LKP测试研究

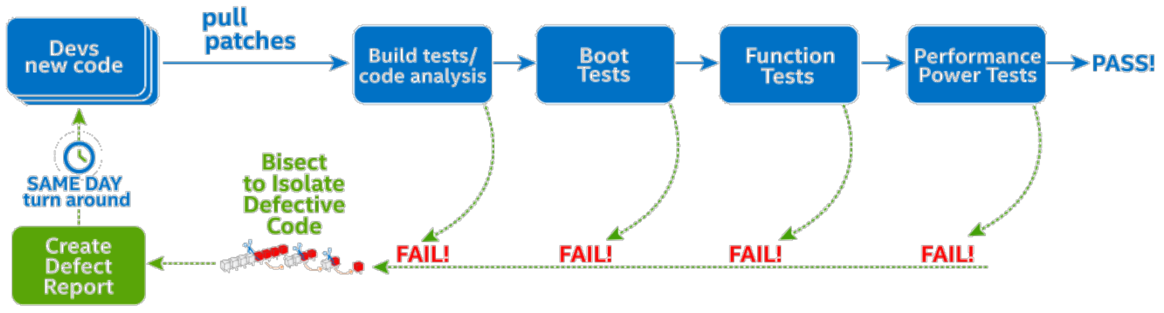
2017/6/15

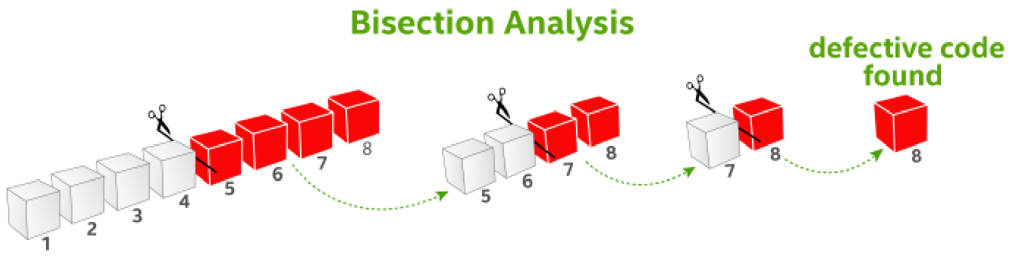
报告人；毛英明
指导老师:向勇

目录

目录

LKP原理介绍





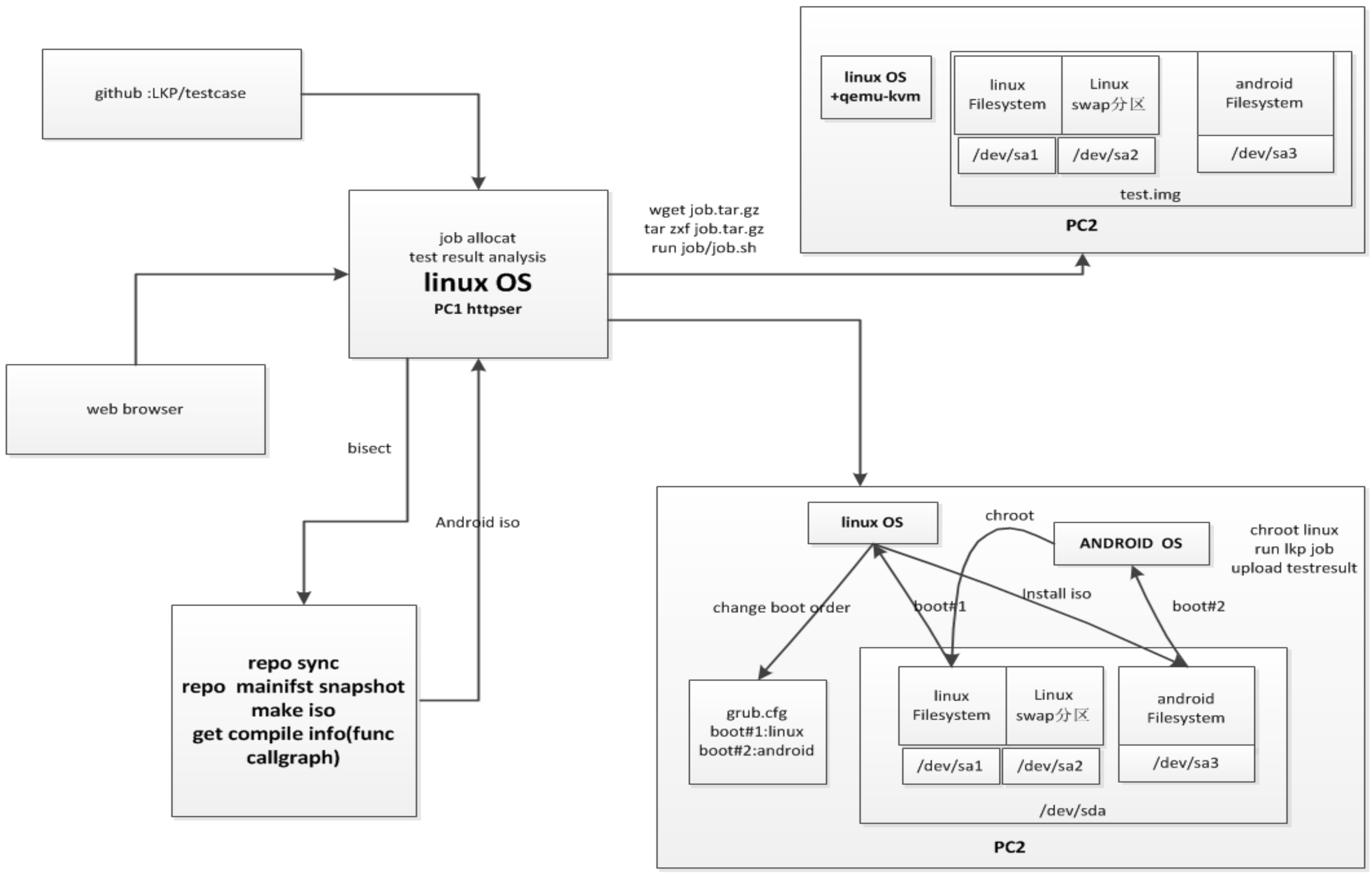
- https://01.org/zh/lkp   linux kernel performance
- 0-Day is unique in that it not only reports the failure, but also the specific patch that caused the failure, and the information to reproduce the problem.

LKP技术难点：

- 1.多次测试取平均值
- 2.找有正（负）相关的变量，如果一个变了另外一个也在变，那么就可以更加的确定是变了
- 3.系统预热(论文中提到了)
- 4.对比前后两次测试结果，如果发现有较大偏差，则认为出现了性能回退。则对内核代码进行bisect，找到出现问题的commit。
- 5.代码开源不彻底（远程测试（代码系统部署），测试数据分析，bisect都不开源）
- https://github.com/fengguang/lkp-tests
- https://github.com/fengguang/lkp-core

