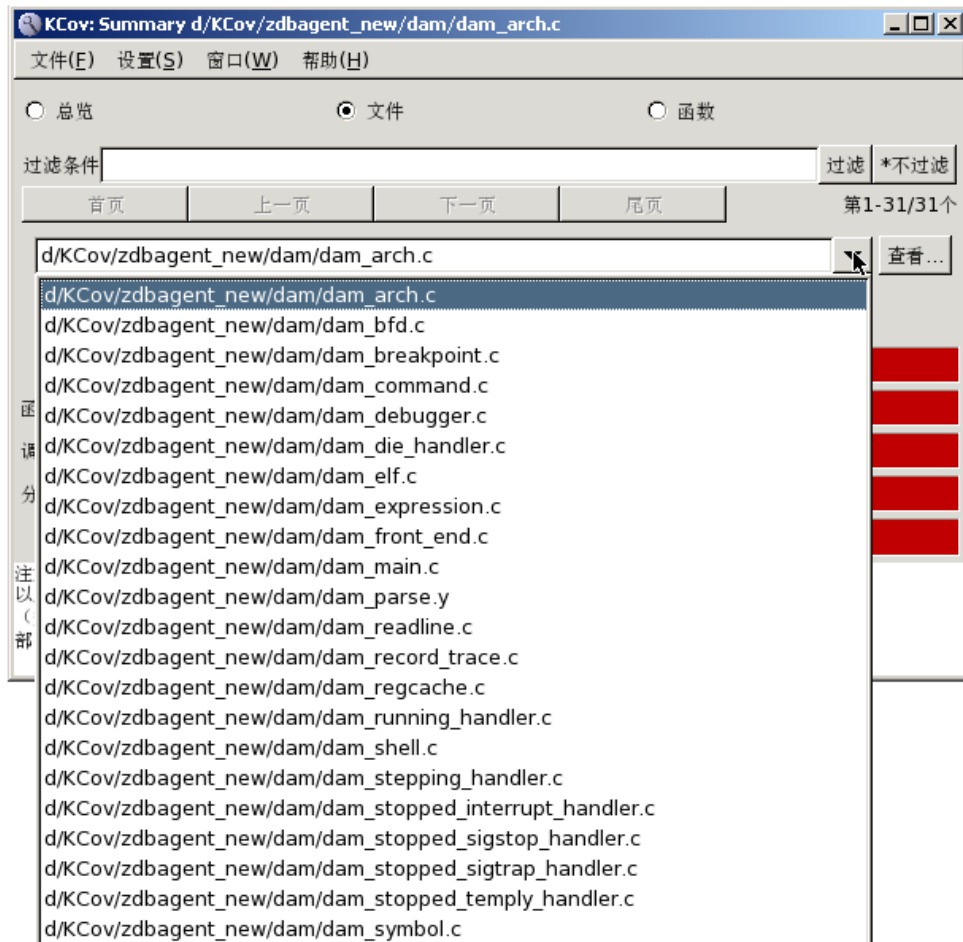


图 5-9 不递归分析目录

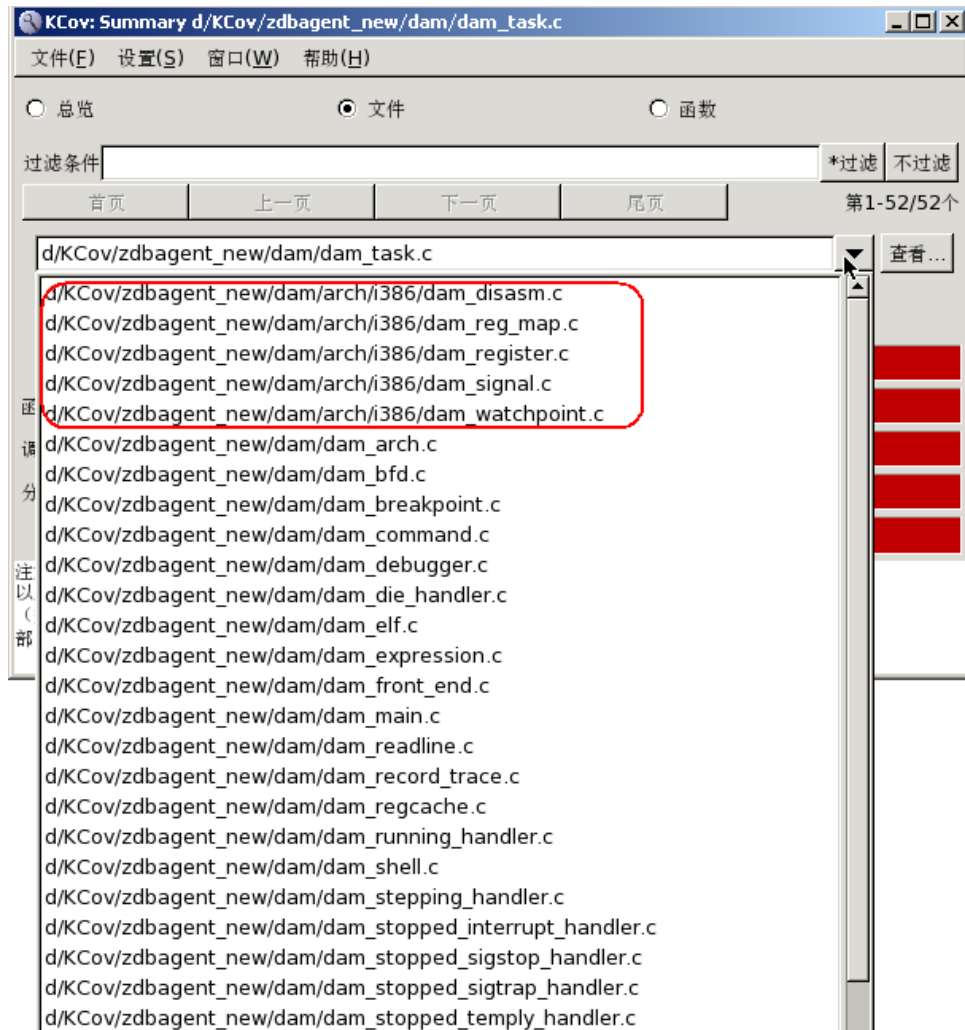


图 5-10 递归分析目录

我们可以很清晰的看到，在进行递归分析目录文件时，KCov 会深入目录下的各级目录，找到待分析的文件，逐个进行代码覆盖率分析。

## ■ 高级设置

- **Elf 文件路径：**导入 Elf 目录，通过 elf 文件来获取源码相关信息，从而在进行覆盖率解析之前，先确定解析文件的范围，与设置的“源文件路径”比对后，准确定位待解析文件，其作用在于缩小覆盖率分析文件的搜索范围，适用于处理同时有多个程序都生成了日志、数据文件，而用户仅对某个库、某个程序的覆盖率关心的情况；
- **库路径：**导入共享库路径，此选项针对用户设置的 elf 文件采用动态链接的方式，则同时对其提供动态库的覆盖率解析功能，解析出的源码也被添加到源码范围中。

## ■ 其他设置——解析过滤

针对 KCov 解析时的过滤功能，提供文件和目录的两种过滤类型。所填写的过滤条件，主要是针对“一般设置”中所设置的**源文件路径**进行过滤，但若未设置源文件路径或源文件路径设置不正确，则针对编译路径进行过滤。

编译路径是指项目编译时的实际路径（全路径），如编译时是在“D:\target\workspace\demo\\*\*\*”，则编译路径则为“/cygdrive/d/target/workspace/demo/\*\*\*”。编译路径可从目标文件中获得，同时日志文件中也记录了编译路径。

在解析时过滤，可以将符合过滤条件的源文件及其相关头文件过滤出来，从而使得解析更有针对性，大大节省了解析时间。关于过滤规则可参照 5.1.6.7。

所有项目设置完毕后，点击**确定**，返回 input 输入窗口，可以查看已添加待分析的测试目录列表。

在 input 窗口中确认待分析的路径设置无误后，点击**确定**，KCov 会依次对所输入的路径进行分析，一旦检测到疑问，出现解析错误，KCov 会对定位的错误给出确切的错误原因，而非多个可能的错误原因，从而帮助用户准确定位错误，大大提高工作效率。如图 5-11 是分析路径 1 时检测到的问题：

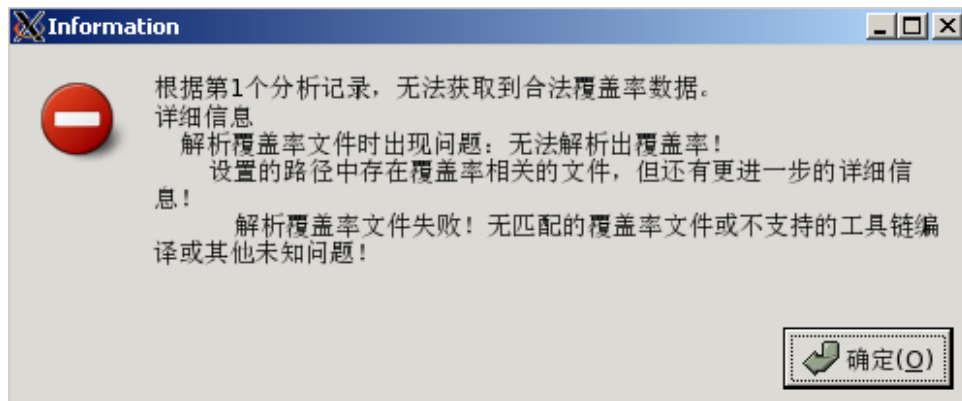


图 5-11 路径 1 错误提示

之后，继续按 input 窗口中文件列表顺序依次分析，如遇问题，依然会给予错误提示，如图 5-12。

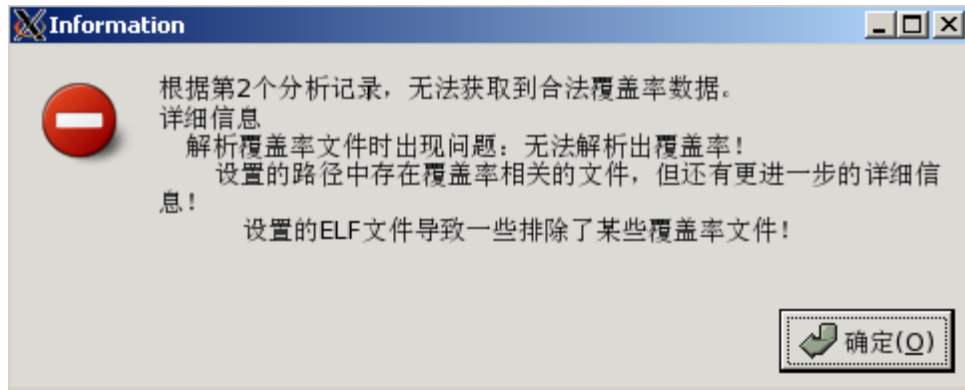


图 5-12 路径 2 错误提示

在整个分析过程, 会有进度条显示, 如图 5-13。一旦在分析过程中, 点击进度条上的**取消**按钮, 则会放弃此次 KCov 的分析, 同时由于数据的重写, 之前路径的解析也将缺失, 无法挽回。

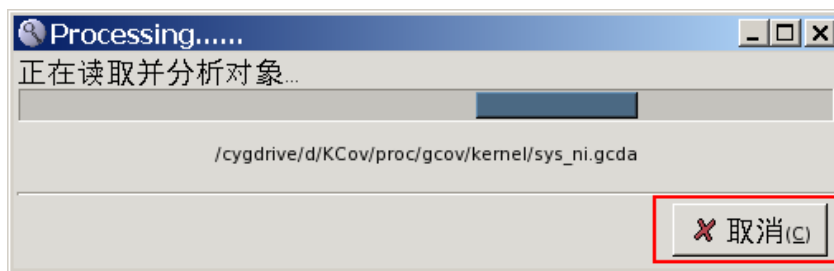


图 5-13 分析过程进度条显示

所有的路径分析结束后, 则弹出包含了对所有正确路径分析结果的总览窗口, 如图 5-14

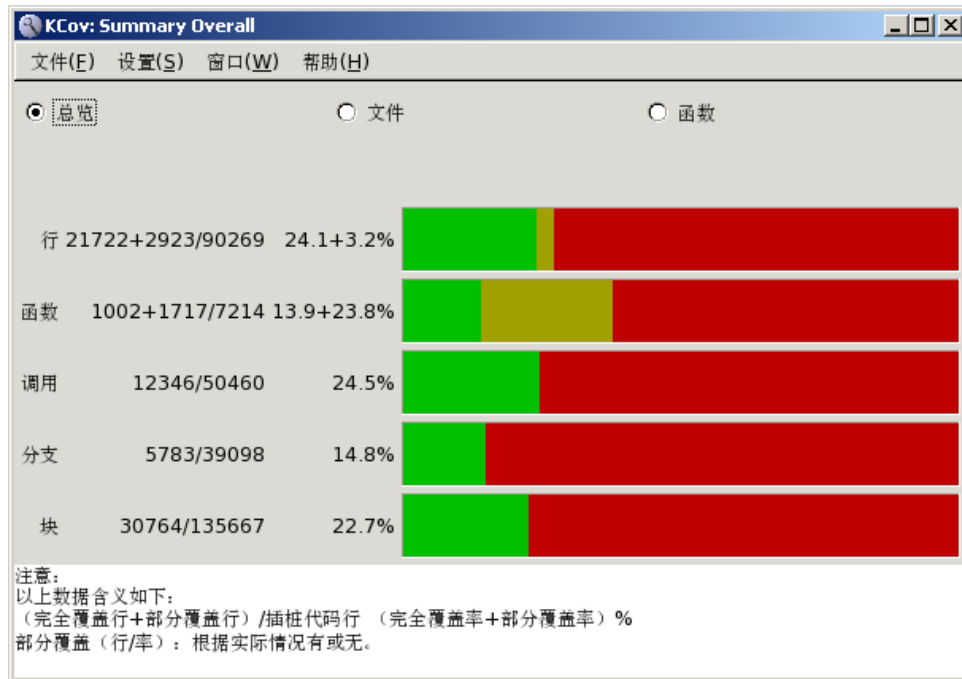


图 5-14 总览窗口

### 5.1.3. 修改待分析文件或目录

若希望对已经设置好待分析的文件或目录进行修改，可在 input 窗口中，选中需进行修改的文件，之后点击 **编辑**，或是右键选中需修改的文件，在右键菜单中选择“编辑”选项，也可以直接双击待修改的文件，在弹出的文件设置窗口中按照需求，自行对源文件路径、日志文件路径或数据文件路径进行修改。

### 5.1.4. 增量添加分析文件或目录

在对选定的文件或目录进行代码分析的同时，可以增量添加新的文件或目录进行分析。只需在分析窗口，点击【文件】->【打开】，即可开启 KCov 代码覆盖率数据的设置窗口，选择新的文件或目录添加到分析列表中。