目录

将lkp应用于android测试



目录

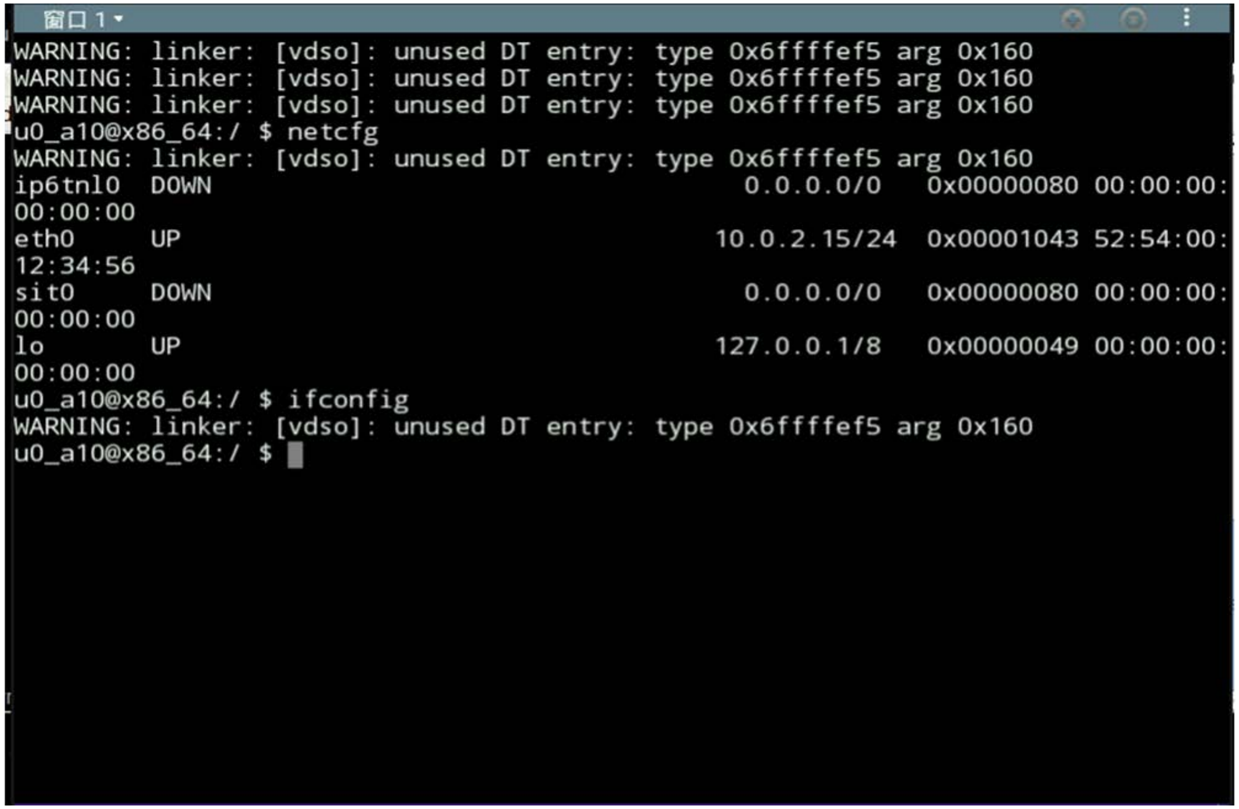
利用LKP测试应用

- 1.把benchmark替换为app【cli/gui app的自动运行,编写uiautomater测试脚本】
- 2.把mointor 替换为app配套的mointor收集app相关的性能信息
- 3.应用是否执行成功的判断,功能测试的覆盖【语义检查和功能覆盖】

- 临时解决方法：
- 1.应用是否正确执行，其实是语义正确性判断，application specific,交给了用户去做，如果用户发现软件或者系统执行不正确，再去用工具去分析原因。
- 2.用户重新执行应用（应用的自动化执行交给用户来做）的同时会启动strace追踪此应用相关的所有进程的系统调用信息。通过检测进程是否exit(>0)，以及是否write(2)，以及是否出现syscall error来推测可能的问题。将语义错误通过可能的进程\syscall执行错误来检查。
- 3.部分解决了测试用例需要一对一编写的问题。
- 4测试功能覆盖的问题依然没有解决。

目录

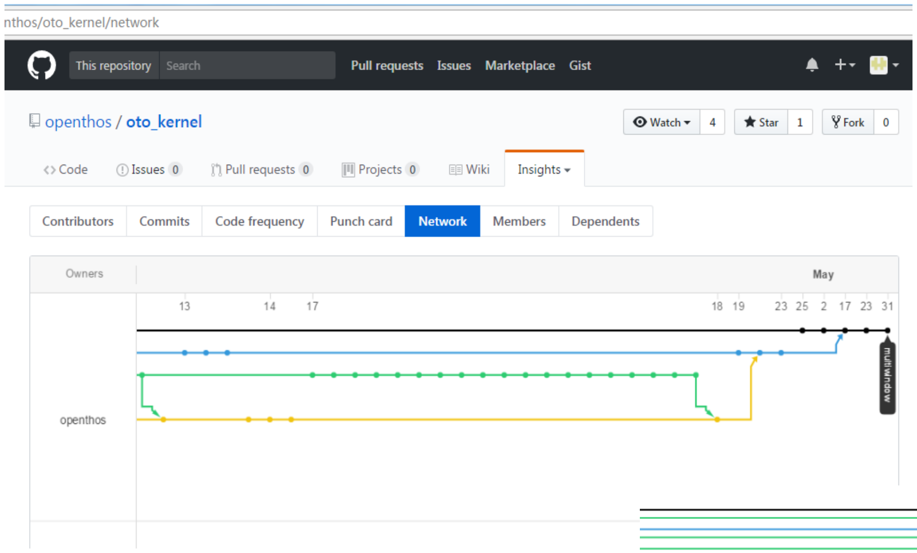
bionic linker warning

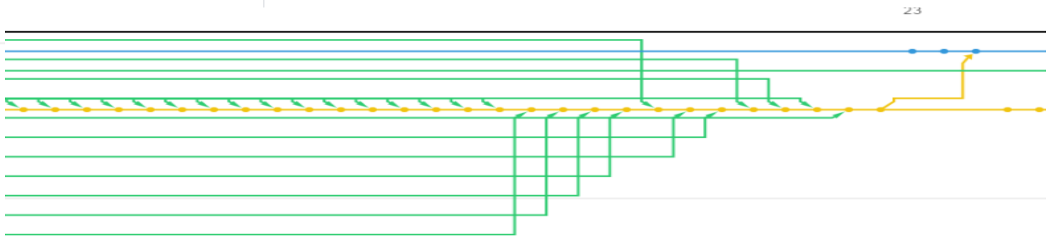


```
WARNING: linker: [vdso]: unused DT entry: type 0x6ffffef5 arg 0x160
WARNING: linker: [vdso]: unused DT entry: type 0x6ffffef5 arg 0x160
WARNING: linker: [vdso]: unused DT entry: type 0x6ffffef5 arg 0x160
u0_a10@x86_64:/ $ netcfg
WARNING: linker: [vdso]: unused DT entry: type 0x6ffffef5 arg 0x160
ip6tnl0  DOWN                            0.0.0.0/0   0x00000080 00:00:00:
00:00:00
eth0     UP                            10.0.2.15/24  0x00001043 52:54:00:
12:34:56
sit0     DOWN                            0.0.0.0/0   0x00000080 00:00:00:
00:00:00
lo       UP                            127.0.0.1/8   0x00000049 00:00:00:
00:00:00
u0_a10@x86_64:/ $ ifconfig
WARNING: linker: [vdso]: unused DT entry: type 0x6ffffef5 arg 0x160
u0_a10@x86_64:/ $ 
```

- kernel 4.4.10有warning
- 发现kernel 4.0.9没有waring（偶然发现）
- 因此推测是内核变动的原因。

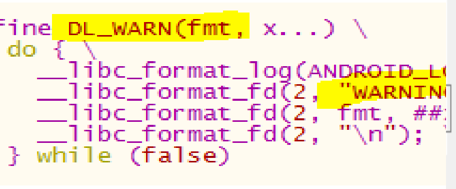
尝试bisect



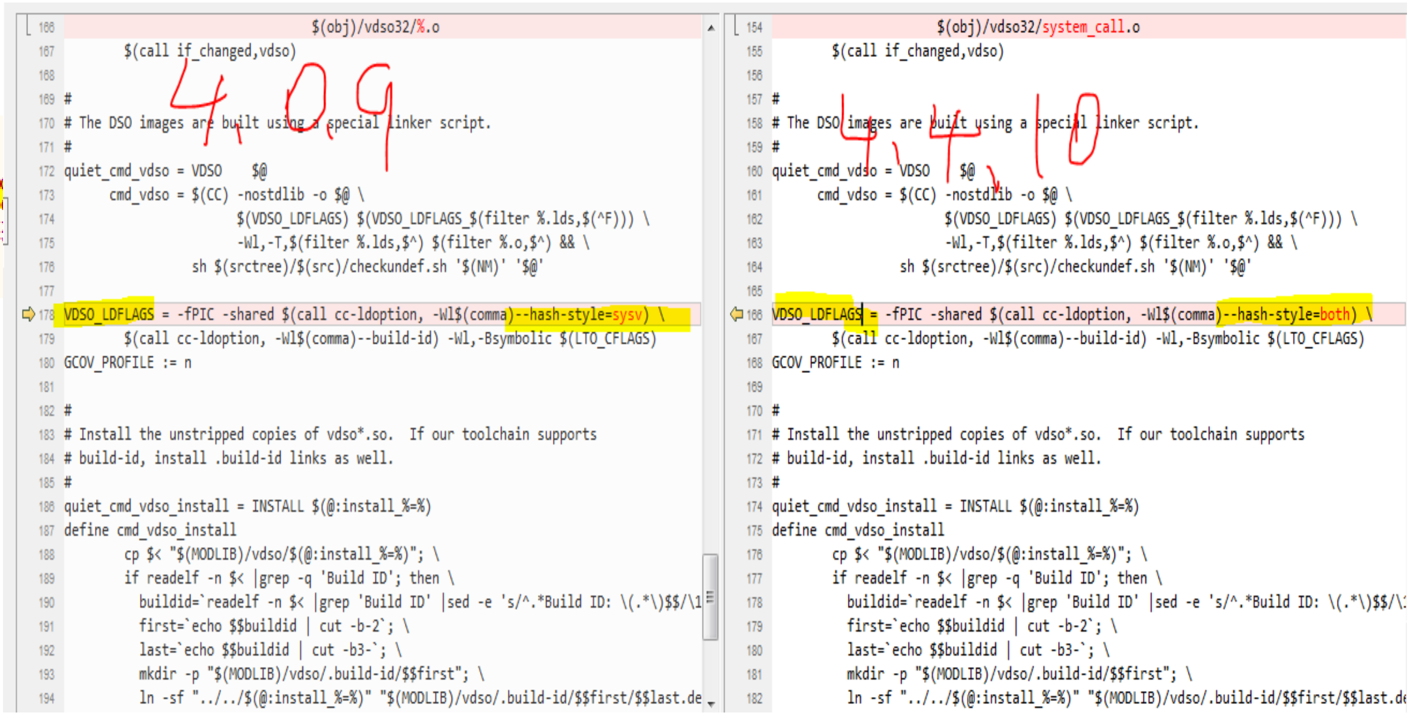


- commit的提交并不是线性的。
- cmt1, cmt2 ,cmt3,cmt4,......cmt5,cmt6,cmt7
- 问题发现的比较晚，问题并不是出现在两个commit或者多个线性的commit之间，而是出现在了两个不同的branch之间。
- git分支经过了切换无法断定该用哪个分支。 放弃了直接bisect。

rootcause定位



```
fine DL_WARN(fmt, x...) \
 do { \
    __libc_format_log(ANDROID_L
    __libc_format_fd(2, "WARNIN
    __libc_format_fd(2, fmt, ##
    __libc_format_fd(2, "\n");
 } while (false)
```



4.0.9

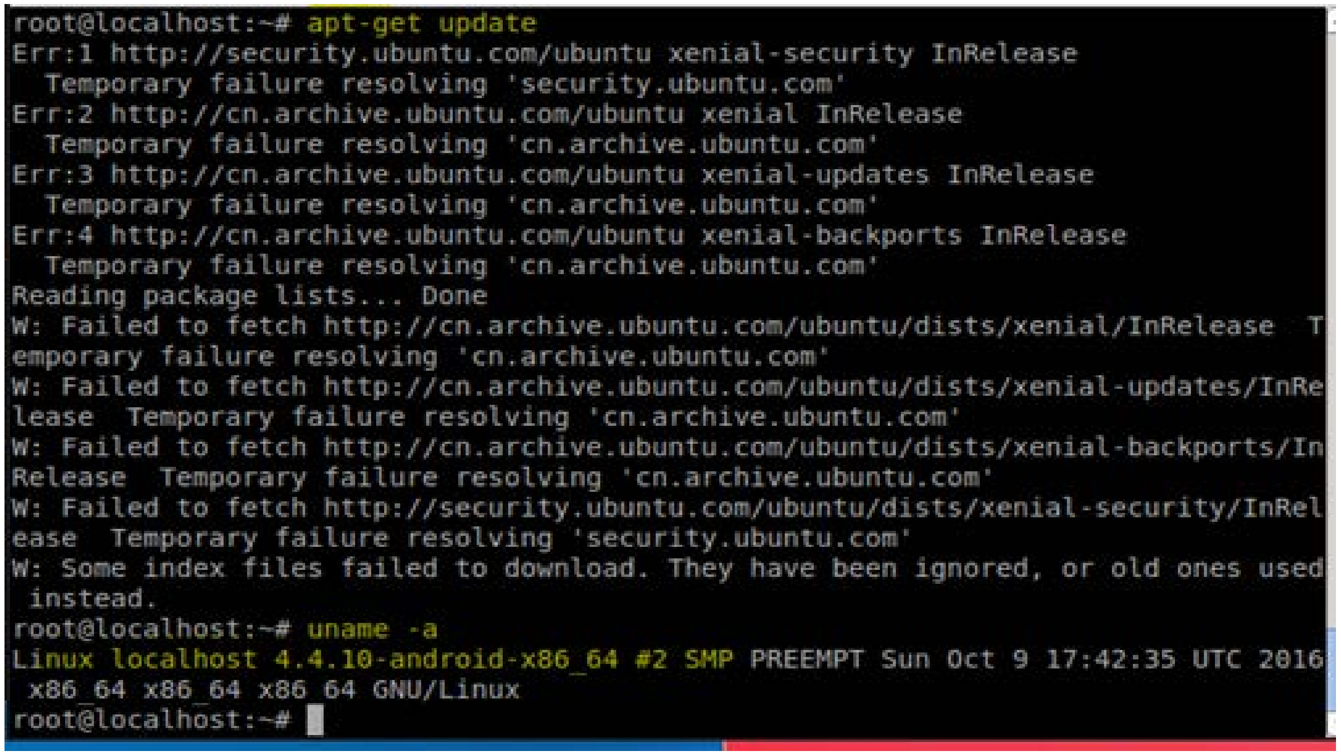
```
                      $(obj)/vdso32/%.o
        $(call if_changed,vdso)

#
# The DSO images are built using a special linker script.
#
quiet_cmd_vdso = VDSO     $@
      cmd_vdso = $(CC) -nostdlib -o $@ \
                       $(VDSO_LDFLAGS) $(VDSO_LDFLAGS_$(filter %.lds,$(^F))) \
                       -Wl,-T,$(filter %.lds,$^) $(filter %.o,$^) && \
                 sh $(srctree)/$(src)/checkundef.sh '$(NM)' '$@'

VDSO_LDFLAGS = -fPIC -shared $(call cc-ldoption, -Wl$(comma)--hash-style=sysv) \
        $(call cc-ldoption, -Wl$(comma)--build-id) -Wl,-Bsymbolic $(LTO_CFLAGS)
GCOV_PROFILE := n

#
# Install the unstripped copies of vdso*.so.  If our toolchain supports
# build-id, install .build-id links as well.
#
quiet_cmd_vdso_install = INSTALL $(@:install_%=%)
define cmd_vdso_install
        cp $< "$(MODLIB)/vdso/$(@:install_%=%)"; \
        if readelf -n $< |grep -q 'Build ID'; then \
          buildid=`readelf -n $< |grep 'Build ID' |sed -e 's/^.*Build ID: \(.*\)$$/\1
          first=`echo $$buildid | cut -b-2`; \
          last=`echo $$buildid | cut -b3-`; \
          mkdir -p "$(MODLIB)/vdso/.build-id/$$first"; \
          ln -sf "../../$(@:install_%=%)" "$(MODLIB)/vdso/.build-id/$$first/$$last.de
```

4.4.10

```
                      $(obj)/vdso32/system_call.o
        $(call if_changed,vdso)

#
# The DSO images are built using a special linker script.
#
quiet_cmd_vdso = VDSO     $@
      cmd_vdso = $(CC) -nostdlib -o $@ \
                       $(VDSO_LDFLAGS) $(VDSO_LDFLAGS_$(filter %.lds,$(^F))) \
                       -Wl,-T,$(filter %.lds,$^) $(filter %.o,$^) && \
                 sh $(srctree)/$(src)/checkundef.sh '$(NM)' '$@'

VDSO_LDFLAGS = -fPIC -shared $(call cc-ldoption, -Wl$(comma)--hash-style=both) \
        $(call cc-ldoption, -Wl$(comma)--build-id) -Wl,-Bsymbolic $(LTO_CFLAGS)
GCOV_PROFILE := n

#
# Install the unstripped copies of vdso*.so.  If our toolchain supports
# build-id, install .build-id links as well.
#
quiet_cmd_vdso_install = INSTALL $(@:install_%=%)
define cmd_vdso_install
        cp $< "$(MODLIB)/vdso/$(@:install_%=%)"; \
        if readelf -n $< |grep -q 'Build ID'; then \
          buildid=`readelf -n $< |grep 'Build ID' |sed -e 's/^.*Build ID: \(.*\)$$/\1
          first=`echo $$buildid | cut -b-2`; \
          last=`echo $$buildid | cut -b3-`; \
          mkdir -p "$(MODLIB)/vdso/.build-id/$$first"; \
          ln -sf "../../$(@:install_%=%)" "$(MODLIB)/vdso/.build-id/$$first/$$last.de
```