1. 例如，当前，我们仅对 thread.c 进行分析：

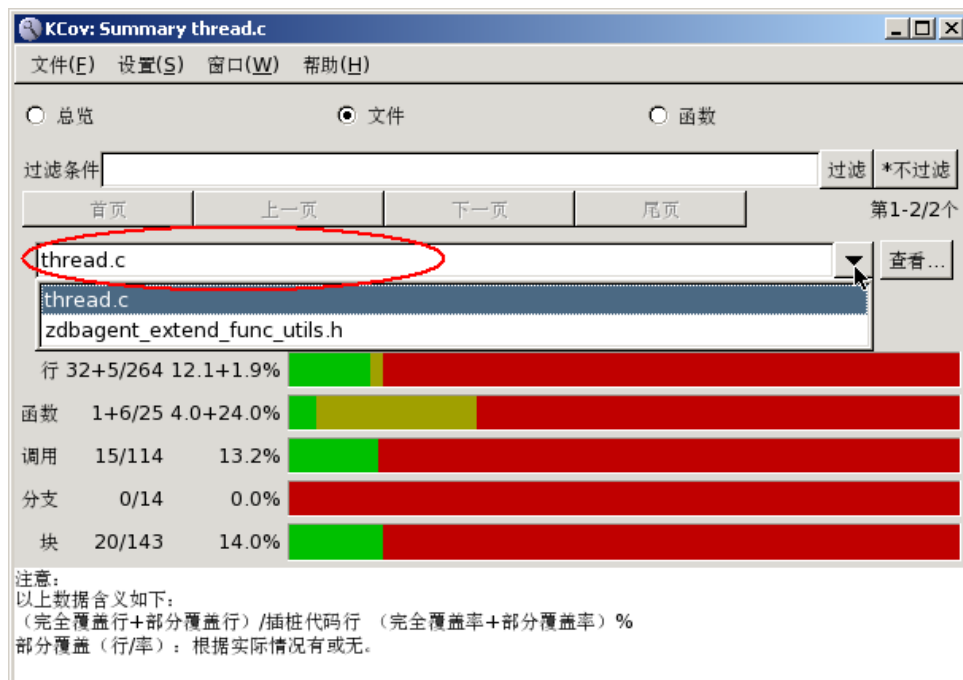


图 5-15 初始文件分析

2. 点击【文件】->【打开】，加入新的待分析文件 User.c。

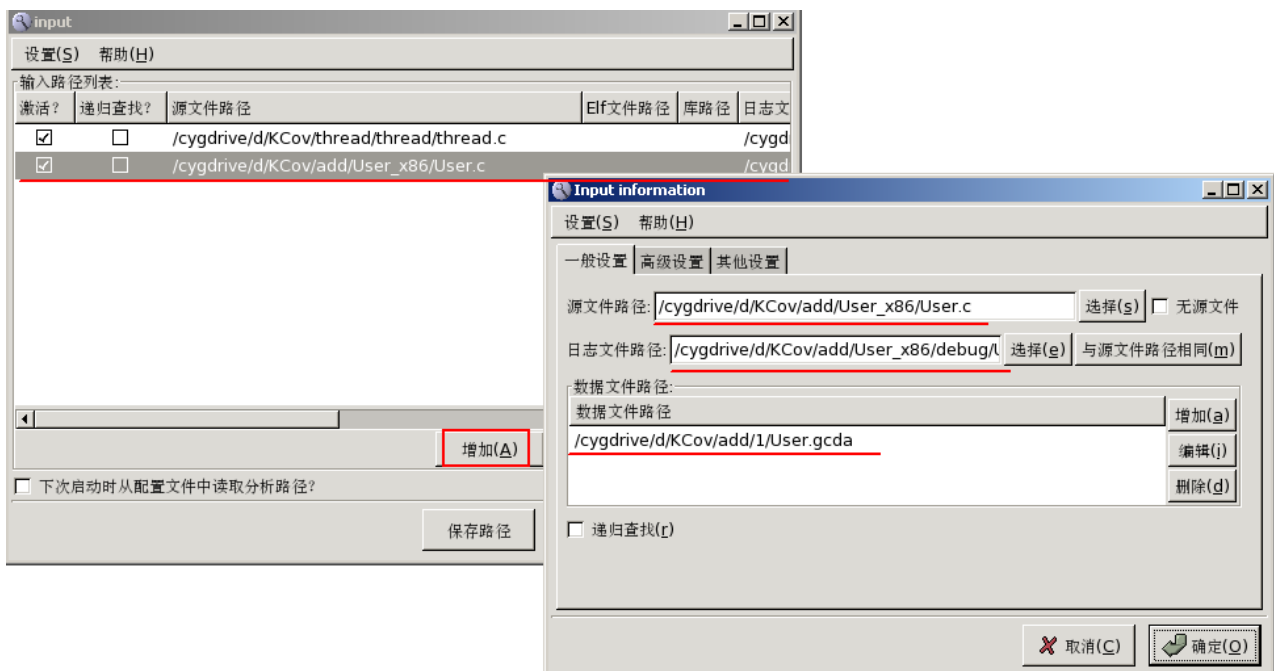


图 5-16 新增 User.c 分析文件

➡ 提示：当点击【文件】->【打开】开启文件选择框后，必须结束“文件添加配置窗口”设置后，才可以进行其他操作，否则，即使将文件选择框进行最小化，运行于后台，对其他窗体的操作也是无效的。

3. User.c 文件被加入到分析列表中：

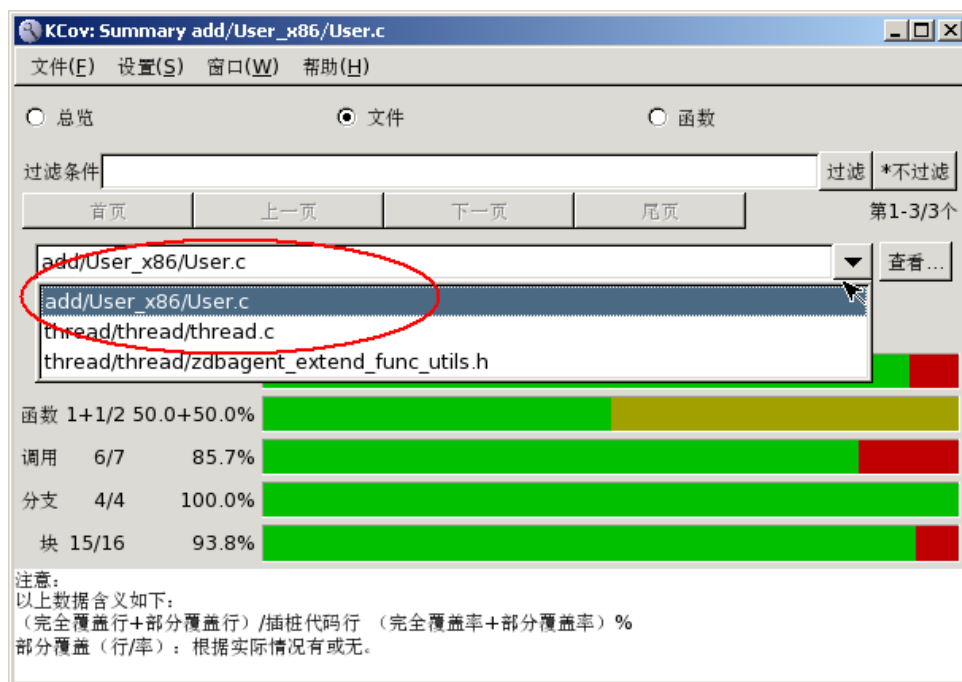


图 5-17 新增文件已加入分析列表进入分析

### 5.1.5. 覆盖率数据累加

对于功能复杂的项目，一次性不可能完成所有代码行的测试，因而如果能将多次运行产生的覆盖率数据进行累加统计，则能够大大简化工作量，节省测试时间。KCov 则应需求，提供了覆盖率数据的累加功能，即是支持多次运行应用程序时，将本次运行涉及到的覆盖率数据在前一次运行产生的覆盖率数据的基础上进行叠加、合并，供用户分析，从而便于用户进行分布测试。

覆盖率数据的累加功能是针对同一应用程序的多次运行而言。不能对代码进行再次编译，即使重新编译完全相同的代码也不被允许。且由于是在已生成的 gcda 基础上进行的数据累加，因而前一次运行生成的 gcda 不能被删除。覆盖率的数据累加功能涉及两种情况：

- 一、相同代码的多次运行，则是将覆盖率数据进行叠加；
- 二、不同代码的多次运行，则是将覆盖率数据进行合并。



针对以上两种情况，分别给以示例帮助大家理解。

## ■ 相同分支运行

将同一应用程序运行三次，分别获取运行一次和运行三次后的两个.gcda 数据文件，使用 KCov 对这两个运行结果进行覆盖率分析，例如：

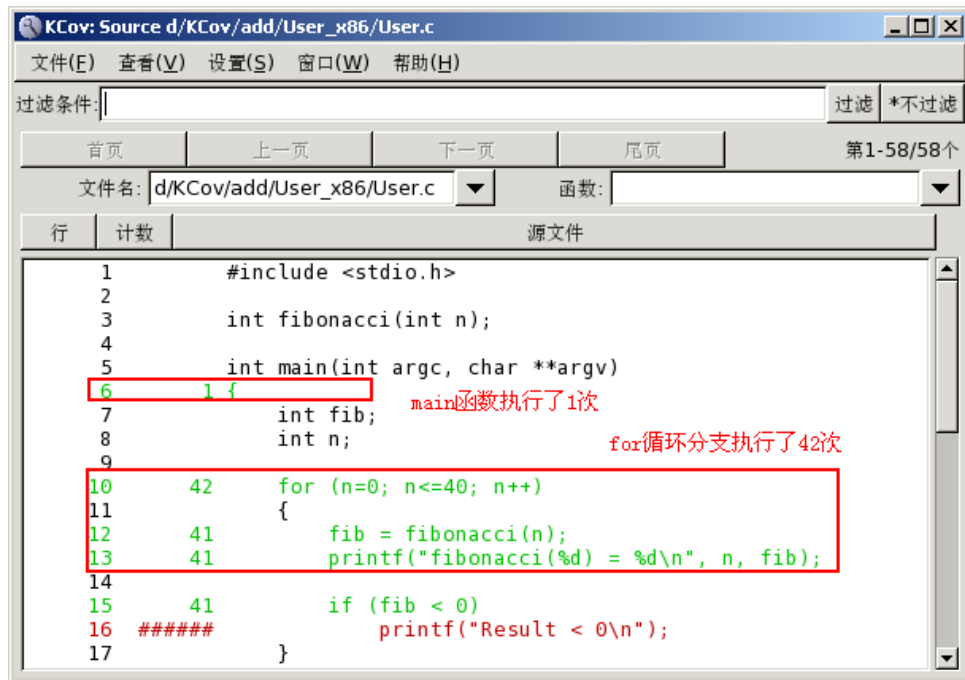


图 5-18 执行一次代码分析结果

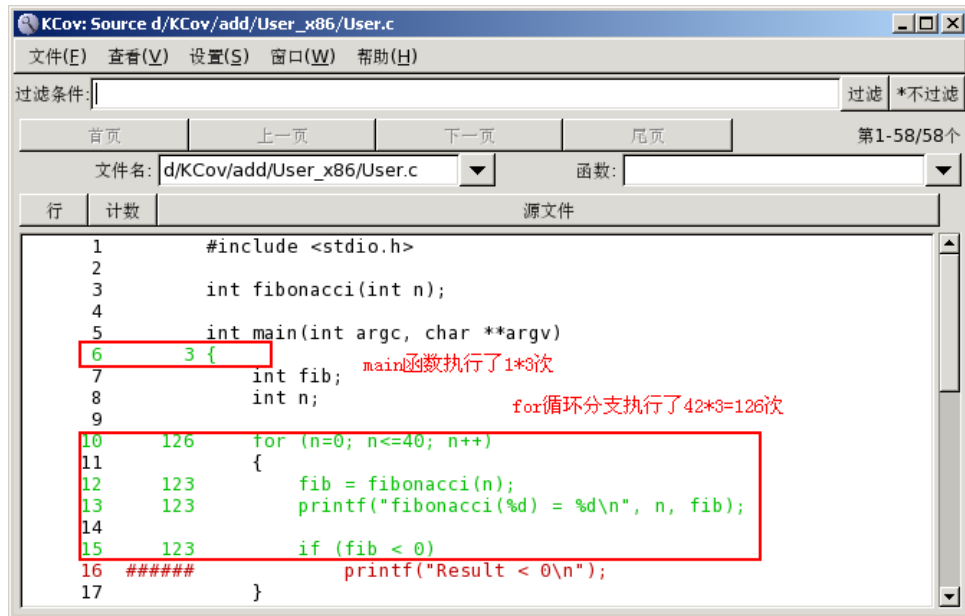


图 5-19 执行 3 次相同分支代码分析结果

我们能够很容易看出，此程序三次运行的都为相同分支，且程序运行第三次所得到的代码覆盖率的结果是第一次测试结果的三倍，表明 KCov 的确实现了对覆盖率数据进行叠加。

## 不同分支运行

编译一个应用程序后，运行一次程序，执行了程序中的一个分支，查看覆盖率分析的结果如图 5-20:

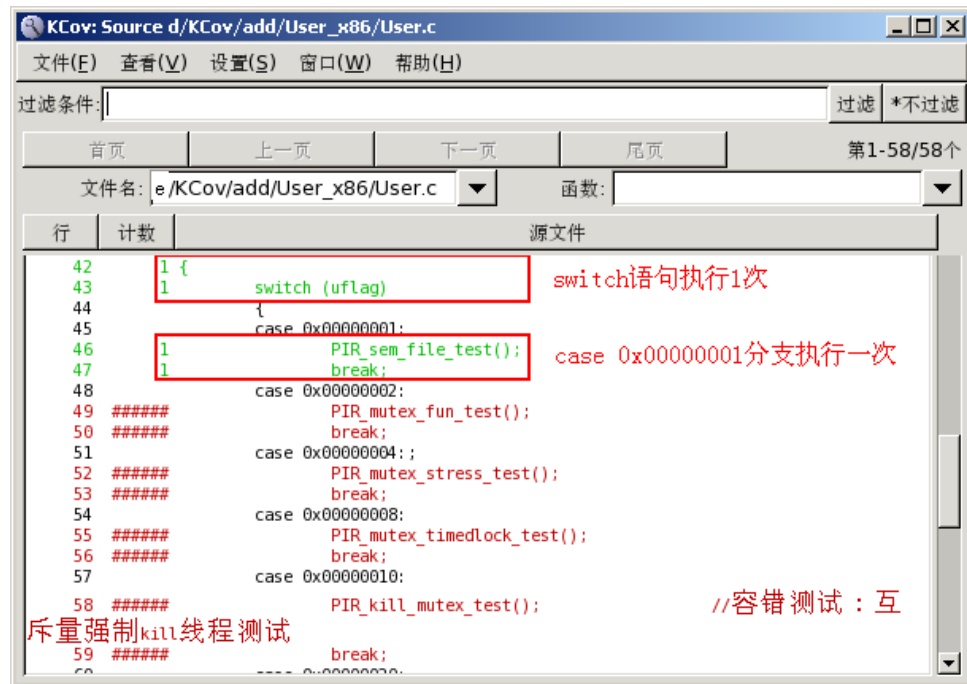


图 5-20 执行一次代码分析结果

再次运行此程序，执行了程序中的另一个分支，查看器代码覆盖率情况，如图 5-21：

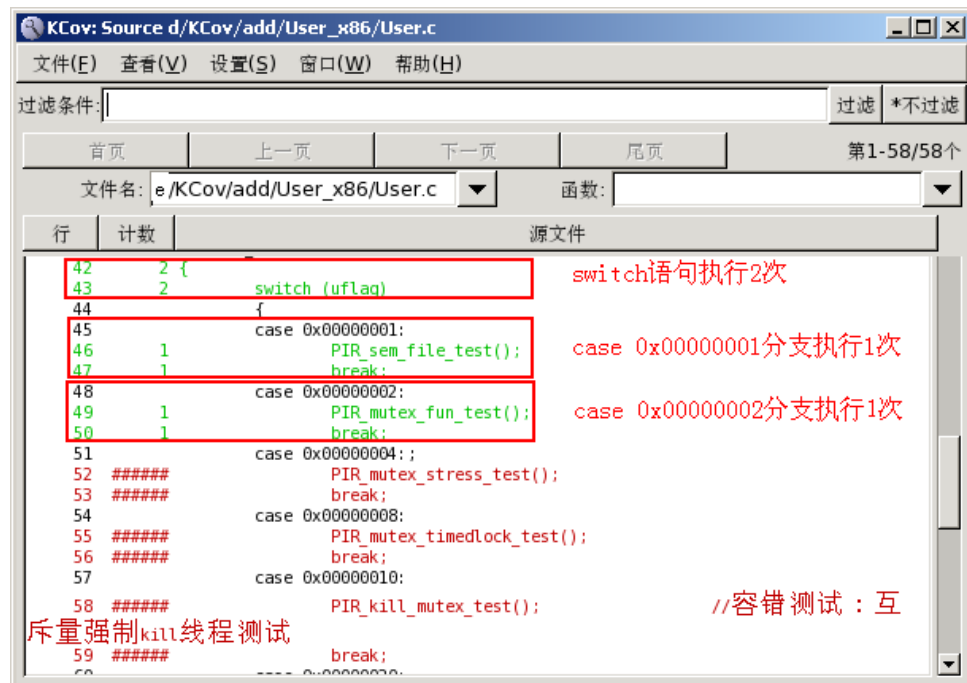


图 5-21 第二次执行不同分支代码分析结果

对比上面两图，可以很清晰的看到，第二次代码覆盖率结果是直接在第一次执行结果的基础上进行的合并。第一次运行，switch 语句被调用一次，执行了 case 0x00000001 分支；第二运行后，不仅显示执行了 case 0x00000002 分支，还同时继承了上次的运行结果，执行了 case 0x00000001 分支，因而 switch 语句被前后总执行了两次。

### 5.1.6. 解析代码覆盖率分析结果

针对用户的需求，分别提供了总览窗口、文件列表窗口、函数列表窗口、源文件窗口、报告窗口 5 大数据分析窗口供用户查看，且每一个视图中的具体查询信息显示都可以由用户自行定制，只需在各个窗口开启后，点击主菜单【查看】进行勾选设置。

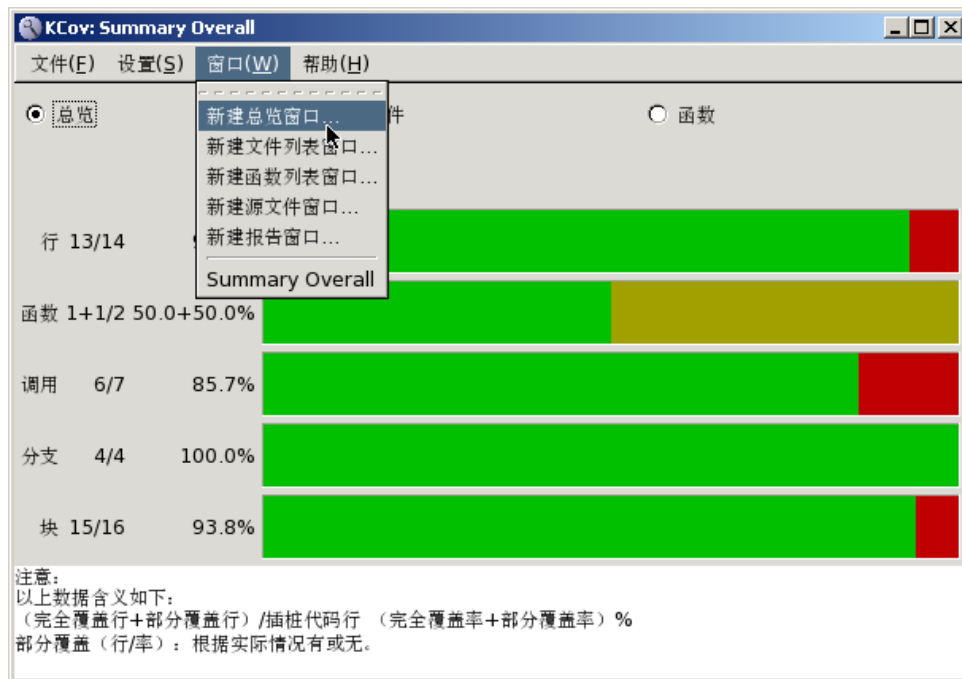


图 5-22 数据分析 Windows 窗口

由于 KCov 对于代码的覆盖率按照代码行级、函数级和逻辑块级三类进行分析，因而在对分析结果进行解析前，我们需了解在 KCov 中这几大关键要素的具体含义，以方便我们对解析数据的理解，其具体定义如下：

#### ■ 行

能产生汇编指令的行，则被定义为作为覆盖率测试的行。如图 5-23 和图 5-24 所示，分支语句在 if、switch 处，产生汇编判断与跳转指令，而 if 对应的 else，switch 对应的 case，已经包含在这些判断与跳转指令中，不产生额外的汇编指令，因此不计入行。

```

void funswitch(int n1)
3 { — 产生汇编栈帧处理指令
3   switch(n1)
    { — 不产生汇编指令
      case 861:
1       printf("861\n");
1       break;
      case 0:
1       printf("0\n");
1       break;
      default:
1       printf("default\n");
        break;
    } — 不产生汇编指令
3 } — 产生汇编返回指令

```

图 5-23 行定义示例 1

```

int fibonacci(int n)
86698883 {
    int fib; — 编译器分配空间，
86698883   if (n <= 0) 但不产生汇编指令
    {
16558014       fib = 0;
    }
70140869   else if (n == 1)
    {
26791429       fib = 1;
    }
    else
    {
43349439       fib = fibonacci(n-1) + fibonacci(n-2);
    }
86698883   return fib; return 产生了汇编返回指令
           } — 这里则不再产生汇编指令

```

图 5-24 行定义示例 2

## ■ 函数

函数的数量统计，是根据预编译后得到的源文件中包含的函数定义来计算的，而不是仅统计某个.c 源代码文件中的函数定义，内联函数和宏函数也包含在函数统计之列。

## ■ 分支

分支的定义在汇编文件基础上，一个逻辑分叉为一个分支。

## ■ 块

块是编译中的一个概念，简单的说是一个具有完整意义的最小源码语句块。常见的块的形式有：一条无嵌套的赋值语句、一次表达式判断（if/switch）、一次函数调用、一次函数头的压栈操作（函数的第

一个“{”、函数的返回（return）以及强制跳转（goto）等。

## ■ 调用

主函数（如 Main 函数），调用指函数内调用的其他函数的次数。

非主函数，调用指函数内调用的其他函数的次数+自身被调用 1 次。

### 5.1.6.1. 总览窗口

在 input 的窗口中设置完毕待分析的文件后，点击**确定**，进入 KCov 代码覆盖率分析总览窗口，如图 5-25，此窗口用于查看当前被检测代码的整体覆盖率情况，包括文件行覆盖率、函数覆盖率、函数调用覆盖率、分支覆盖率和块覆盖率等。针对各项的覆盖率情况，在 KCov 中使用了不同颜色进行标注，使得查看更为直观，且此颜色可根据用户习惯自行设置，具体参考 5.1.6.6。在针对不同覆盖率查看类型上方，均提供了一个分页显示条，以方便当存在大量文件或函数时，能够进行快速的查找与定位。同时，当文件名或函数名由于放置路径过深或者是名称过长的情况下，可使用左右键拖动进行查看，此查看功能对于所有的下拉框都有效。

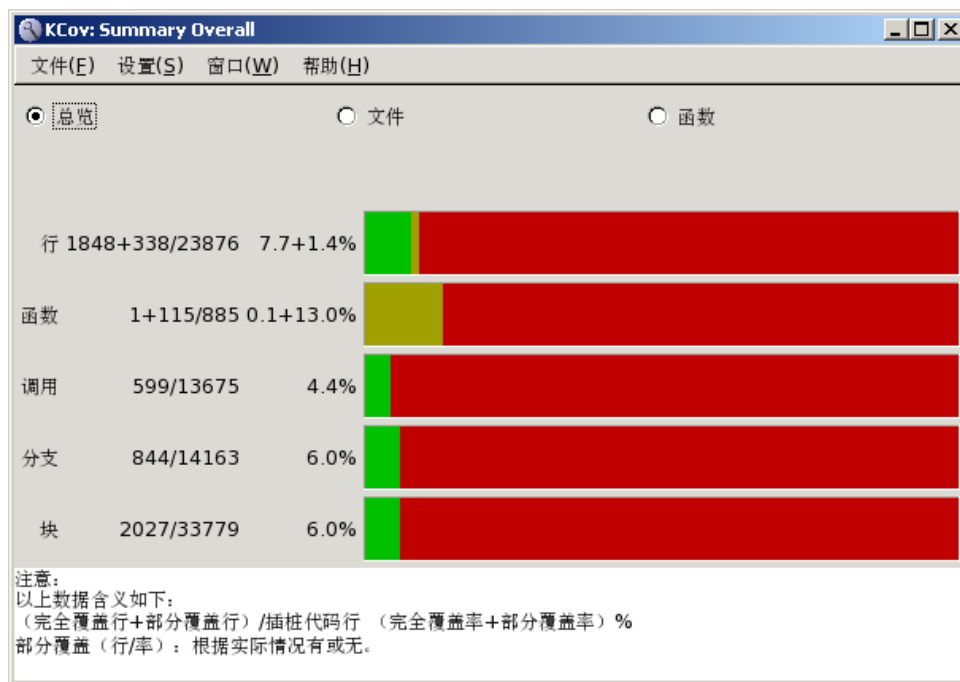


图 5-25 总览窗口

## ■ 文件列表框

当选择一个或多个文件进行覆盖率分析时，在总览窗口的【文件】中将显示所有包含覆盖率数据的相关文件。如选择了 SemTest\_U.c 和 User.c 进行分析时，【文件】项中会把所有包含覆盖率数据的相关文

件显示出来，供用户分析：

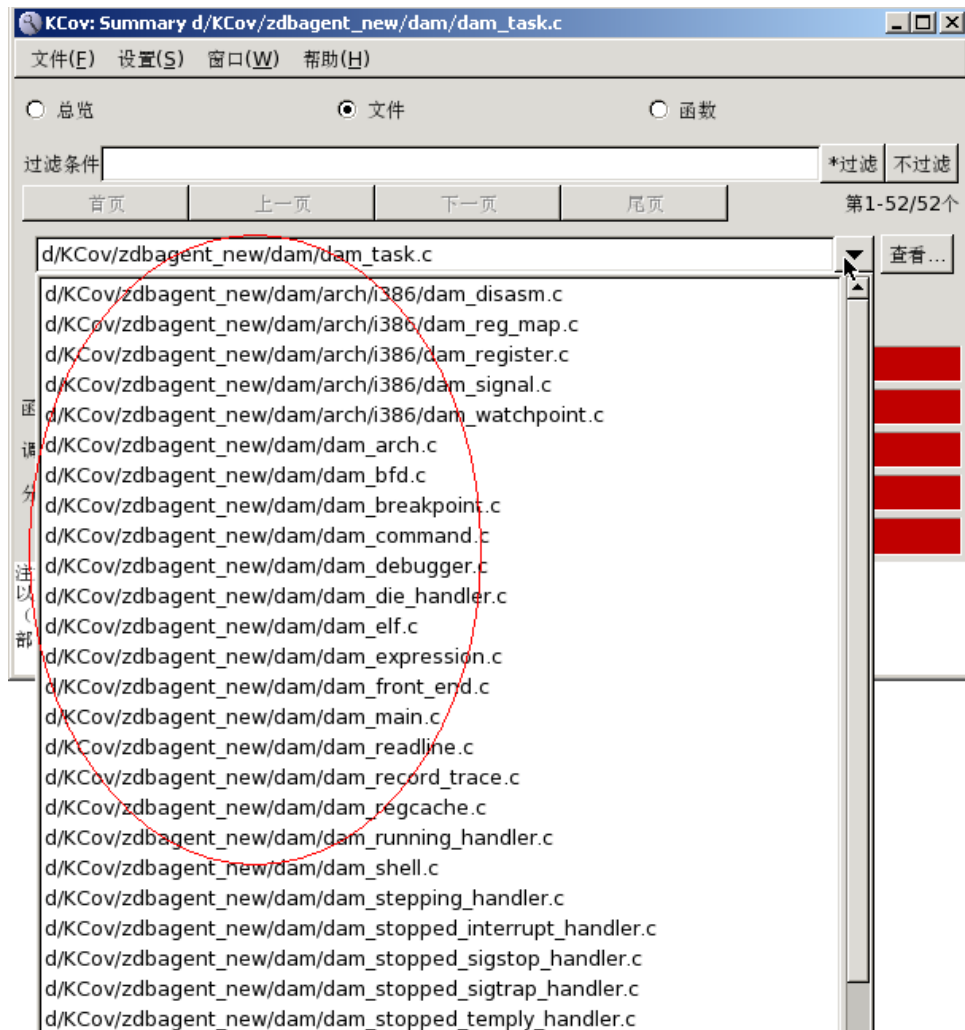


图 5-26 显示所有相关文件

选中需要具体查看的文件后，点击右侧的**查看**，弹出源码查看窗口，如图 5-27，具体参见 5.1.6.2。

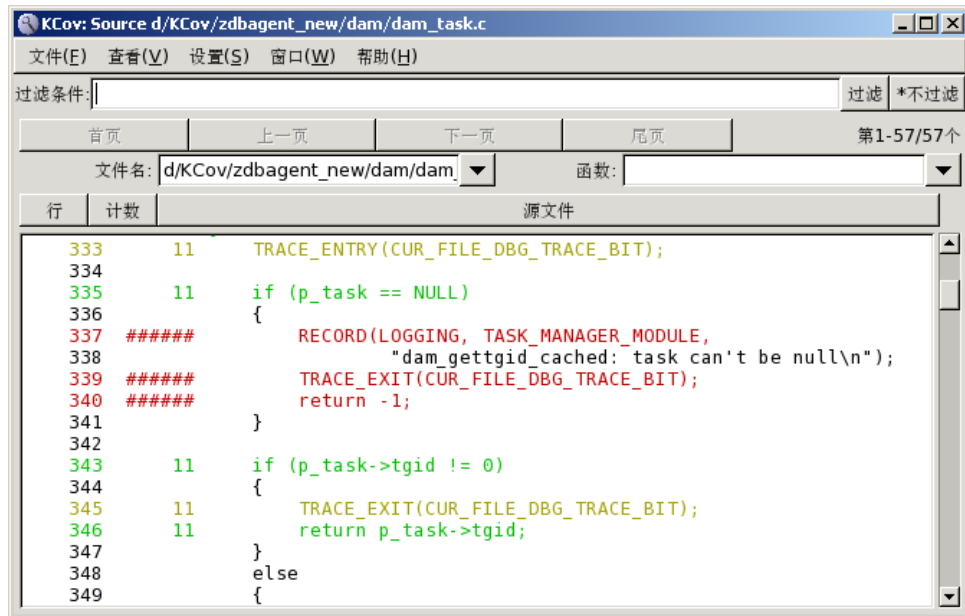


图 5-27 查看文件

➔ 提示：若找不到源文件，则“文件”项右侧的查看按钮呈现灰色，无法点击进行源文件的查看。

## ■ 函数列表框

同文件列表框，在总览窗口下的【函数】中会显示当前分析文件中所有包含覆盖率数据的相关函数，如图 5-28：



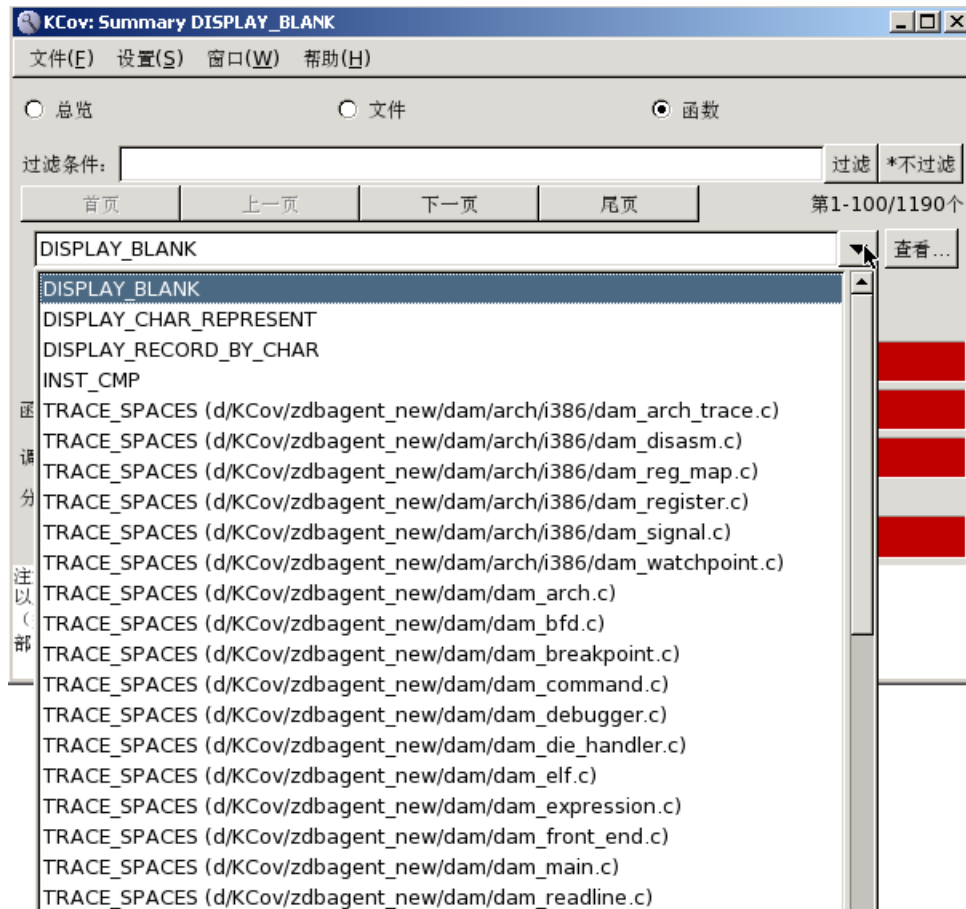


图 5-28 显示所有相关函数

选中需要具体查看的函数后，点击右侧的**查看**，弹出源码查看窗口，进行函数查看，具体参见 5.1.6.2。

但我们在函数列表框中选中 global constructors keyed to XXX 或 global destructors keyed to XXX 或 \_\_static\_initialization\_and\_destruction\_三类函数时，此刻 KCov 会弹出提示对话框，如图 5-29 所示，提示用户该函数为编译器自动生成的初始化函数，无对应的源代码，因此无法查看源文件。这是由于为支持 c++ 的代码覆盖率统计，编译器会自动生成一个全局函数，用于执行获取代码覆盖率的初始化操作。此类函数记录在 gcno 日志文件中，运行时也生成了覆盖率信息，但没有源码信息。因此，在总览窗口，不能使用“查看”按钮对此类函数源代码的查看，在函数列表窗口及源文件窗口的函数列表中选择此类函数，也不能查看其源代码。