- 通过strace发现在execve之后，main函数执行之前，执行的write(2)。那么说明这段代码应该属于linker管辖。
- 通过cat /proc/pid/maps 和 readelf, file, ldd都可以知道此可执行文件对应的linker文件的绝对路径。然后以warning内容搜索linker程序对应的源代码(bionic)。通过gdb在warning处下断点，栈回溯找到了出错路径。
- 发现内核vdso文件多了不认识的entry。推测是内核makefile变动。最后只diff了vdso部分的makefile发现了问题。

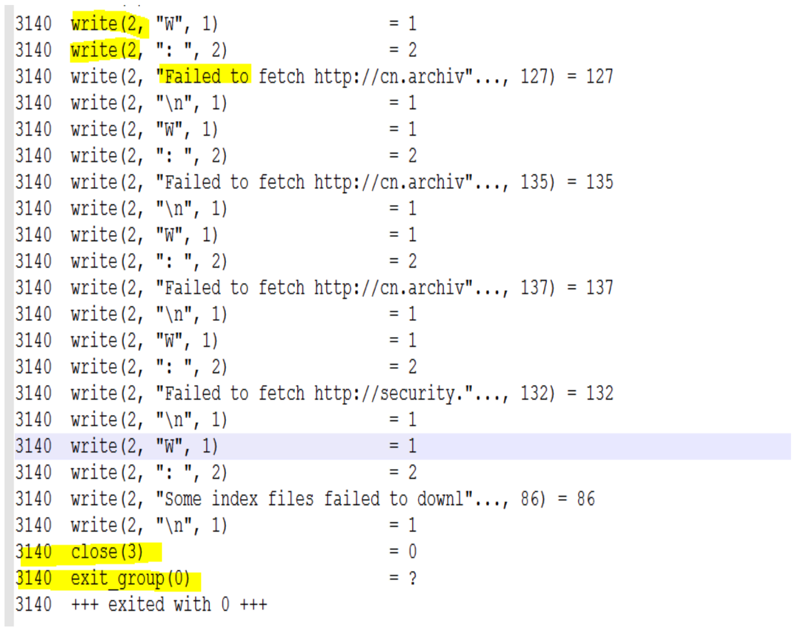
apt-get无法使用



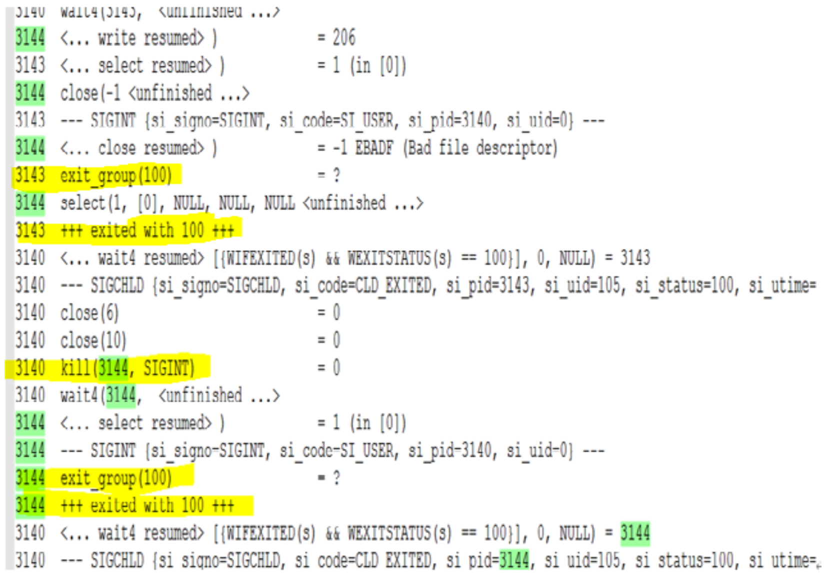
```
root@localhost:~# apt-get update
Err:1 http://security.ubuntu.com/ubuntu xenial-security InRelease
  Temporary failure resolving 'security.ubuntu.com'
Err:2 http://cn.archive.ubuntu.com/ubuntu xenial InRelease
  Temporary failure resolving 'cn.archive.ubuntu.com'
Err:3 http://cn.archive.ubuntu.com/ubuntu xenial-updates InRelease
  Temporary failure resolving 'cn.archive.ubuntu.com'
Err:4 http://cn.archive.ubuntu.com/ubuntu xenial-backports InRelease
  Temporary failure resolving 'cn.archive.ubuntu.com'
Reading package lists... Done
W: Failed to fetch http://cn.archive.ubuntu.com/ubuntu/dists/xenial/InRelease  T
emporary failure resolving 'cn.archive.ubuntu.com'
W: Failed to fetch http://cn.archive.ubuntu.com/ubuntu/dists/xenial-updates/InRe
lease  Temporary failure resolving 'cn.archive.ubuntu.com'
W: Failed to fetch http://cn.archive.ubuntu.com/ubuntu/dists/xenial-backports/In
Release  Temporary failure resolving 'cn.archive.ubuntu.com'
W: Failed to fetch http://security.ubuntu.com/ubuntu/dists/xenial-security/InRel
ease  Temporary failure resolving 'security.ubuntu.com'
W: Some index files failed to download. They have been ignored, or old ones used
 instead.
root@localhost:~# uname -a
Linux localhost 4.4.10-android-x86_64 #2 SMP PREEMPT Sun Oct 9 17:42:35 UTC 2016
 x86_64 x86_64 x86_64 GNU/Linux
root@localhost:~#
```

- kernel 4.4.10 ubuntu16.04 apt-get无法使用。
- kernel 4.0.9 ubuntu16.04 apt-get可以使用（偶然发现）
- 因此可以断定是内核的原因。同样由于代码版本差异太大，没有直接使用bisect。