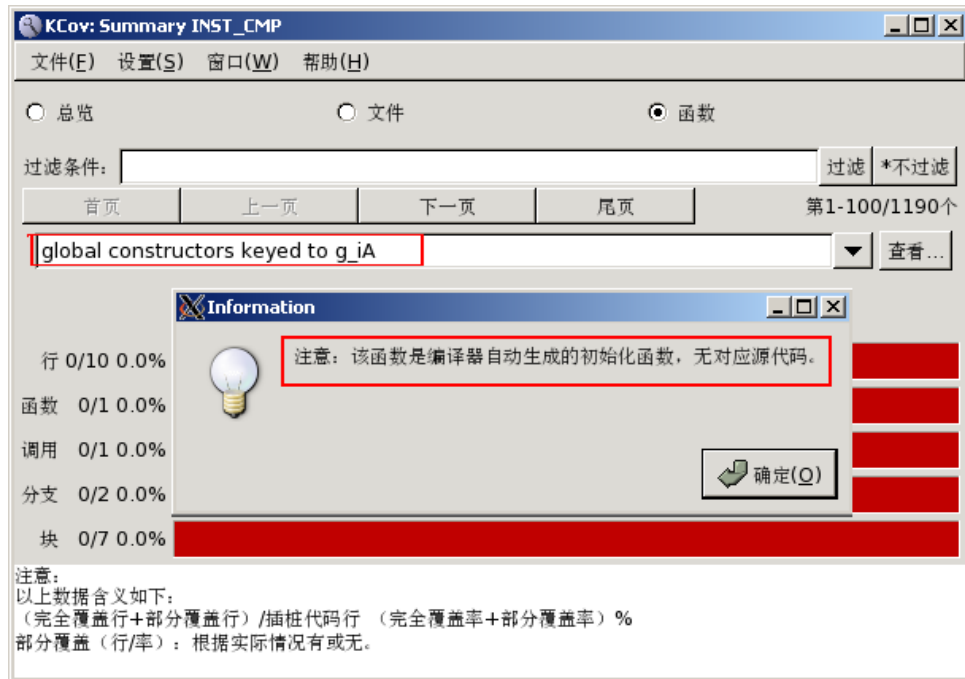


图 5-29 无法查看覆盖率情况

## ■ 解析代码覆盖率分析数值

在总览窗口下半部分，我们可以对参与检测的文件的代码覆盖率有一个整体的把控。其通过代码行、函数、逻辑块等几方面向我们展示了文件代码的覆盖率情况，现简单介绍一下其数值分布意义，以便于用户进行查看。

由于代码存在三种执行情况：完全执行、部分执行、未执行，因而通过计算被执行代码行的数值所占总代码行的比例来衡量其代码执行情况，参考图 5-25，行 1848+338/23876 7.7+1.4%，其含义为总行数为 23876，完全执行的有 1848 行，有 338 行为部分执行，因而其行覆盖率为**完全覆盖率**：1848/23876=7.7%+**部分覆盖率** 338/23876=1.4%，之后的颜色也非常直观的显示出代码覆盖率情况，且此颜色可通过设置参数的方式自行定义，具体设置方式请参考 5.1.6.6。行覆盖率、调用覆盖率、分支覆盖率及块覆盖率其计算方式相同。

### 5.1.6.2. 源文件窗口

在 KCov 代码分析过程中，提供了对源码的查看功能，以方便用户根据分析的结果对代码进行同步查看、修改。

通过点击【窗口】->【新建源文件窗口】的方式，开启源码的查看窗口，也可以如 5.1.6.1 中介绍文件和函数列表框时，及设置行选择范围时，点击右侧的**查看**打开或者在文件列表窗口和函数列表窗口中，双击行等多种方式开启源码窗口。

在源码查看窗口中，既可以查看源码文件，也可以选择与此源码相关的函数进行查看，只需先选定查看的源文件【文件名】后，在其右侧点击选择【函数】，进行文件与函数的查看切换。选定之后，则即刻在下方窗口中用灰色背景色高亮显示函数源代码，如图 5-30：

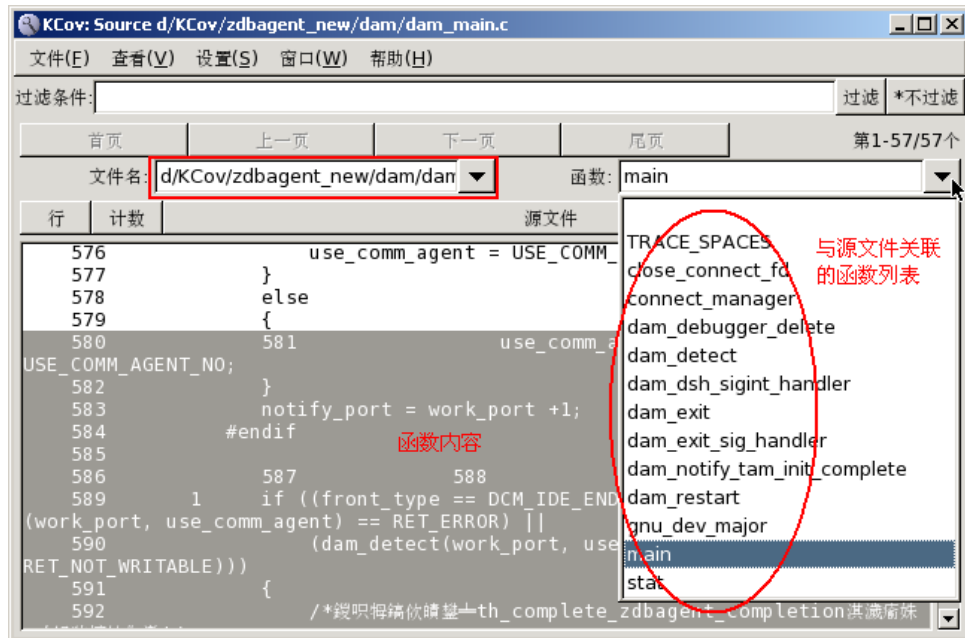


图 5-30 查看相关函数

➔ 提示：对于选择.h 文件查看，则无法显示出文件中定义的函数。

## 查看菜单

每个视图的显示都可以自行定义，点击主菜单【查看】，查看此视图提供的显示选项，根据需要进行勾选，对视图内容进行过滤显示，如图 5-31：

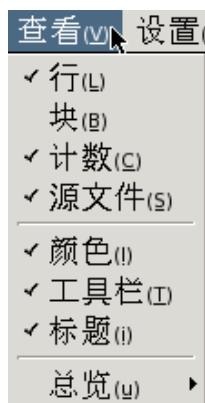


图 5-31 文件查看视图的查看菜单

针对如上设置，当前的文件查看视图显示如下：

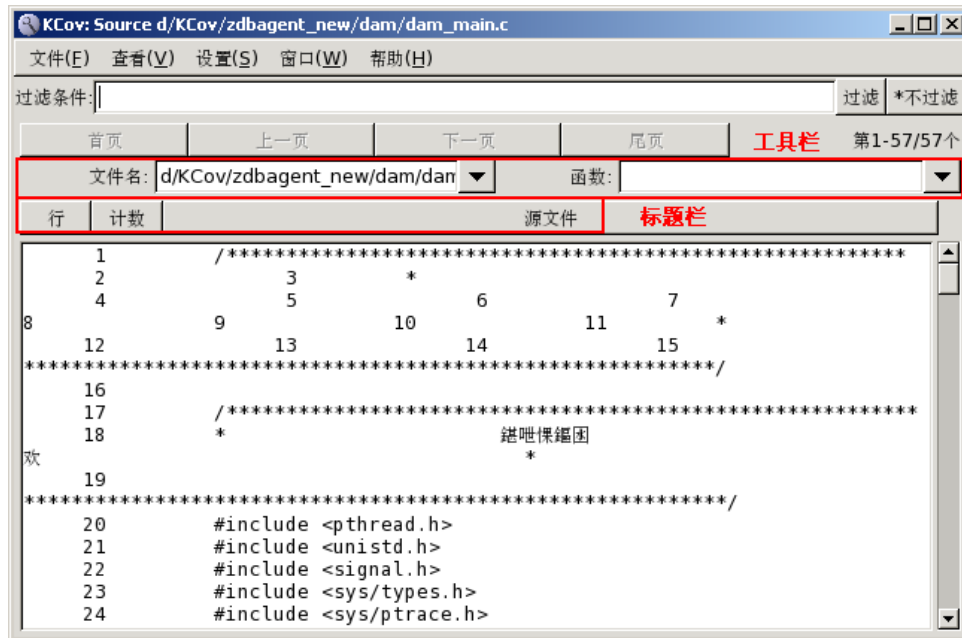


图 5-32 文件查看视图显示

- 行：显示行号；
  - 块：显示当前代码行在所属函数中的块号；
  - 计数：显示当前代码行被执行次数；
  - 源文件：显示源代码；
  - 颜色：显示代码行的不同覆盖状态用颜色标明以示区别；
  - 工具栏：显示工具栏；
  - 标题：显示标题；
  - 总览：针对文件、函数两类显示其总体信息，这需要与当前源码窗口选择的查看类型相一致。
- ✧ 若当前选择查看文件方式，如图 5-33，则点击【查看】->【总览】->【文件】，会弹出一个新的针对此文件信息的总览窗口，如图 5-34：

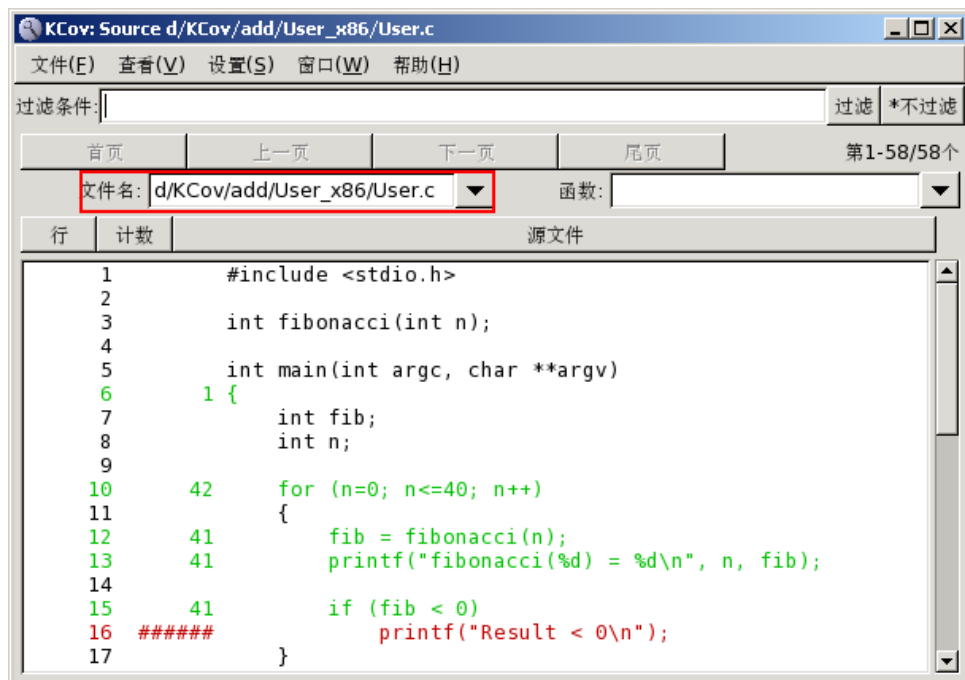


图 5-33 查看源文件

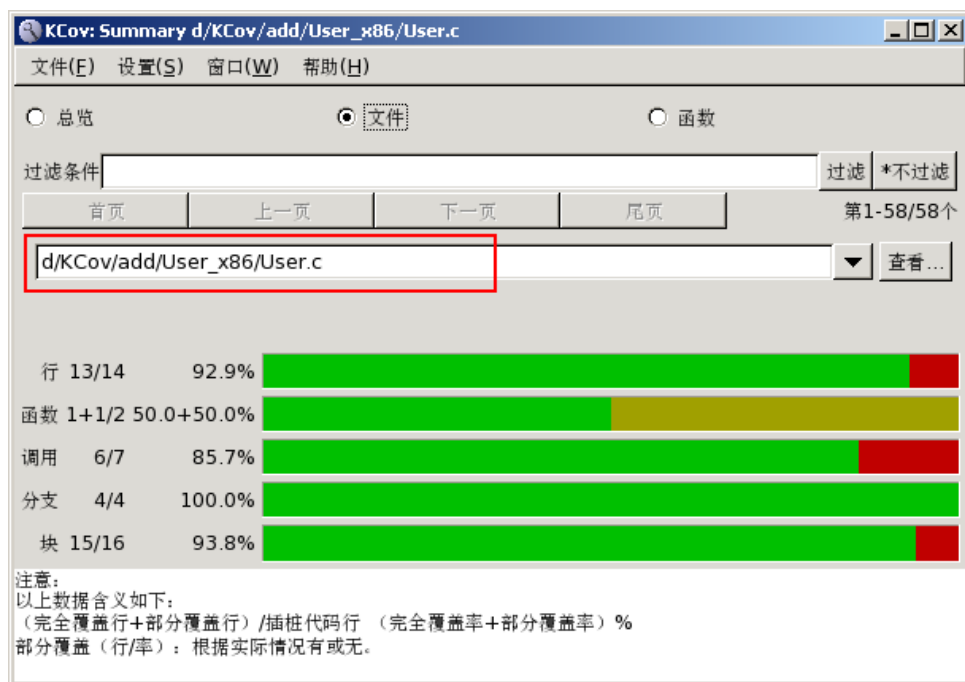


图 5-34 显示“文件”信息总览窗口

- ✧ 若当前选中查看某个函数，如图 5-35，则点击【查看】->【总览】->【函数】，则会弹出一个新的针对此函数信息的总览窗口，如图 5-36：

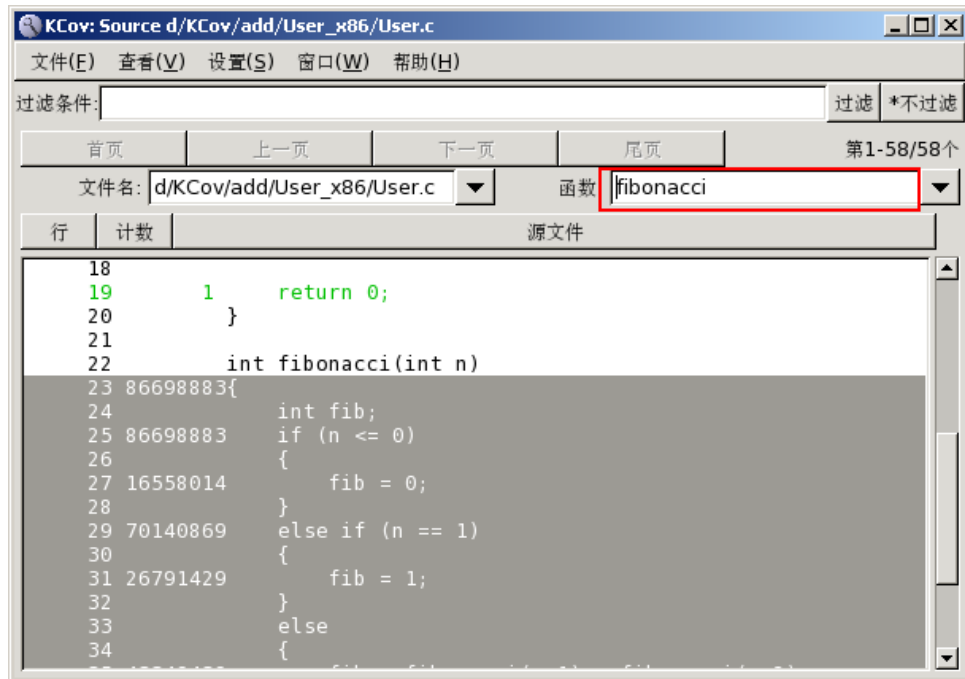


图 5-35 查看函数

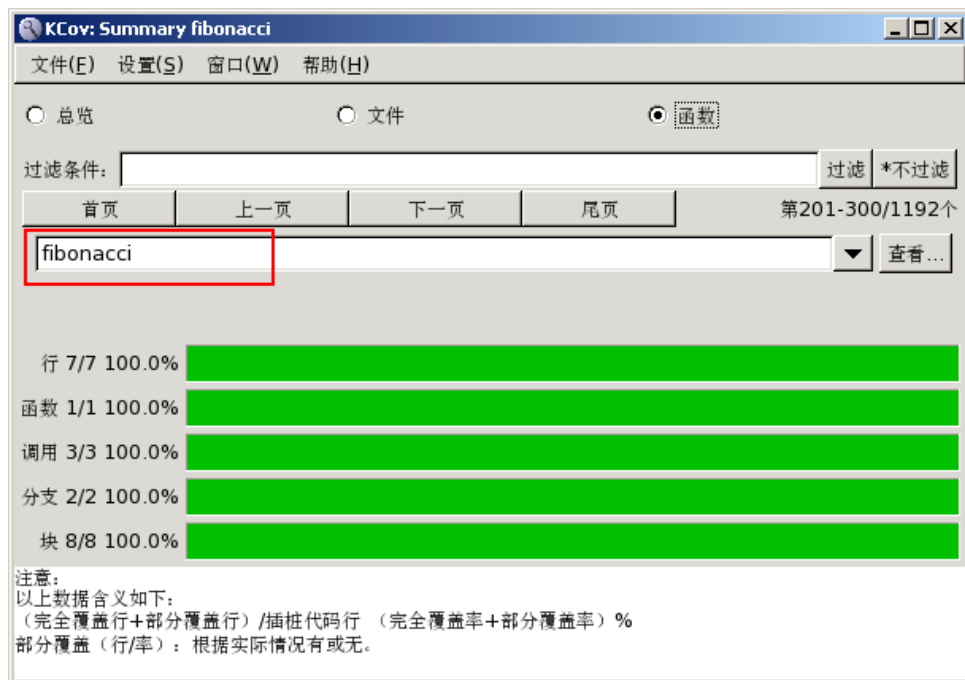


图 5-36 显示“函数”信息总览窗口

在文件查看视图，各列依次表示行号、块号、执行次数和该行的代码行。使用不同的颜色对代码的执行情况进行标注，且此颜色可自行设置，具体参考 5.1.6.6。代码执行次数分为三种情况：数字表示该