rootcause的定位



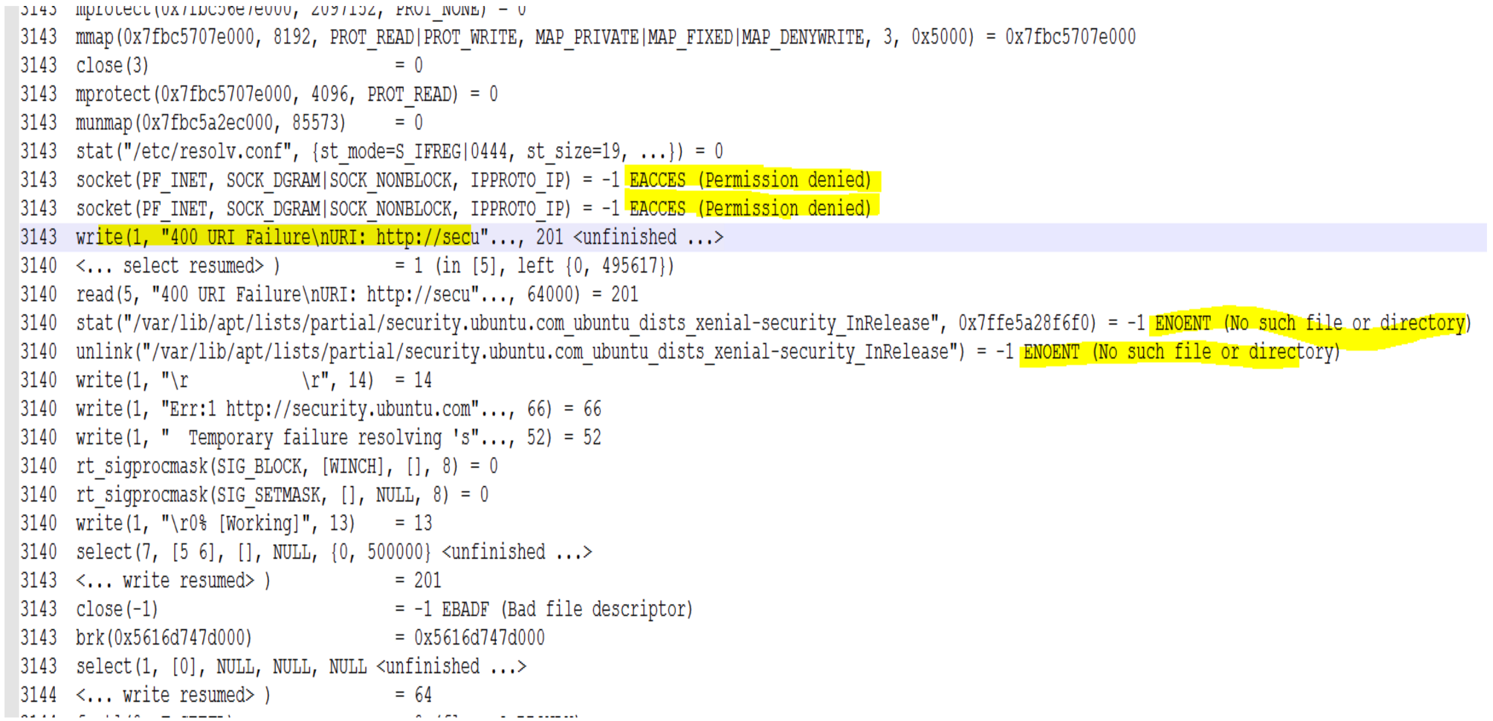
```
3140  write(2, "W", 1)                   = 1
3140  write(2, ": ", 2)                  = 2
3140  write(2, "Failed to fetch http://cn.archiv"..., 127) = 127
3140  write(2, "\n", 1)                  = 1
3140  write(2, "W", 1)                   = 1
3140  write(2, ": ", 2)                  = 2
3140  write(2, "Failed to fetch http://cn.archiv"..., 135) = 135
3140  write(2, "\n", 1)                  = 1
3140  write(2, "W", 1)                   = 1
3140  write(2, ": ", 2)                  = 2
3140  write(2, "Failed to fetch http://cn.archiv"..., 137) = 137
3140  write(2, "\n", 1)                  = 1
3140  write(2, "W", 1)                   = 1
3140  write(2, ": ", 2)                  = 2
3140  write(2, "Failed to fetch http://security."..., 132) = 132
3140  write(2, "\n", 1)                  = 1
3140  write(2, "W", 1)                   = 1
3140  write(2, ": ", 2)                  = 2
3140  write(2, "Some index files failed to downl"..., 86) = 86
3140  write(2, "\n", 1)                  = 1
3140  close(3)                           = 0
3140  exit_group(0)                      = ?
3140  +++ exited with 0 +++
```



```
3140  wait4(3143,  <unfinished ...>
3144  <... write resumed> )              = 206
3143  <... select resumed> )             = 1 (in [0])
3144  close(-1 <unfinished ...>
3143  --- SIGINT {si_signo=SIGINT, si_code=SI_USER, si_pid=3140, si_uid=0} ---
3144  <... close resumed> )              = -1 EBADF (Bad file descriptor)
3143  exit_group(100)                    = ?
3144  select(1, [0], NULL, NULL, NULL <unfinished ...>
3143  +++ exited with 100 +++
3140  <... wait4 resumed> [{WIFEXITED(s) && WEXITSTATUS(s) == 100}], 0, NULL) = 3143
3140  --- SIGCHLD {si_signo=SIGCHLD, si_code=CLD_EXITED, si_pid=3143, si_uid=105, si_status=100, si_utime=
3140  close(6)                           = 0
3140  close(10)                          = 0
3140  kill(3144, SIGINT)                 = 0
3140  wait4(3144,  <unfinished ...>
3144  <... select resumed> )             = 1 (in [0])
3144  --- SIGINT {si_signo=SIGINT, si_code=SI_USER, si_pid=3140, si_uid=0} ---
3144  exit_group(100)                    = ?
3144  +++ exited with 100 +++
3140  <... wait4 resumed> [{WIFEXITED(s) && WEXITSTATUS(s) == 100}], 0, NULL) = 3144
3140  --- SIGCHLD {si_signo=SIGCHLD, si_code=CLD_EXITED, si_pid=3144, si_uid=105, si_status=100, si_utime=
```

- 查找write(2)和exit(>0)相关信息

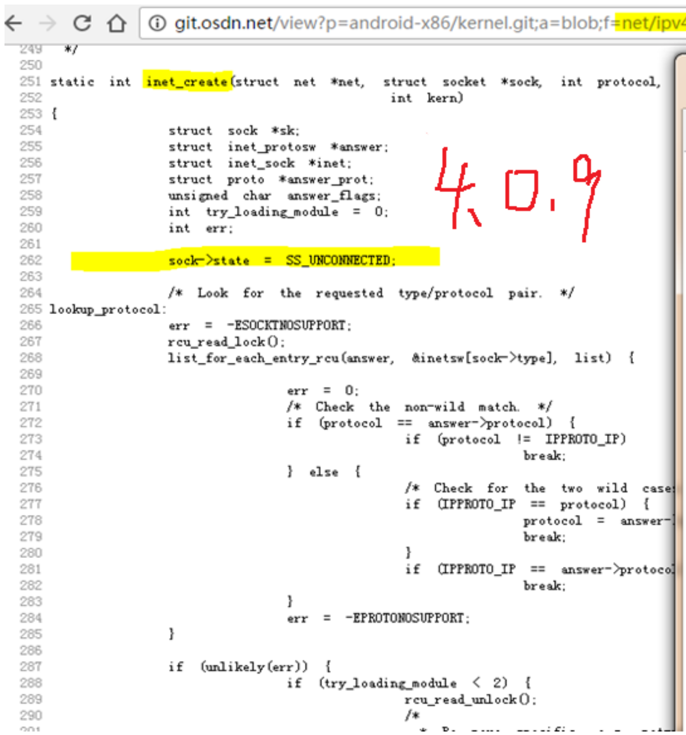
原因分析



```
3143  mprotect(0x7fbc56e7e000, 2097152, PROT_NONE) = 0
3143  mmap(0x7fbc5707e000, 8192, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x5000) = 0x7fbc5707e000
3143  close(3)                          = 0
3143  mprotect(0x7fbc5707e000, 4096, PROT_READ) = 0
3143  munmap(0x7fbc5a2ec000, 85573)     = 0
3143  stat("/etc/resolv.conf", {st_mode=S_IFREG|0444, st_size=19, ...}) = 0
3143  socket(PF_INET, SOCK_DGRAM|SOCK_NONBLOCK, IPPROTO_IP) = -1 EACCES (Permission denied)
3143  socket(PF_INET, SOCK_DGRAM|SOCK_NONBLOCK, IPPROTO_IP) = -1 EACCES (Permission denied)
3143  write(1, "400 URI Failure\nURI: http://secu"..., 201 <unfinished ...>
3140  <... select resumed> )            = 1 (in [5], left {0, 495617})
3140  read(5, "400 URI Failure\nURI: http://secu"..., 64000) = 201
3140  stat("/var/lib/apt/lists/partial/security.ubuntu.com_ubuntu_dists_xenial-security_InRelease", 0x7ffe5a28f6f0) = -1 ENOENT (No such file or directory)
3140  unlink("/var/lib/apt/lists/partial/security.ubuntu.com_ubuntu_dists_xenial-security_InRelease") = -1 ENOENT (No such file or directory)
3140  write(1, "\r            \r", 14)  = 14
3140  write(1, "Err:1 http://security.ubuntu.com"..., 66) = 66
3140  write(1, "  Temporary failure resolving 's"..., 52) = 52
3140  rt_sigprocmask(SIG_BLOCK, [WINCH], [], 8) = 0
3140  rt_sigprocmask(SIG_SETMASK, [], NULL, 8) = 0
3140  write(1, "\r0% [Working]", 13)     = 13
3140  select(7, [5 6], [], NULL, {0, 500000} <unfinished ...>
3143  <... write resumed> )             = 201
3143  close(-1)                         = -1 EBADF (Bad file descriptor)
3143  brk(0x5616d747d000)               = 0x5616d747d000
3143  select(1, [0], NULL, NULL, NULL <unfinished ...>
3144  <... write resumed> )             = 64
```

- apt-get 命令分为多个进程，有的进程负责下载软包信息列表，有的进程负责更新本地数据库。但是负责下载软件包信息列表的进程，在连接网络(解析dns)的时候出现了错误，和dns server的socket连接创建失败，导致进程退出，最终任务执行失败。
- 出错码EACCES

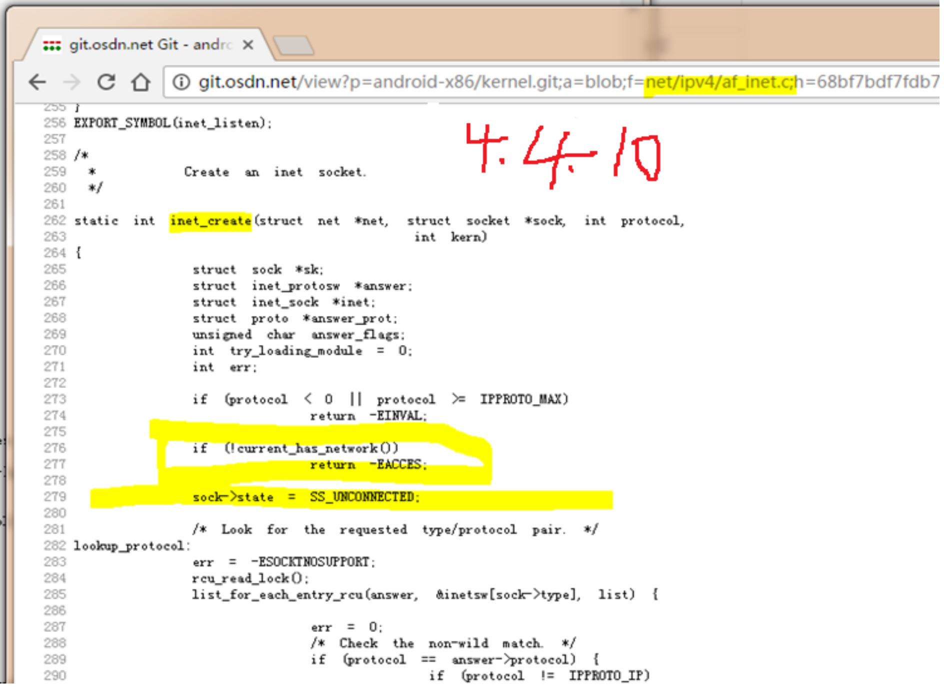
Kernel source code diff



git.osdn.net/view?p=android-x86/kernel.git;a=blob;f=net/ipv4/af_inet.c;h=61edc496b7d0991b555d8623de5ed62b84d436e

4.0.9

```
 */

static int inet_create(struct net *net, struct socket *sock, int protocol,
                       int kern)
{
        struct sock *sk;
        struct inet_protosw *answer;
        struct inet_sock *inet;
        struct proto *answer_prot;
        unsigned char answer_flags;
        int try_loading_module = 0;
        int err;

        sock->state = SS_UNCONNECTED;

        /* Look for the requested type/protocol pair. */
lookup_protocol:
        err = -ESOCKTNOSUPPORT;
        rcu_read_lock();
        list_for_each_entry_rcu(answer, &inetsw[sock->type], list) {

                err = 0;
                /* Check the non-wild match. */
                if (protocol == answer->protocol) {
                        if (protocol != IPPROTO_IP)
                                break;
                } else {
                        /* Check for the two wild case
                        if (IPPROTO_IP == protocol) {
                                protocol = answer-
                                break;
                        }
                        if (IPPROTO_IP == answer->protoco
                                break;
                }
                err = -EPROTONOSUPPORT;
        }

        if (unlikely(err)) {
                if (try_loading_module < 2) {
                        rcu_read_unlock();
                        /*
```



git.osdn.net Git - andr

git.osdn.net/view?p=android-x86/kernel.git;a=blob;f=net/ipv4/af_inet.c;h=68bf7bdf7fdb7

4.4.10

```
}
EXPORT_SYMBOL(inet_listen);

/*
 *      Create an inet socket.
 */

static int inet_create(struct net *net, struct socket *sock, int protocol,
                       int kern)
{
        struct sock *sk;
        struct inet_protosw *answer;
        struct inet_sock *inet;
        struct proto *answer_prot;
        unsigned char answer_flags;
        int try_loading_module = 0;
        int err;

        if (protocol < 0 || protocol >= IPPROTO_MAX)
                return -EINVAL;

        if (!current_has_network())
                return -EACCES;

        sock->state = SS_UNCONNECTED;

        /* Look for the requested type/protocol pair. */
lookup_protocol:
        err = -ESOCKTNOSUPPORT;
        rcu_read_lock();
        list_for_each_entry_rcu(answer, &inetsw[sock->type], list) {

                err = 0;
                /* Check the non-wild match. */
                if (protocol == answer->protocol) {
                        if (protocol != IPPROTO_IP)
```

- 根据socket syscall函数向内核态的调用路径进行追踪，发现了两个内核版本的inet_create函数实现不同。出错码EACCES也匹配上了
- syscall和函数的api都没有发生变化，但是函数的语义发生了变化。

回归测试总结

- 直接bisect有可能代码版本差异太大，太复杂，不好实现，但是如果有指导信息，则可以利用指导信息进行带指导的sourcecode diff，能够迅速的缩小diff范围，加快rootcause定位速度。
- 出错要找出错误的触发路径。
- 不仅仅要获取普通的monitor信息，还要获取一些，静态信息、动态信息
- LKP只是一个测试框架。要测试出有价值的信息，还需要在测试用例的设计，测试结果的分析上面多做工作。

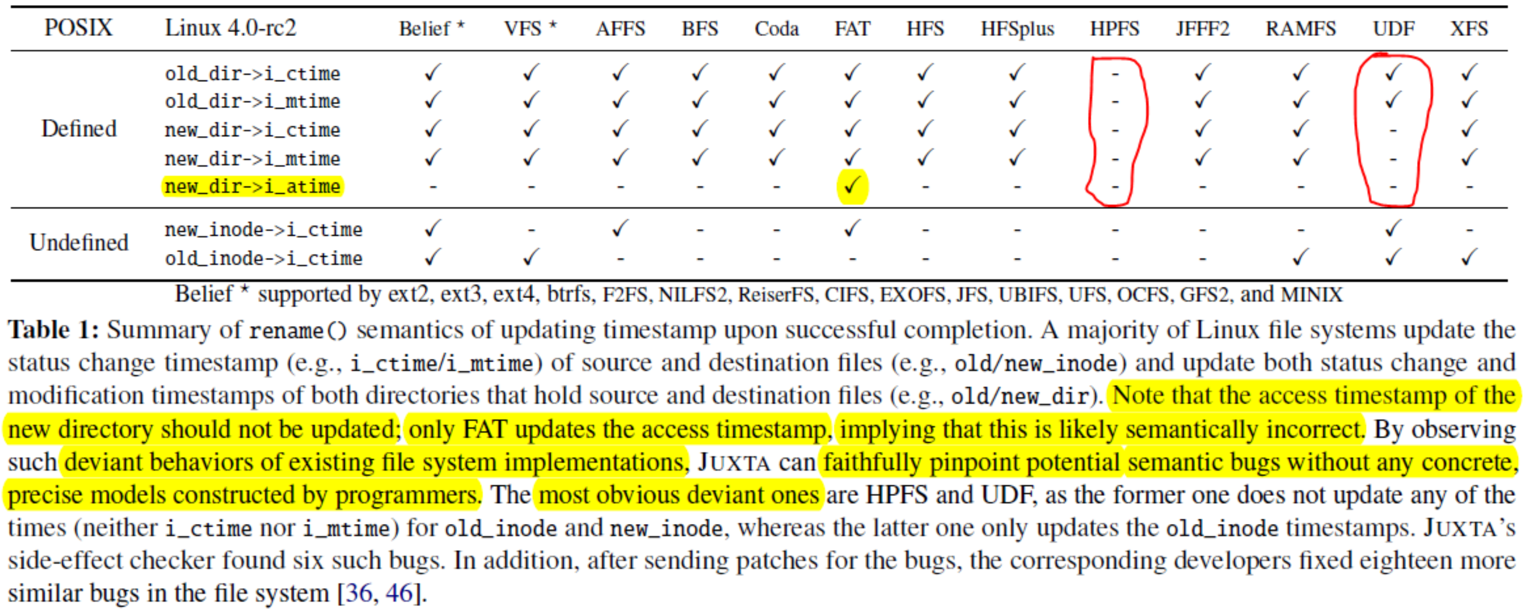
目录

语义测试相关工作

- SOSP’15 Cross-checking Semantic Correctness: The Case of Finding File System Bugs