行代码的执行次数，####表示该行代码未执行，为空表示该行代码未打桩（即不纳入测试范围）。这样我们就可以直接分析每一行的执行情况。

### 5.1.6.3. 文件列表窗口

点击【窗口】->【新建文件列表窗口】，开启文件列表。在此窗口中列举了所有待 KCov 进行覆盖率分析的源文件信息，根据如图 5-37 设置的查看信息显示需求，当前的文件列表窗口具体覆盖率信息显示如图 5-38：

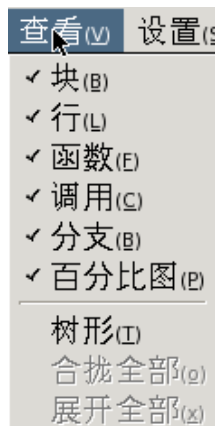


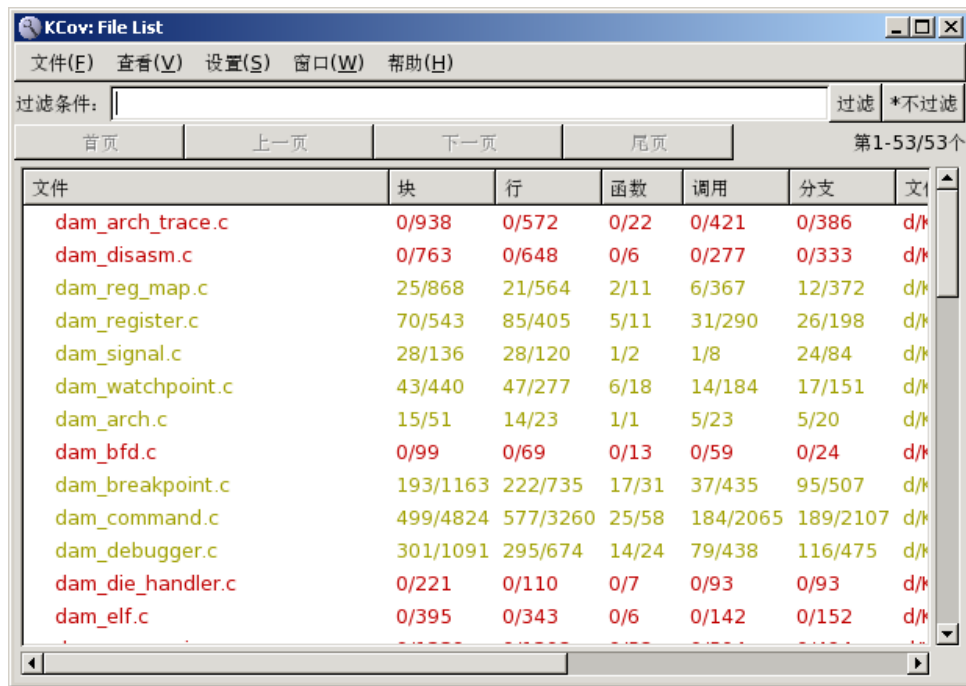
图 5-37 文件列表查看设置

文件	块	行	函数	调用	分支	文件路径
dam_arch_trace.c	0.00	0.00	0.00	0.00	0.00	d/KCov/zdbagent_
dam_disasm.c	0.00	0.00	0.00	0.00	0.00	d/KCov/zdbagent_
dam_reg_map.c	2.88	3.72	18.18	1.63	3.23	d/KCov/zdbagent_
dam_register.c	12.89	20.99	45.45	10.69	13.13	d/KCov/zdbagent_
dam_signal.c	20.59	23.33	50.00	12.50	28.57	d/KCov/zdbagent_
dam_watchpoint.c	9.77	16.97	33.33	7.61	11.26	d/KCov/zdbagent_
dam_arch.c	29.41	60.87	100.00	21.74	25.00	d/KCov/zdbagent_
dam_bfd.c	0.00	0.00	0.00	0.00	0.00	d/KCov/zdbagent_
dam_breakpoint.c	16.60	30.20	54.84	8.51	18.74	d/KCov/zdbagent_
dam_command.c	10.34	17.70	43.10	8.91	8.97	d/KCov/zdbagent_
dam_debugger.c	27.59	43.77	58.33	18.04	24.42	d/KCov/zdbagent_
dam_die_handler.c	0.00	0.00	0.00	0.00	0.00	d/KCov/zdbagent_
dam_elf.c	0.00	0.00	0.00	0.00	0.00	d/KCov/zdbagent_

图 5-38 文件列表显示 1

针对不同的覆盖率分析文件，依次显示了其块覆盖率、行覆盖率、函数覆盖率、函数调用关系覆盖率及分支覆盖率。

由于设置了【查看】->【百分比图】，都是以百分比的形式对各覆盖率信息进行显示，否则会具体显示出【覆盖信息/总信息】如下：



The screenshot shows the 'KCov: File List' window. It has a menu bar with '文件(F)', '查看(V)', '设置(S)', '窗口(W)', and '帮助(H)'. Below the menu is a search bar labeled '过滤条件:' with buttons '过滤' and '\*不过滤'. There are also navigation buttons: '首页', '上一页', '下一页', and '尾页'. A status bar at the top right indicates '第1-53/53个'. The main area is a table with the following columns: '文件', '块', '行', '函数', '调用', '分支', and '文'. The table lists 15 files with their respective coverage statistics.

文件	块	行	函数	调用	分支	文
dam_arch_trace.c	0/938	0/572	0/22	0/421	0/386	d/
dam_disasm.c	0/763	0/648	0/6	0/277	0/333	d/
dam_reg_map.c	25/868	21/564	2/11	6/367	12/372	d/
dam_register.c	70/543	85/405	5/11	31/290	26/198	d/
dam_signal.c	28/136	28/120	1/2	1/8	24/84	d/
dam_watchpoint.c	43/440	47/277	6/18	14/184	17/151	d/
dam_arch.c	15/51	14/23	1/1	5/23	5/20	d/
dam_bfd.c	0/99	0/69	0/13	0/59	0/24	d/
dam_breakpoint.c	193/1163	222/735	17/31	37/435	95/507	d/
dam_command.c	499/4824	577/3260	25/58	184/2065	189/2107	d/
dam_debugger.c	301/1091	295/674	14/24	79/438	116/475	d/
dam_die_handler.c	0/221	0/110	0/7	0/93	0/93	d/
dam_elf.c	0/395	0/343	0/6	0/142	0/152	d/

图 5-39 文件列表显示 2

对于显示出的被检测文件可以分别按照块、代码行、函数等覆盖率数值大小进行排序，只需点击相应的标题选项即可。同时，双击被检测文件，还可打开其源代码查看窗口。

若设置了【查看】->【树形】选项，则会显示分析文件所在的目录树形结构，如图 5-40：



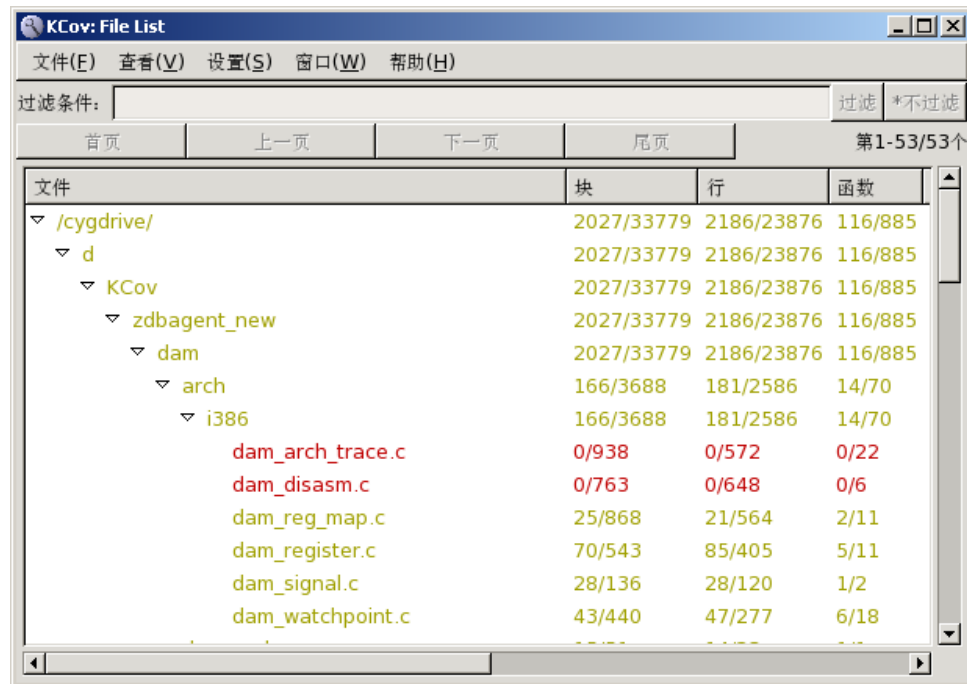


图 5-40 树形结构显示文件信息

#### 5.1.6.4. 函数列表窗口

点击【窗口】->【新建函数列表窗口】，开启函数列表。此窗口中显示当前分析文件中所有包含覆盖率数据的相关函数，及其具体的覆盖率信息。同样，若是采用如图 5-41 设置的信息显示，则当前的函数列表显示窗口如图 5-42:

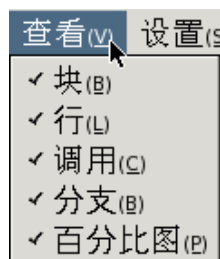
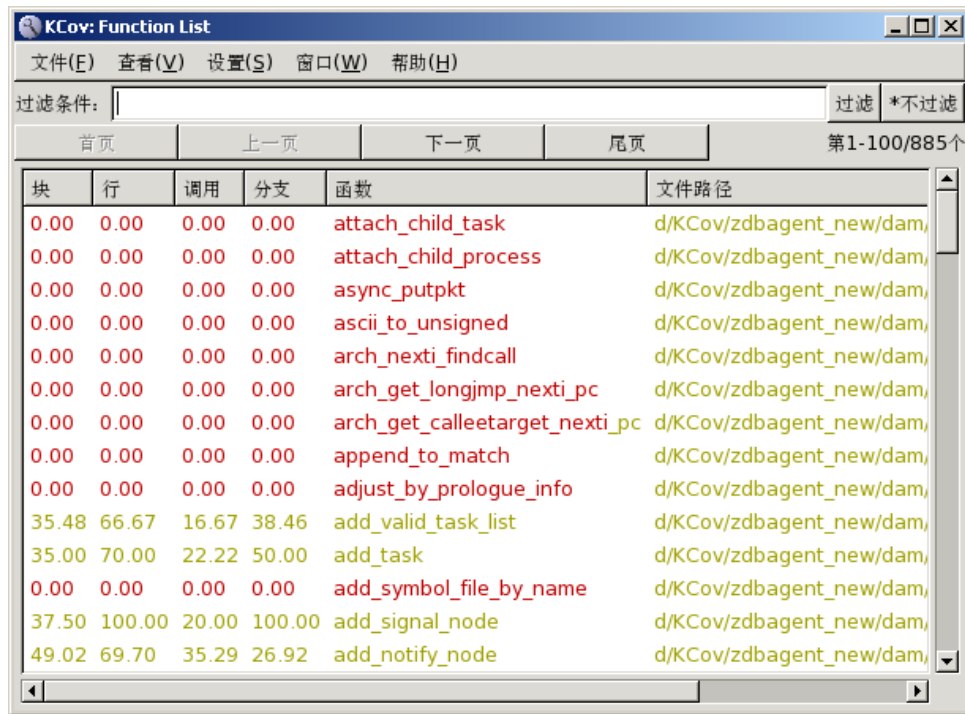


图 5-41 函数列表的 View 设置



块	行	调用	分支	函数	文件路径
0.00	0.00	0.00	0.00	attach_child_task	d/KCov/zdbagent_new/dam,
0.00	0.00	0.00	0.00	attach_child_process	d/KCov/zdbagent_new/dam,
0.00	0.00	0.00	0.00	async_putpkt	d/KCov/zdbagent_new/dam,
0.00	0.00	0.00	0.00	ascii_to_unsigned	d/KCov/zdbagent_new/dam,
0.00	0.00	0.00	0.00	arch_nexti_findcall	d/KCov/zdbagent_new/dam,
0.00	0.00	0.00	0.00	arch_get_longjmp_nexti_pc	d/KCov/zdbagent_new/dam,
0.00	0.00	0.00	0.00	arch_get_calleetarget_nexti_pc	d/KCov/zdbagent_new/dam,
0.00	0.00	0.00	0.00	append_to_match	d/KCov/zdbagent_new/dam,
0.00	0.00	0.00	0.00	adjust_by_prologue_info	d/KCov/zdbagent_new/dam,
35.48	66.67	16.67	38.46	add_valid_task_list	d/KCov/zdbagent_new/dam,
35.00	70.00	22.22	50.00	add_task	d/KCov/zdbagent_new/dam,
0.00	0.00	0.00	0.00	add_symbol_file_by_name	d/KCov/zdbagent_new/dam,
37.50	100.00	20.00	100.00	add_signal_node	d/KCov/zdbagent_new/dam,
49.02	69.70	35.29	26.92	add_notify_node	d/KCov/zdbagent_new/dam,

图 5-42 函数列表

函数列表中也能够针对具体函数，提供其块、代码行、函数调用关系、分支等一系列的详细覆盖信息。

对于各个函数也可以分别按照块、代码行、分支等覆盖率数值大小进行排序，只需点击相应的标题选项即可。同时，双击被检测函数，还可打开其源码查看窗口。

#### 5.1.6.5. 报告窗口

点击【窗口】->【新建报告窗口】，开启报告窗口，将呈现代码覆盖率测试的最终报告，如图 5-43，可以查看对所有文件的概要统计报告、每个目录的概要统计报告、每个文件中未测试的函数统计报告、每个文件中覆盖率低的函数统计报告、每个文件中未完全覆盖的函数统计报告等几大报告。

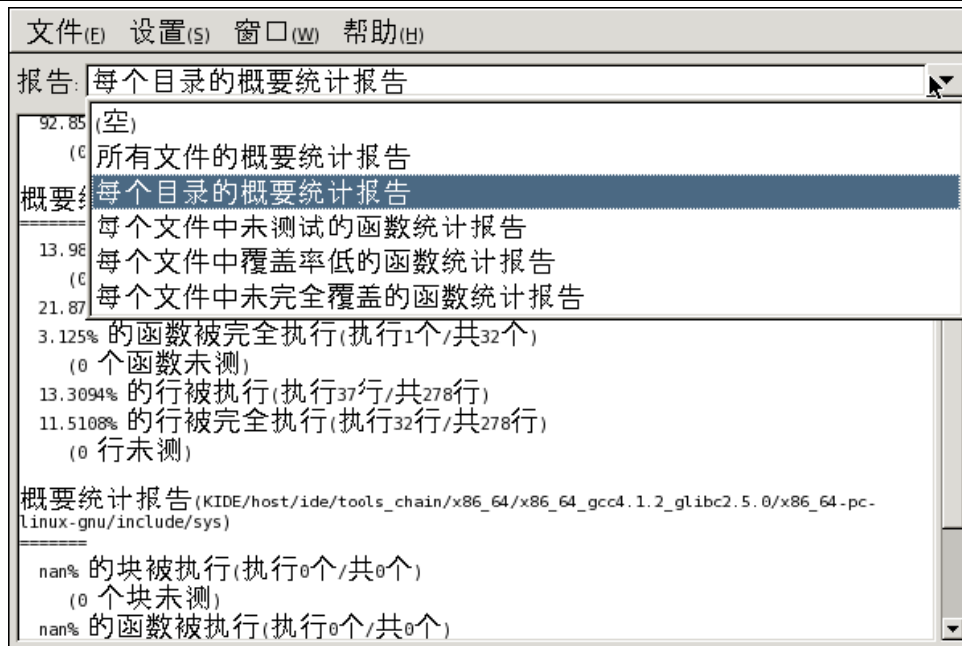


图 5-43 总体报告

➔ 提示：nan 是“not a number”的缩写，是 C/C++ 语言的标准输出，表示“无效数字”。

#### 5.1.6.6. 设置参数

点击【设置】->【首选项】，开启参数设置窗口：



图 5-44 开启参数设置窗口

#### ■ 窗口复用设置

若是对查询窗口进行了复用设置，则当再次选中相同类型查看窗口时，会替换掉当前的显示，保证显示窗口的唯一存在，且显示为最新需求的。目前，提供了对源码查看、总览窗口、文件列表、函数列表及总体报告等 5 大窗口的复用显示。点击【一般设置】，开启复用设置窗口。

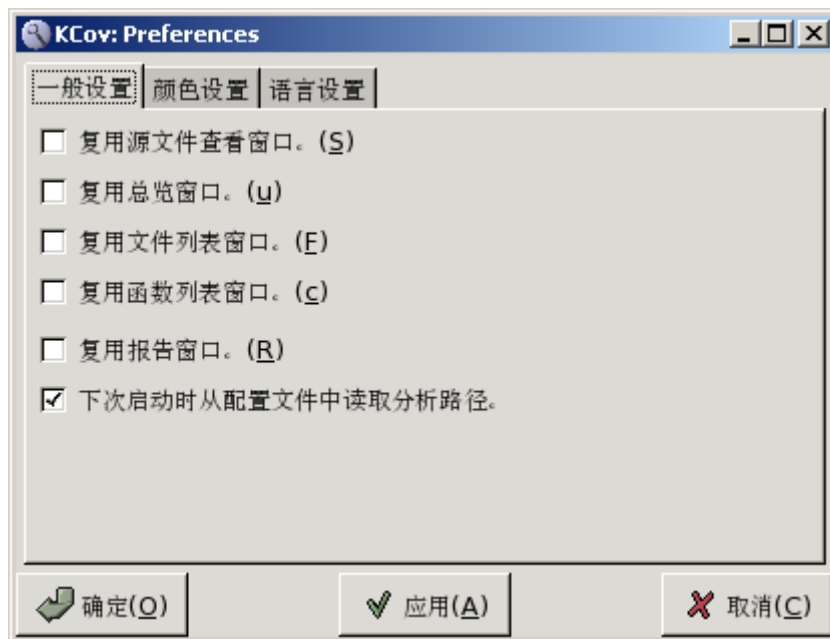


图 5-45 代码及总览窗口复用设置

#### ■ 自定义代码覆盖率颜色标记

可以对 KCov 分析覆盖率的数据颜色显示进行自定义，将用不同的颜色来标注代码的不同执行情况。点击【颜色设置】，开启颜色设置窗口。



图 5-46 自定义颜色

## ■ 自定义代码覆盖率语言

KCov 代码覆盖率工具有两种语言可供选择，即中文和英文，以便适应不同语言习惯的用户使用。点击【语言设置】，开启语言设置窗口。

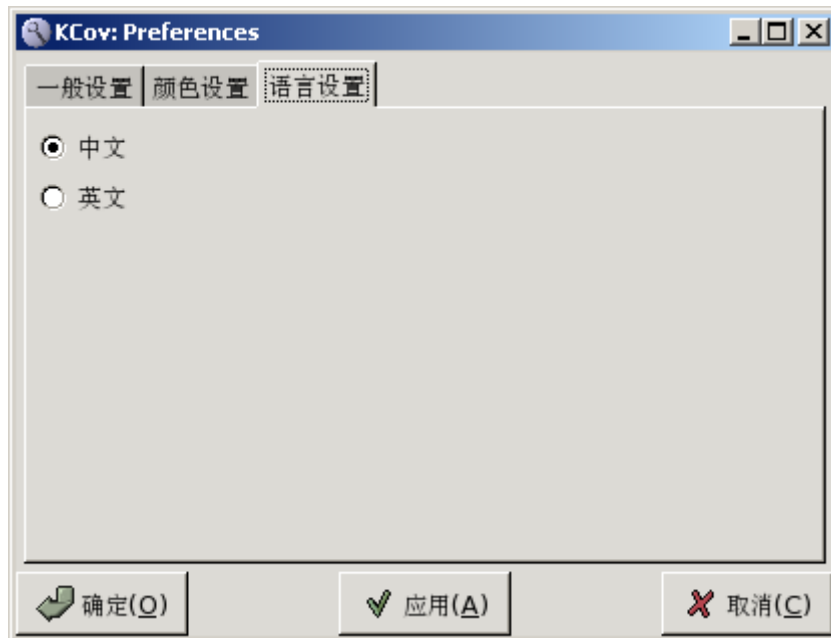



图 5-47 语言设置

 提示：修改了语言设置后，需重启 KCov 才能使得设置生效。

### 5.1.6.7. 过滤条件设置

目前，KCov 在总览窗口、文件列表窗口、函数列表窗口及源码查看窗口提供了以关键字搜索的过滤功能，用以满足用户查看代码覆盖率的不同需求。针对不同显示窗口提供了不同的过滤类型：

- 文件列表窗口：提供目录、文件两种过滤类型；
- 函数列表窗口：提供目录、文件、函数三种过滤类型；
- 源码查看窗口：提供目录、文件两种过滤类型。

其中，文件列表和函数列表在总览窗口中均有显示，因而 KCov 一并提供过滤功能。

KCov 提供了包含与排除两类过滤条件。无论采用何种过滤类型，都必须遵循如下过滤规则：

1. 匹配模式：“\*”匹配任意字符串（不包括/），“\*\*”匹配任意字符串（包括/），“？”匹配任意一个字符（不包括/）；

2. 函数过滤类型以字符^开头，目录过滤类型以字符\结尾，文件过滤类型无特殊要求；
3. 若要使用“排除”条件，则需在过滤条件前添加“-”符号，否则使用包含条件，“-”优先级高于“^”，所以当对函数类型进行排除过滤时，应在字符串前依次添加“-^”；
4. 多个条件过滤，以“;”进行分隔；
5. 文件类型、函数类型、目录类型可以进行组合过滤。

过滤的对象是针对界面显示的覆盖率分析结果，即文件列表集合与函数列表集合来进行过滤操作。

► **注意事项：**

1. 当“\*\*”与“？”连用，则 KCov 直接识别为“\*\*”，“\*”和“？”连用，KCov 直接识别为“\*”进行处理；
2. 多个条件的组合过滤，过滤结果是分别满足单个过滤条件得到的集合之和；
3. 文件列表窗口提供了两种过滤类型：当指定为**文件类型**过滤时，若过滤条件中包含“\*\*”或“/”，则 KCov 将从**全路径**中查找匹配字符串，获得符合条件的文件列表或函数列表，否则仅从**文件名**中查找匹配字符串，获得符合条件的文件列表，或文件包含的函数列表；当指定为**目录类型**过滤时，若过滤条件中包含“\*\*”或“/”，KCov 将从**全路径**中查找匹配字符串，获得符合条件的目录包含的文件列表或函数列表，否则针对每一个**目录名**本身进行匹配，获得符合条件的目录包含的文件列表或函数列表；
4. 函数列表中，当指定为**函数类型**过滤时，若过滤条件中包含“\*\*”或“/”，KCov 将从**全路径**中查找匹配字符串，获得符合条件的目录文件包含的函数列表。否则仅针对**函数名**本身进行匹配，获得符合条件的函数列表。

■ **例一：从文件信息列表中过滤出包含字符串“arch”的文件**

总览窗口的文件列表信息，在所有包含覆盖率数据信息的文件路径中将包含“arch”字符串的文件过滤出来，如图 5-48，采用“\*\*arch\*\*”过滤条件进行过滤，由于采用的是“\*\*arch\*\*”，表示 arch 字符串前后为任意长度字符（包含/字符），所以是对整条文件路径进行过滤，因而不仅仅包括文件名中包含“arch”字符串的文件，也包括路径信息中包含 arch 的文件：

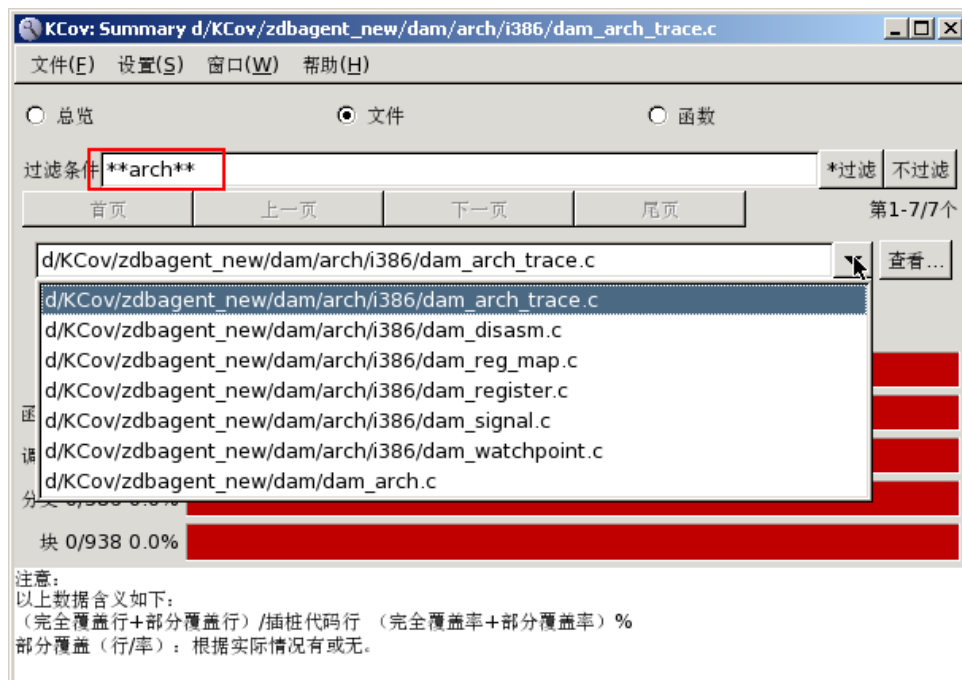


图 5-48 文件过滤

#### ■ 例二：对模糊记忆字符串的查找过滤

在总览窗口的文件列表信息，若是希望查找路径中包含“disasm”字符串的文件，但却只模糊记得其中几个字符，则可使用过滤条件“\*d??asm\*”，“?”代表一位字符，但是不代表“/”，如图 5-49：

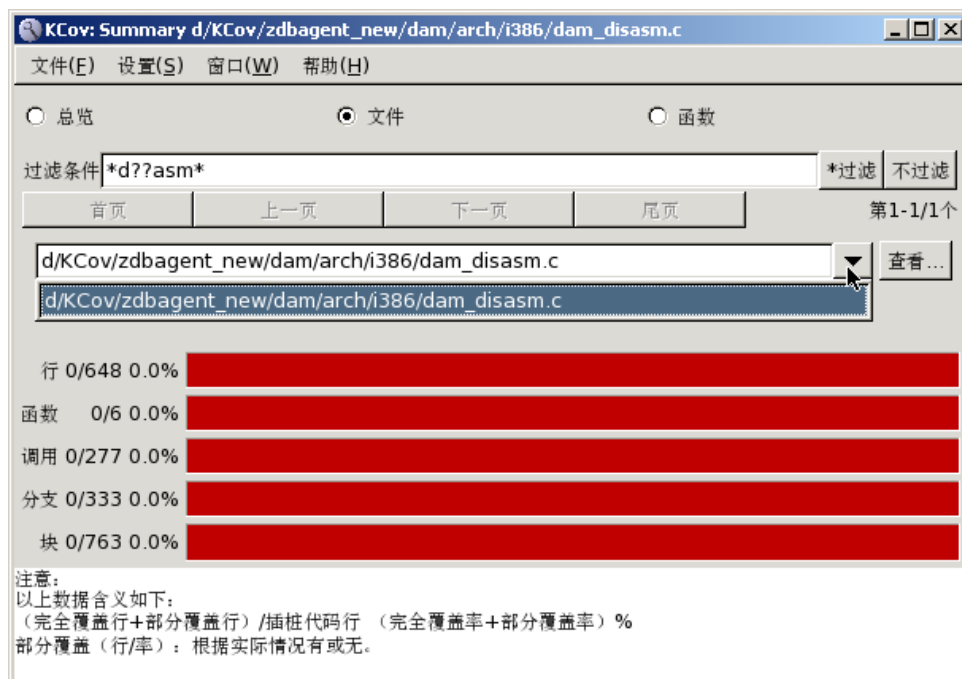


图 5-49 模糊查找

### ■ 例三：从函数信息列表中将“SPACES”字符串结尾的函数排除

总览窗口的函数列表信息，在所有包含覆盖率数据信息的函数中，将所有以“SPACES”字符串结尾的函数排除，显示过滤后的函数列表信息，如图 5-50，采用的是“-^\*SPACES”过滤条件：

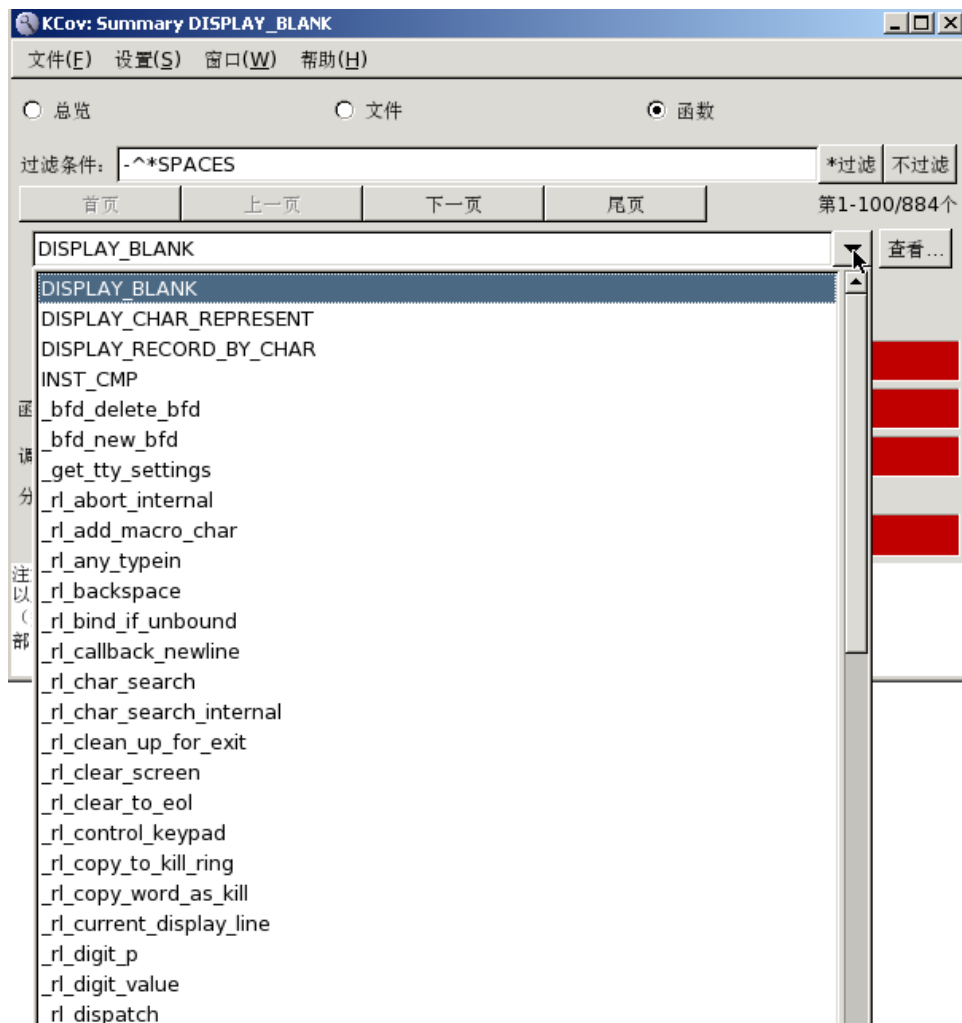


图 5-50 函数过滤

### ■ 例四：从文件信息列表中过滤出包含“i386”字符串的目录

总览窗口的文件列表信息中，在所有包含覆盖率数据信息的文件路径中，将包含“i386”字符串的目录过滤出来，如图 5-51，采用的是“\*\*i386\”过滤条件，将文件路径信息中包含“i386”的目录过滤了出来。



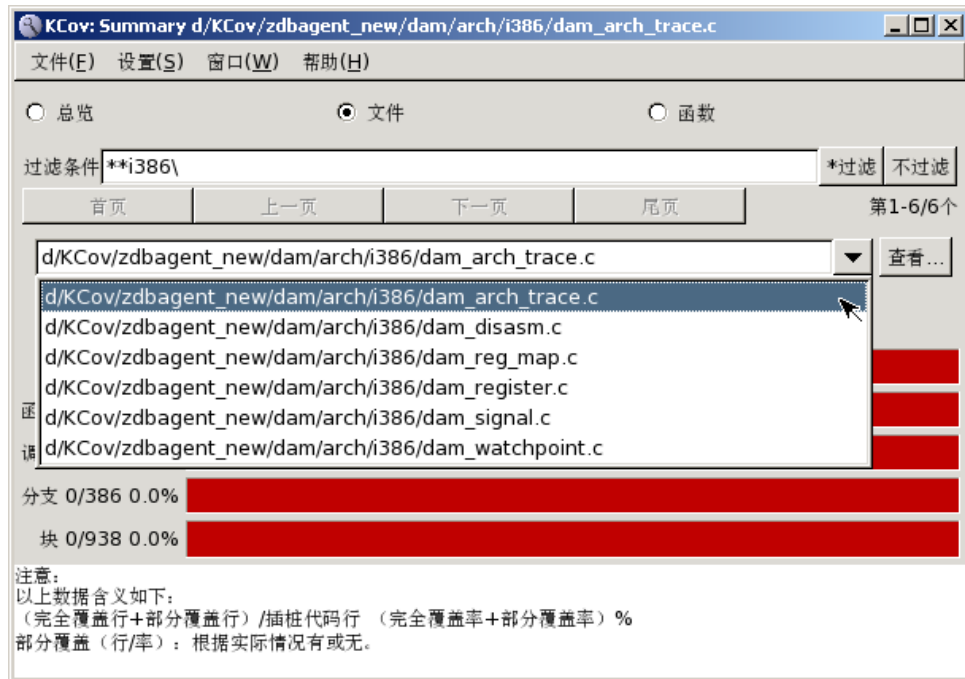


图 5-51 目录过滤

■ 例五：将包含字符串“i386”的目录和包含字符串“handler”的文件均过滤出来

总览窗口的文件列表信息中，在所有包含覆盖率数据信息的文件路径中，将包含“i386”字符串的目录过滤出来；同时，将包含“handler”字符串的文件一并过滤出来。如图 5-52，采用的是“\*\*i386\;\*\*handler\*\*”过滤条件，组合过滤，过滤条件以“;”分隔。