- 什么是bug?
- deviant behavior or as a different behavior among multiple implementations
- 什么是shallow bug 什么是semantic bug （需要domain specific knowledge）
- 真理掌握在多数人的手中。
- corsscheck就是交叉检测，利用一种标准的多种实现，进行对比评价，找出不合群的实现的点。对VFS多种不同的文件系统实现进行交叉检测。找出正确的语义实现（大众）是什么样的，以及错误的实现（小众）。

rename("old_dir/a", "new_dir/b");



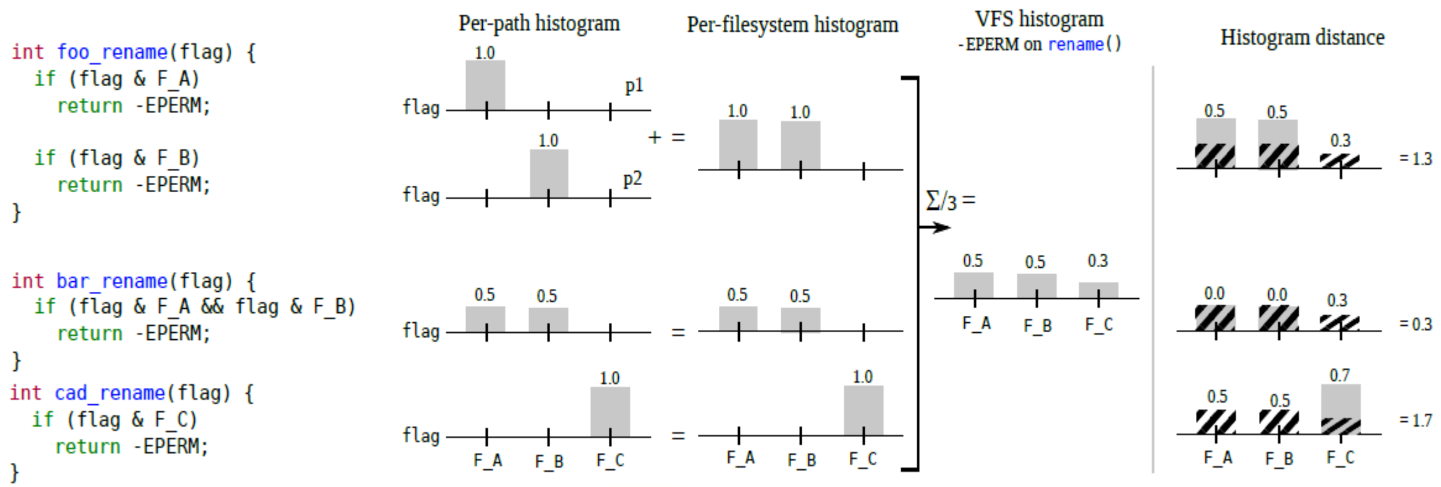
POSIX	Linux 4.0-rc2	Belief *	VFS *	AFFS	BFS	Coda	FAT	HFS	HFSplus	HPFS	JFFF2	RAMFS	UDF	XFS
Defined	old_dir->i_ctime	✓	✓	✓	✓	✓	✓	✓	✓	-	✓	✓	✓	✓
	old_dir->i_mtime	✓	✓	✓	✓	✓	✓	✓	✓	-	✓	✓	✓	✓
	new_dir->i_ctime	✓	✓	✓	✓	✓	✓	✓	✓	-	✓	✓	-	✓
	new_dir->i_mtime	✓	✓	✓	✓	✓	✓	✓	✓	-	✓	✓	-	✓
	new_dir->i_atime	-	-	-	-	-	✓	-	-	-	-	-	-	-
Undefined	new_inode->i_ctime	✓	-	✓	-	-	✓	-	-	-	-	-	✓	-
	old_inode->i_ctime	✓	✓	-	-	-	-	-	-	-	-	✓	✓	✓

Belief * supported by ext2, ext3, ext4, btrfs, F2FS, NILFS2, ReiserFS, CIFS, EXOFS, JFS, UBIFS, UFS, OCFS, GFS2, and MINIX

Table 1: Summary of rename() semantics of updating timestamp upon successful completion. A majority of Linux file systems update the status change timestamp (e.g., i_ctime/i_mtime) of source and destination files (e.g., old/new_inode) and update both status change and modification timestamps of both directories that hold source and destination files (e.g., old/new_dir). Note that the access timestamp of the new directory should not be updated; only FAT updates the access timestamp, implying that this is likely semantically incorrect. By observing such deviant behaviors of existing file system implementations, JUXTA can faithfully pinpoint potential semantic bugs without any concrete, precise models constructed by programmers. The most obvious deviant ones are HPFS and UDF, as the former one does not update any of the times (neither i_ctime nor i_mtime) for old_inode and new_inode, whereas the latter one only updates the old_inode timestamps. JUXTA's side-effect checker found six such bugs. In addition, after sending patches for the bugs, the corresponding developers fixed eighteen more similar bugs in the file system [36, 46].

- ctime指inode上一次变动的时间，mtime指文件内容上一次变动的时间，atime指文件上一次打开的时间。
- 技术难点：符号执行。本身各种文件系统实现就是不同的，如何进行抽象的比较。统计直方图的比较算法。取平均值，然后计算各个点和平均值的差距，差距最大的就是变异（deviant）的。

Histogram-based comparison



```
int foo_rename(flag) {
  if (flag & F_A)
    return -EPERM;

  if (flag & F_B)
    return -EPERM;
}

int bar_rename(flag) {
  if (flag & F_A && flag & F_B)
    return -EPERM;
}
int cad_rename(flag) {
  if (flag & F_C)
    return -EPERM;
}
```

Figure 4: Histogram representation of `rename()` in three contrived file systems (`foo`, `bar` and `cad`) on the `-EPERM` path. Histogram-based comparison represents a universal variable (`flag`) as one dimension (x-axis). To describe `rename()`, it computes the average of all file system histograms, and deducts the average from per-file system histograms for comparison. For example, `foo` is sensitive (+0.5) and `cad` is insensitive (−0.5) on the `F_A` flag. Globally, `cad` behaves the most differently (deviant) from `foo` and `bar` (1.7) in terms of `-EPERM` path.

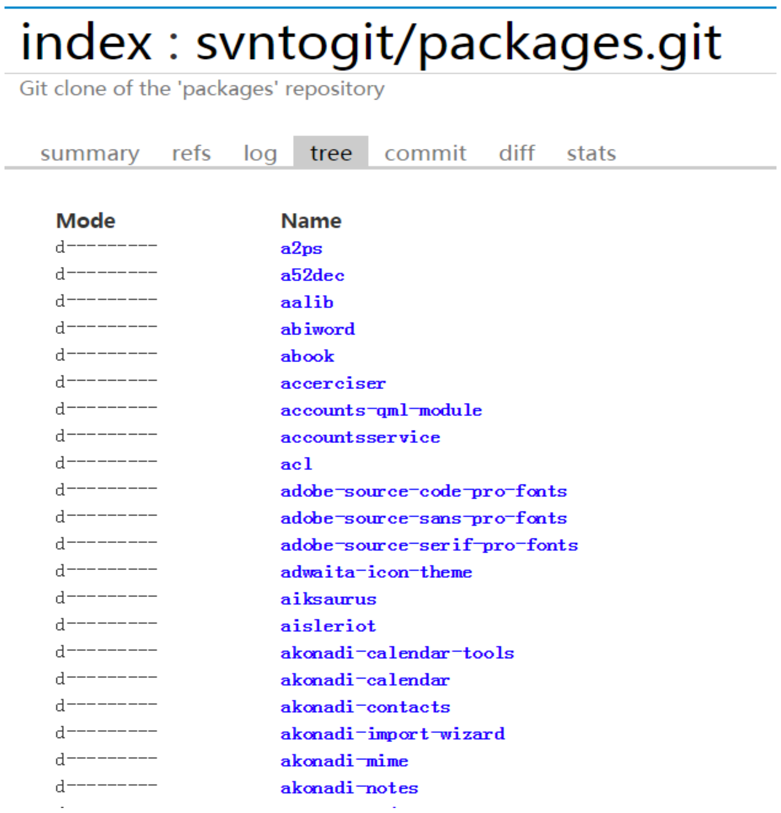
- 最后文献的结尾提到，可以用于回归测试。一个kernel的多种版本的实现，来比较差异。我们可以通过多次测试找离群点。多次测试，过程中，代码发生了变化，如果有离群点，则表示这次测试是比较特殊的，需要特别留意。不管是否正确。
- 首先相同的环境测试多次，看看执行执行路径里面有哪些是恒定不变的？哪些是变的？噪音？pid是变的，时间戳是变的…。