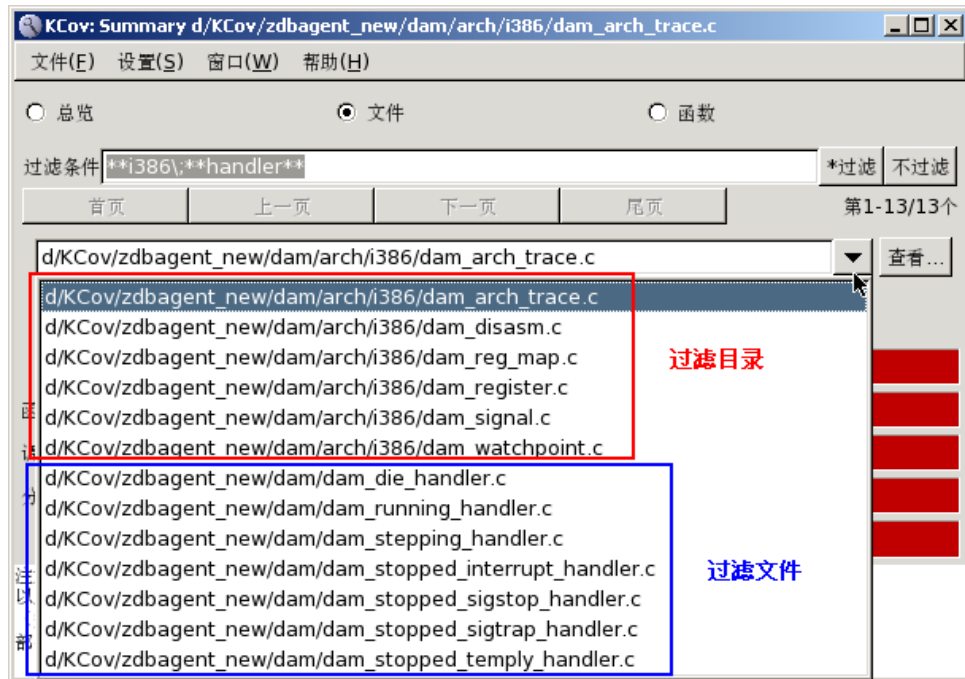


图 5-52 组合过滤

■ 例六：在包含“i386”字符串的目录下过滤出包含“arch”字符串的文件

总览窗口的文件列表信息中，在所有包含覆盖率数据信息的文件路径中，将包含“i386”字符串的目录下含有“arch”字符串的文件过滤出来。采用过滤条件“\*\*i386/\*arch\*.c”，如图 5-53：

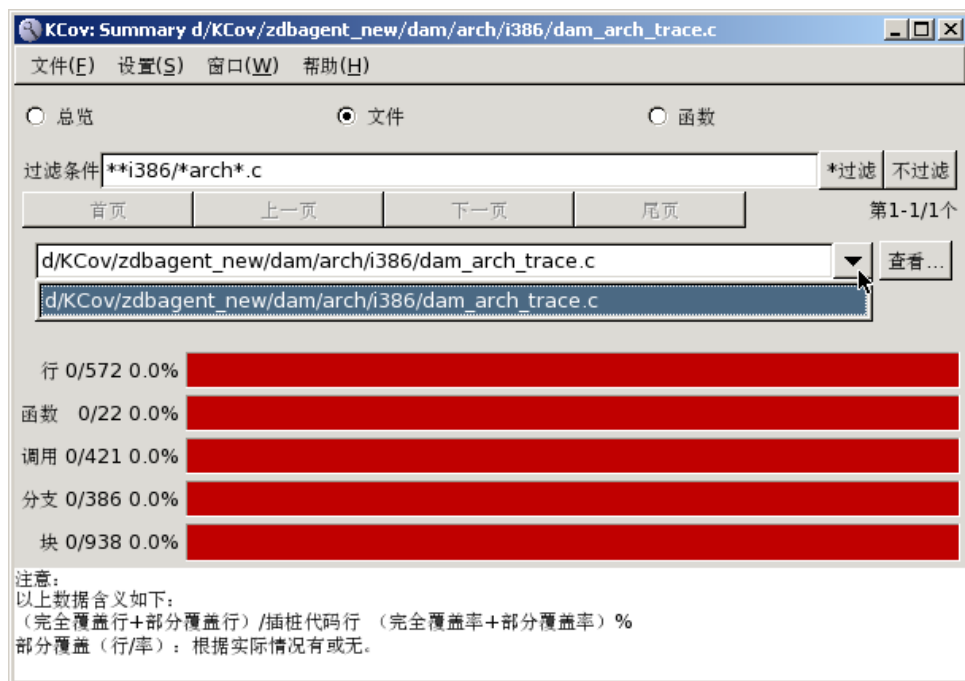


图 5-53 目录与文件的交集过滤

### 5.1.7. 保存覆盖率数据

对于当前 KCov 生成的覆盖率数据的代码解析结果，我们可以保存为 html 文件供事后分析使用。只需在分析窗口，点击【文件】->【保存项目】，即可开启保存覆盖率数据的设置窗口，如图 5-54：

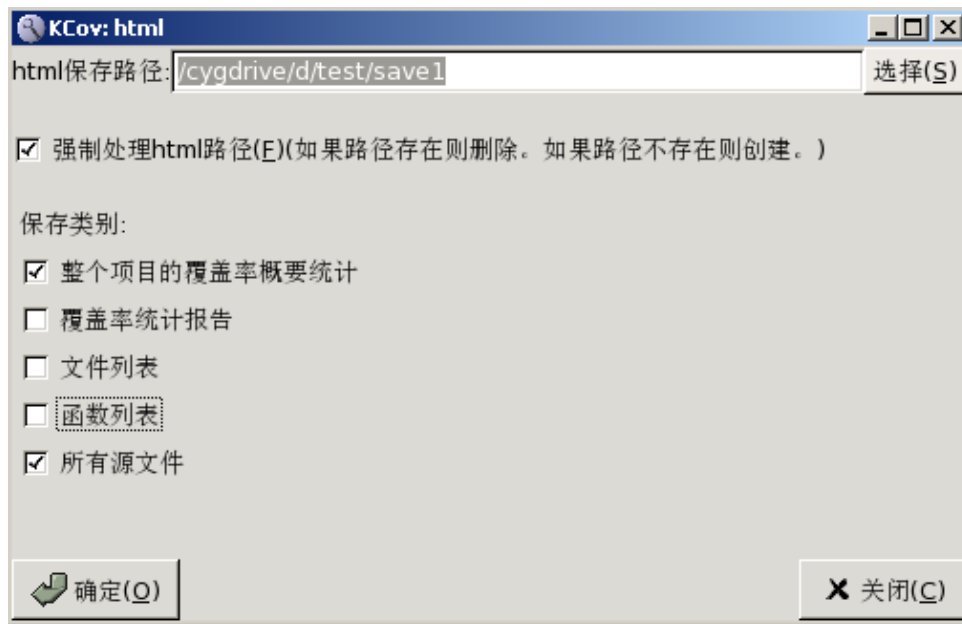


图 5-54 保存覆盖率数据

我们需要设置保存为 html 文件的路径，其默认保存在指定的路径下的 kcov-project-html 目录。


强制处理 html 路径：勾选此选项后，若设置的 html 保存路径不为空，则先清空再保存 html 格式的覆盖率数据，若设置的 html 保存路径不存在，则先进行创建。

这里提供了五种保存类别：

- 整个项目的覆盖率概要统计：覆盖率数据的概要统计信息，包含目录层次结构的覆盖率信息。如果同时选择了保存“所有源文件”，则可以直接点击源文件名查看源文件的覆盖率信息；
- 覆盖率统计报告：显示统计报告列表。包括所有文件的概要统计报告、每个目录的概要统计报告、每个文件中未测试的函数统计报告、每个文件中覆盖率低的函数统计报告、每个文件中未完全覆盖的函数统计报告等 5 大报告；
- 文件列表：使用列表方式保存所有文件的覆盖率统计信息。如果同时选择了保存“所有源文件”，则可以直接点击源文件名查看源文件的覆盖率信息；
- 函数列表：使用列表方式保存所有文件相关联的包含覆盖率数据的函数的覆盖率数据统计信息。

如果同时选择了保存“所有源文件”，则可以通过点击函数名查看该函数对应的源文件的覆盖率信息；

- 所有源文件：保存所有源文件代码及相关的覆盖率信息。选择该类别时，至少同时也选择了“整个项目的覆盖率概要统计”、“文件列表”、“函数列表”中的一项或多项。

 提示：保存的覆盖率数据仅针对当前 KCov 设置的分析路径。

设置完成后，点击**确定**，即将保存了覆盖率数据信息的 html 文件生成到指定文件夹下的 kcov-project-html 目录下，例如：我们设置保存路径为“/cygdrive/d/test/save1/”，勾选“整个项目的覆盖率概要统计”和“所有源文件”，则保存在如下目录中，如图 5-55，html 保存内容如图 5-56。

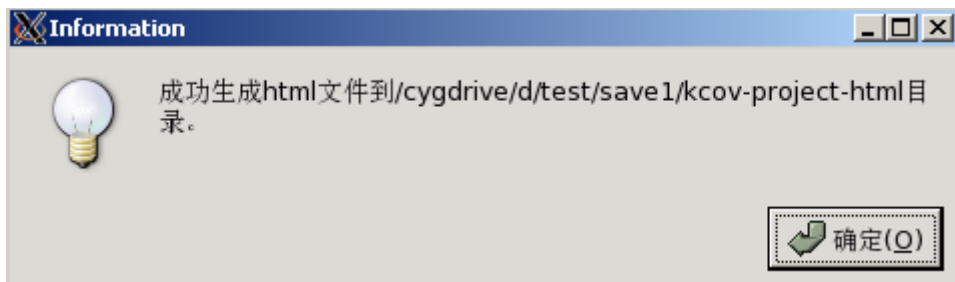


图 5-55 成功生成 html 文件



图 5-56 保存“整个项目的覆盖率概要统计”首页页面

同时，在 html 文件中保存了覆盖率数据的目录结构层次，即是在 KCov 代码覆盖率分析结果视图的下方显示的**路径**信息，通过点击进入，可以查看到与目录结构层次一样的结构，如图 5-57，同时还可以对当前目录下的目录和文件的覆盖率信息进行查看，若同时对所有源文件信息进行了保存，则可以打开相应的源文件 html 进行查看，同时会显示出该源码所有函数的函数级覆盖率数据，如。

路径	块	行	函数	调用	分支
test/cgel3.0/linux/arch	35.40	440/1243	44.48	516/1160	63.16
test/cgel3.0/linux/block	3.21	198/6170	6.01	283/4705	14.19
test/cgel3.0/linux/drivers	16.85	2847/16896	20.33	2538/12484	25.68
test/cgel3.0/linux/grsecurity	56.48	61/108	57.22	103/180	56.48
test/cgel3.0/linux/include					
test/cgel3.0/linux/init	29.65	698/2354	46.48	699/1504	66.67
test/cgel3.0/linux/ipc	3.67	124/3383	6.91	186/2690	22.29
test/cgel3.0/linux/kernel	24.62	6172/25066	30.17	4823/15986	39.31
test/cgel3.0/linux/lib	34.84	1191/3418	40.15	1178/2934	49.06
test/cgel3.0/linux/mm	30.50	5811/19053	34.75	3881/11168	46.41
test/cgel3.0/linux/net	22.78	13178/57841	27.84	10393/37333	39.66
test/cgel3.0/linux/security	32.59	44/135	36.00	45/125	38.10

图 5-57 显示目录结构层次

文件的覆盖率信息:					
行	52+61/209	24.9+29.2%			
函数	0+17/21	0.0+81.0%			
调用	43/117	36.8%			
分支	18/65	27.7%			
块	95/238	39.9%			

函数的覆盖率信息:					
函数	块	行	调用	分支	
__dma_sync_page	0.00	0/3	0.00	0/4	0.00
__dma_sync	0.00	0/9	0.00	0/11	0.00
dma_alloc_init	65.52	19/29	62.50	10/16	71.43
__dma_free_coherent	0.00	0/45	0.00	0/33	0.00
__dma_alloc_coherent	65.00	26/40	71.43	30/42	68.42
vm_region_find	0.00	0/12	22.22	2/9	0.00
vm_region_alloc	50.77	33/65	71.70	38/53	50.00
flush_tlb_kernel_range	0.00	0/2	0.00	0/3	0.00
lowmem_page_address	100.00	1/1	100.00	2/2	100.00
pmd_offset	100.00	1/1	100.00	2/2	100.00
ptep_get_and_clear	0.00	0/4	75.00	3/4	0.00
set_pte_at	100.00	1/1	100.00	3/3	100.00
pgd_none	100.00	1/1	100.00	1/1	100.00
alloc_pages_node	76.92	10/13	88.89	8/9	100.00
list_del	0.00	0/2	80.00	4/5	0.00
local_irq_save_ptr	0.00	0/2	50.00	1/2	0.00
test_ti_thread_flag	0.00	0/2	100.00	1/1	0.00
current_thread_info	0.00	0/1	100.00	1/1	0.00
get_order	100.00	1/1	100.00	4/4	100.00
clear_bit	0.00	0/2	50.00	1/2	0.00
set_bit	100.00	2/2	100.00	2/2	100.00

图 5-58 显示源码所有函数的函数覆盖率信息

## ■ 保存单个源文件覆盖率数据

除了可以对整个项目的覆盖率数据信息进行保存之外，KCov 还提供了对单个源文件覆盖率数据的保存功能，只需在源文件窗口，点击【文件】->【另存为 html】或者【文件】->【另存为文本（txt）】，如图 5-59，即可将源文件的覆盖率数据保存为 html 文件或者是文本文件。

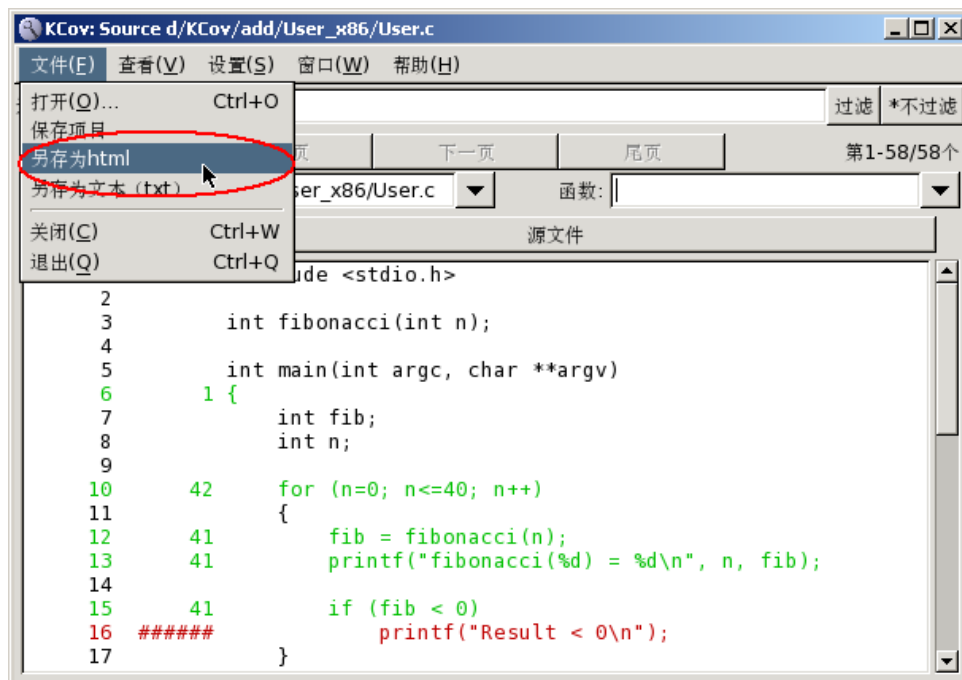


图 5-59 保存单个源文件

在弹出的保存覆盖率设置窗口，只需设置其保存路径，如图 5-60，则会生成与源文件同名的 html 文件或是 txt 文件。如保存 User.c 的覆盖率数据为 html，则会生成 User.c.html 文件。