测试覆盖问题

- 动态执行，很难进行覆盖测试，
- 需要静态分析得出其有限状态机模型。
- （根据所有的callback函数来生成testcase）
- 理论上静态分析能够覆盖的比较全一些。此时需要利用到符号执行技术

目录

archlinux的测试

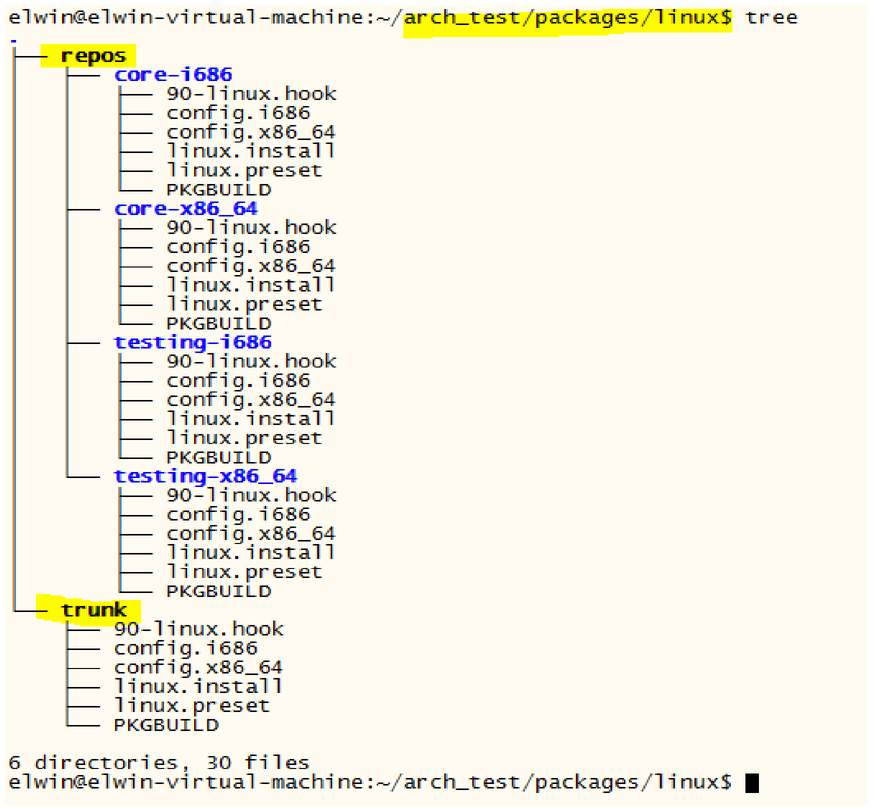


index : svntogit/packages.git

Git clone of the 'packages' repository

summary refs log tree commit diff stats

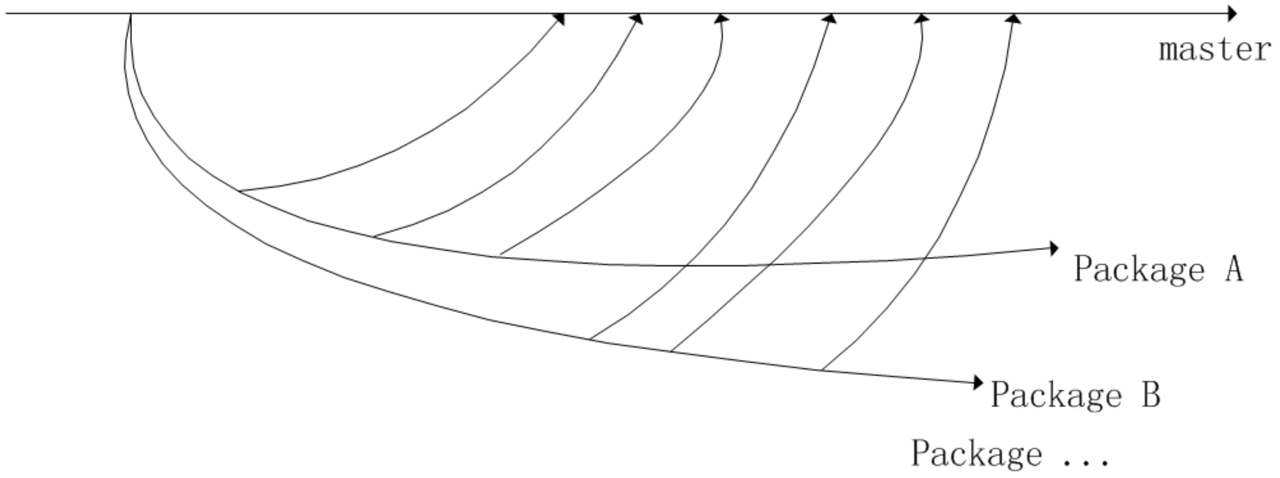
Mode	Name
d---------	a2ps
d---------	a52dec
d---------	aalib
d---------	abiword
d---------	abook
d---------	accerciser
d---------	accounts-qml-module
d---------	accountsservice
d---------	acl
d---------	adobe-source-code-pro-fonts
d---------	adobe-source-sans-pro-fonts
d---------	adobe-source-serif-pro-fonts
d---------	adwaita-icon-theme
d---------	aiksaurus
d---------	aisleriot
d---------	akonadi-calendar-tools
d---------	akonadi-calendar
d---------	akonadi-contacts
d---------	akonadi-import-wizard
d---------	akonadi-mime
d---------	akonadi-notes



```
elwin@elwin-virtual-machine:~/arch_test/packages/linux$ tree
.
├── repos
│   ├── core-i686
│   │   ├── 90-linux.hook
│   │   ├── config.i686
│   │   ├── config.x86_64
│   │   ├── linux.install
│   │   ├── linux.preset
│   │   └── PKGBUILD
│   ├── core-x86_64
│   │   ├── 90-linux.hook
│   │   ├── config.i686
│   │   ├── config.x86_64
│   │   ├── linux.install
│   │   ├── linux.preset
│   │   └── PKGBUILD
│   ├── testing-i686
│   │   ├── 90-linux.hook
│   │   ├── config.i686
│   │   ├── config.x86_64
│   │   ├── linux.install
│   │   ├── linux.preset
│   │   └── PKGBUILD
│   └── testing-x86_64
│       ├── 90-linux.hook
│       ├── config.i686
│       ├── config.x86_64
│       ├── linux.install
│       ├── linux.preset
│       └── PKGBUILD
└── trunk
    ├── 90-linux.hook
    ├── config.i686
    ├── config.x86_64
    ├── linux.install
    ├── linux.preset
    └── PKGBUILD

6 directories, 30 files
elwin@elwin-virtual-machine:~/arch_test/packages/linux$ █
```

- packages.git 类似于repo，通过多个分支来管理不同的软件包。所有分支结果在master分支上的合并形成了整个软件仓库的信息。
- https://git.archlinux.org/svntogit/packages.git

packages.git



- 软件滚动更新，master分支和各个分支上面的软件包都是最新的。
- 软件仓库可以通过git来追溯版本信息。
- makepkg -s自动安装编译依赖和运行依赖。
- pacman -U / -S 安装软件包更方便，可以单独安装软件包，可以直接升级内核，重新启动就可以使用新的内核。
- 不需要双系统，chroot环境，更简单一些

批评指正

- Q&A
- 谢谢！