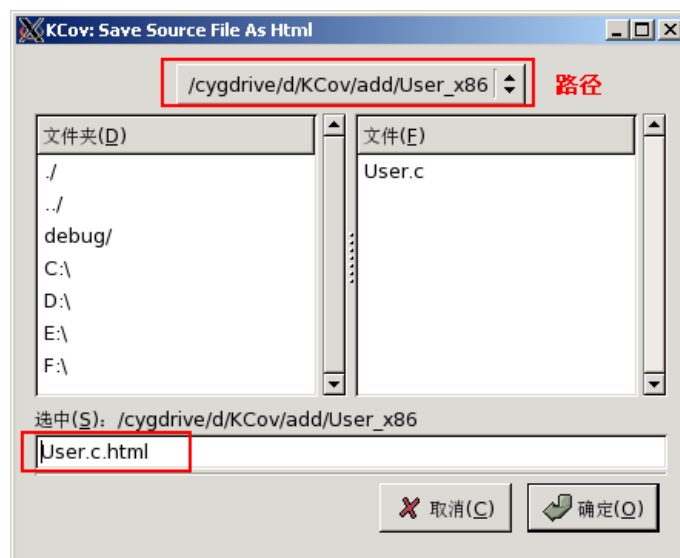


图 5-60 设置保存路径

## 5.2. 使用命令行 KCov

KCov 不仅仅提供了图形界面,而且还提供了命令行显示方式便于不同需求的用户使用。命令行 KCov 数据分析工具提供了对覆盖率数据的分析报告功能。能够对以下三种情况:

- 单个目录（包括子目录）
- 单个文件
- 多个目录（包括子目录）

进行函数/逻辑块/代码行等级别的覆盖率数据统计分析,在命令行中显示覆盖率分析结果。

不同于图形界面,将 KCov 的 input 窗口启动后,再进行待分析的数据文件设置,命令行方式在启动 KCov 的同时,需指定日志文件路径和数据文件路径。KCov 通过提供用户态程序 tkcov.exe 来完成命令行 KCov 的代码覆盖率分析,tkcov.exe 位于安装目录/host/ide/kcov/bin 目录下,在 cywin 窗口下启动命令即可。

启动代码覆盖率分析命令

```
./tkcov.exe <解析记录参数> <导出结果参数> [其他参数]
```

### ■ 解析记录参数

关于解析记录参数有三种设置方式:

1. 通过直接输入参数设置:

```
<-d <data-path[,data-path]>> <-n <note-path>> <-u | -s <source-path>> [-U] [-r] [-l  
<library-path[,library-path]>] [-e <elf-path[,elf-path]>] [-f] [-p filter]
```

表 5-1 解析记录参数 I

参数定义	功能与含义
-d <data-path[,data-path]	必选项。指定数据文件路径,可以是目录或文件。data-path 为指定的数据文件路径,支持多条路径,以','分隔
-n <note-path	必选项。指定日志文件路径,可以是目录或文件

-u   -s <source-path>	-u 无源码, -s 配置源码路径, 两个参数必设置其中一个
-U	去激活参数, 配置不解析指定的解析记录
-r	指定递归参数, 针对多级目录生效, 单个文件无效
-l <library-path[,library-path]>	配置多个动态库所在的路径, library-path 为指定的动态库路径, 支持多个库路径, 以','分隔
-e <elf-path[,elf-path]>	可选项。指定 elf 文件路径。elf-path 为指定的 elf 文件路径, 支持多个文件路径以','分隔, 用于缩小覆盖率分析文件的搜索范围, 适用于处理同时有多个程序都生成了日志、数据文件, 而用户仅对某些库、某些程序的覆盖率关心的情况
-f	是否激活解析的过滤条件
-p filter	配置解析的过滤条件

## 2. 通过-I 参数设置

表 5-2 解析记录参数 II

参数定义	功能与含义
-I <"record">	配置多条解析记录, 可通过同时指定多个-I 参数的方式来实现指定多条解析记录, 不指定-I 参数, 则默认只有一条记录, record 接收的参数形式就是直接设置参数方式: [<-d <data-path[,data-path]>> <-n <note-path>> <-u   -s <source-path>> [-U] [-r] [-l <library-path[,library-path]>] [-e <elf-path[,elf-path]>] [-f] [-p filter]]这些参数

## 3. 通过-c 读取配置文件中的覆盖率数据方式设置:

表 5-3 解析记录参数 III

参数定义	功能与含义
------	-------



-c	从配置文件中读取解析记录，指定-c 后会忽略其他解析记录参数-l、-d、-n、-s、-u、-U、-r、-e、-I、-f、-p，配置文件是指 KCov 工具的配置文件，由 KCov 工具生成、位置为：KIDE 安装目录/host/ide/kcov/kcovcfg.xml
----	--

## ■ 导出结果参数

对于导出覆盖率数据分为直接打印报告信息与保存覆盖率数据为 html 文件两种。

### 1. 直接打印报告信息

**-R <type>**

表 5-4 导出结果参数 I

参数定义	功能与含义
-R <type>	<p>-R 指定获取覆盖率数据分析报告</p> <ul style="list-style-type: none"> <li>➤ -R list: 可输出的报告类别</li> <li>➤ -R summary_all: 输出所有文件的概要统计报告</li> <li>➤ -R summary_per_directory: 输出每个目录的概要统计报告</li> <li>➤ -R untested_functions_per_file: 输出每个文件中未测试的函数统计报告</li> <li>➤ -R poorly_covered_funcs_per_file: 输出每个文件中覆盖率低的函数统计报告</li> <li>➤ -R incompletely_covered_functions_per_file: 输出每个文件中未完全覆盖的函数统计报告</li> <li>➤ -R all: 输出以上所有类别的报告</li> </ul>

### 2. 保存覆盖率数据为 html 文件

保存为 html 文件又分为两种方式：

从配置文件中读取“保存项目信息”来保存：

**-m**

直接设置参数保存：

**[-S <path> -h -t <type[,type]**

表 5-5 导出结果参数 II

参数定义	功能与含义
-m	从配置文件中获取“保存项目”信息，即 html 格式的覆盖率数据保存到配置文件中的“html 保存路径”，“保存类别”，是否“保存路径强制处理”来处理保存目录，指定-m 后，会忽略其他导出结果参数-R、-S、-h、-t
-S <path>	-S 指定 html 格式的覆盖率数据的保存路径
-h	配置“保存路径强制处理”功能，当-S 指定的 path 不存在，则创建 path；若存在不为空，则先清空再保存 html 格式的覆盖率数据，-h 在指定了-S 参数才有效
-t <type[,type]	指定保存类别，在指定了-S 参数才有效 <ul style="list-style-type: none"> <li>➤ -t summary: 保存总览级别类型数据；</li> <li>➤ -t report: 保存报告级别类型数据；</li> <li>➤ -t file: 保存文件级别类型数据；</li> <li>➤ -t function: 保存函数级别类型数据；</li> <li>➤ -t source: 保存源代码级别类型数据；</li> <li>➤ -t all: 保存以上所有类型数据；</li> </ul>

## 其他参数

表 5-6 其他参数

参数定义	功能与含义
-v	打印 KCov 版本信息
help	获取命令的帮助信息

KCov 命令行完整命令参考如下：

```
./tkcov.exe <[-d <data-path[,data-path]>> <-n <note-path>> <-u | -s <source-path>> [-U] [-r] [-l <library-path[,library-path]>] [-e <elf-path[,elf-path]>] [-f] [-p filter]] | [-l "record"] | [-c] ] <-R <type>
```

```
| <[-S <path> -h -t <type[,type]>] | [-m]>> [-v] [--help]
```

➡ 提示：若只针对单个文件进行覆盖率分析和输出统计报告，则可将日志文件路径或数据文件路径中的一个或两个指定为文件路径。否则，则是针对目录进行覆盖率分析和输出目录的统计报告，对目录的分析注意添加递归参数-r。

例如，使用命令行 KCov 对目录进行覆盖率分析，生成所有覆盖率数据的分析报告，如图 5-61：

```
周雨@ZTE-20110705PWL /cygdrive/f/Download/KCov/kcov/bin
$ ./tkcov.exe -d /cygdrive/d/test/add/1/,/cygdrive/d/test/add/2 -n /cygdrive/d/
test/add/User_x86/debug/ -R all -u

Notice:
-----
Input detail parameter method
1. Using the parameter 'd' to set data path as format "-d <path[,path]>". This p
arameter is required!
2. Using the parameter 'n' to set note path as format "-n <path>". This paramete
r is required!
3. Using the parameter 's' to set source path as format "-s <path>". Using the p
arameter 'u' to set no source. These two parameters required one!
4. Using the parameter 'S' to set save path of html result as format "-S <path>"
. Using the parameter 'R' to set report type, and format as "-R <type>". These
two parameters require one!
5. Using the parameter 'I' to set other analysis record as format "-I <"<-d path
> <-n path> <-u|-s path> [...]"]>". This parameter is optional and can be specifi
ed multiple times!
-----
Read from configuration file method
1. Using the parameter 'c' to read analysis record from configuration file. This
parameter is valid when the configuration file contain records info!
2. Using the parameter 'm' to read output html setting from configuration file.
This parameter is valid when the configuration file contain html info
-----
Other
For more information, please check KCov's user manual.
-----

valid records sum is 1:
valid record(No. 1) is active:
data path is : /cygdrive/d/test/add/1/,/cygdrive/d/test/add/2
note path is : /cygdrive/d/test/add/User_x86/debug/
source path is : none

=====Report=====
Summary (all files)
=====
 93.75% blocks executed (15 of 16)
    (0 blocks suppressed)
100% functions executed (2 of 2)
50% functions completely executed (1 of 2)
    (0 functions suppressed)
92.8571% lines executed (13 of 14)
92.8571% lines completely executed (13 of 14)
    (0 lines suppressed)
```

图 5-61 命令行 KCov 覆盖率分析报告

同时，我们也可以将生成的覆盖率数据的代码解析结果，保存为 html 文件供事后分析使用，如图 5-62，使用了参数-S 指定 html 格式的覆盖率数据的保存路径为 D:\test\add\path，-h 配置“保存路径强制处理”功能，-t 指定保存类别，此处指定为 report，保存报告级别类型数据。

```
周雨@ZTE-20110705PWL /cygdrive/f/Download/KCov/kcov/bin
$ ./tkcov.exe -d /cygdrive/d/test/add/1/ -n /cygdrive/d/test/add/User_x86/debug
/ -u -S /cygdrive/d/test/add/path -h -t report

Notice:
-----
Input detail parameter method
1. Using the parameter 'd' to set data path as format "-d <path[,path]>". This p
arameter is required!
2. Using the parameter 'n' to set note path as format "-n <path>". This paramete
r is required!
3. Using the parameter 's' to set source path as format "-s <path>". Using the p
arameter 'u' to set no source. These two parameters required one!
4. Using the parameter 'S' to set save path of html result as format "-S <path>"
. Using the parameter 'R' to set report type, and format as " -R <type>". These
two parameters require one!
5. Using the parameter 'I' to set other analysis record as format "-I <"<-d path
> <-n path> <-u|-s path> [...]>". This parameter is optional and can be specifi
ed multiple times!
-----
Read from configuration file method
1. Using the parameter 'c' to read analysis record from configuration file. This
parameter is valid when the configuration file contain records info!
2. Using the parameter 'm' to read output html setting from configuration file.
This parameter is valid when the configuration file contain html info
-----
Other
For more information, please check KCov's user manual.
-----

valid records sum is 1:
valid record(No. 1) is active:
data path is : /cygdrive/d/test/add/1/
note path is : /cygdrive/d/test/add/User_x86/debug/
source path is : none

Save coverage data as html format information:
save directory is : /cygdrive/d/test/add/path
save directory force handle is : true
save type is : report

Save coverage data as html format success!
```

图 5-62 命令行保存覆盖率数据为 html

设置完成后，即将保存了覆盖率数据信息的 html 文件生成到指定路径下的 kcov-project-html 目录下，如此处 D:\test\add\path\kcov-project-html\下生成的统计报告 report\report.html，如图 5-63。



图 5-63 保存统计报告信息

## 6. FAQ

### 6.1. 点击 KCov 分析工具图标无响应

- 关键字： 点击 KCov 图标      启动失败
- 版本： all
- 问题描述：

通过点击 KCov 分析图标  启动 KCov，但无任何响应，这是为什么？

- 分析及处理：

经分析，出现此状况是由于 Xwin 未能正常启动所导致。

XWin 缺省使用 6000 端口，而查看用户机器，发现 6000 端口已被占用。

```
C:\Documents and Settings\zte>netstat -a -n  
  
Active Connections
```

Proto	Local Address	Foreign Address	State
TCP	0.0.0.0:135	0.0.0.0:0	LISTENING
TCP	0.0.0.0:445	0.0.0.0:0	LISTENING
TCP	0.0.0.0:1125	0.0.0.0:0	LISTENING
TCP	0.0.0.0:3389	0.0.0.0:0	LISTENING
TCP	0.0.0.0:3511	0.0.0.0:0	LISTENING
TCP	0.0.0.0:3521	0.0.0.0:0	LISTENING
TCP	0.0.0.0:3526	0.0.0.0:0	LISTENING
TCP	0.0.0.0:6000	0.0.0.0:0	LISTENING
TCP	0.0.0.0:8009	0.0.0.0:0	LISTENING
TCP	0.0.0.0:8080	0.0.0.0:0	LISTENING

此时，需要为 XWin 重新指定空闲端口，方可解决此问题，具体操作如下：

修改 KCov 的启动脚本 startkcov.sh，有两处需要修改（比如 6001 端口空闲）：

➤ 第一处：

```
export DISPLAY=127.0.0.1:0.0
```

改为

```
export DISPLAY=127.0.0.1:1.0
```

➤ 第二处：

```
XWin -multiwindow -clipboard -silent-dup-error -once &
```

改为

```
XWin :1 -multiwindow -clipboard -silent-dup-error -once &
```

备注：XWin 指定端口以 6000 为基准，上述修改红色部分与端口号一一对应，即 6001，则对应修改为 1；6002，则修改为 2，以此类推。

## 7. 开源 LICENSE 说明

### 7.1. GPL 简介

GPL 是 GNU 通用公共许可证的简称，为大多数的 GNU 程序和超过半数的自由软件所采用，Linux

内核就是基于 GPL 协议而开源。GPL 许可协议对在 GPL 条款下发布的程序以及基于程序的衍生著作的复制与发布行为提出了保留著作权标识及无担保声明、附上完整、相对应的机器可判读源码等较为严格的要求。GPL 规定，如果将程序做为独立的、个别的著作加以发布，可以不要求提供源码。但如果作为基于源程序所生著作的一部分而发布，就要求提供源码。

成都研究所 OS 平台基于开源社区研发提供适用于各产品线的嵌入式 Linux 产品及相关工具，从总体而言，主要应该遵循 GPL 许可协议的条款。Linux 是基于 GPL2.0 发布的开源软件，CGEL 是基于 Linux 内核进行修改完成的新作品，因此，根据 GPL 协议第 2 条的规定，CGEL 属于修改后的衍生作品，并且不属于例外情况。因此，CGEL 在发布时需开放源代码，具体形式需要符合 GPL 协议第 3 条的规定。

## 7.2. 开发指导

为尽可能地保护公司在 Linux 方面的自有研发成果，特别是产品线的应用程序，同时顺从于 GPL 协议条款的规定，这里为大家提供一些开发指导原则供参考，以便有效地规避由于顺从 GPL 条款所带来的风险。

### ■ 应用程序尽可能运行于用户态

应用程序尽量放到用户态运行，应用程序对于内核的访问、功能使用主要通过信号、数据、文件系统、系统调用，基本属于松耦合，而且绝大部分系统调用都是通过 C 库封装进行，能比较充分地证明应用的独立性、可移植性，避免开源。

### ■ 以动态链接方式链接开源库

以 GLIBC 库为例，应用程序对于库的链接方式应尽量采用动态链接，这样可遵循 LGPL，避免公开源码。如果需要静态链接，则可以设计一个封装容器，与开源库静态链接在一起，这样只需开放封装容器源码。其余应用程序以动态链接方式与封装容器链接，只需要提供二进制文件。

### ■ 内核应用采用模块加载方式

内核应用可以采用静态链接、模块加载方式加入内。

静态链接方式是将所有的模块（包括应用）都编译到内核中，形成一个整体的内核映像文件。这种情况下，比较难以鉴别应用程序与内核的独立性、可移植性，按照 GPL 的规定，就很可能被要求开发源代码。

模块加载方式是指将上层应用独立地编译连接成标准 linux 所支持的模块文件 (\*.mod)。运行时，首先启动 Linux 内核，然后通过 insmod 命令将各模块按照彼此间的依赖关系插入到内核。该方式下应用程序相对独立于 Linux 内核，有可能被证明为独立的作品，与 OS 无关，认为是纯粹的聚集行为，从而可以

不用开放源码。

因此，对于运行于内核态的驱动和应用软件应尽量采用模块加载方式进行独立编译，通过模块的可动态装载、卸载，以及跨 OS 的可移植性来证明应用与 OS 的独立性。

## 7.3. 源码获取方式

广东中兴新支点所发布的 CGEL 操作系统是基于开源软件 Linux 2.6.21 版本开发的，遵从 GPL 协议开放源代码。如果你需要源码，请发送源码申请邮件到 [cgel@gd-linux.com](mailto:cgel@gd-linux.com) 联系获取源码。