Unit-testing Busybox ls

- We performed unit-testing Busybox `ls` by using CROWN
  - We tested 14 functions of Busybox ls (1100 lines long)
  - Note that this is a refined testing activity compared to the previous testing activity for 10 Busybox utilities in a system-level testing

Busybox `ls` Requirement Specification

- POSIX specification (IEEE Std 1003.1, 2004 ed.) is a good requirement specification document for ls
  - A4 ~10 page description for all options

- We defined test oracles using **`assert`** statements based on the POSIX specification
  - However, it still required human expertise on Busybox ls code to define concrete assert statements from given high-level requirements

The Open Group Base Specifications Issue 6
IEEE Std 1003.1, 2004 Edition

NAME

ls - list directory contents

SYNOPSIS

```
[XSI] ls [-CFRacdilqrtu1][-H | -L ]⌦[-fgmnopsx]⌫[file...]
```

DESCRIPTION

For each operand that names a file of a type other than directory or symbolic link to a directory, *ls* shall write the name of the file as well as any requested, associated information. For each operand that names a file of type directory, *ls* shall write the names of files contained within the directory as well as any requested, associated information. If one of the **-d**, **-F**, or **-l** options are specified, and one of the **-H** or **-L** options are not specified, for each operand that names a file of type symbolic link to a directory, *ls* shall write the name of the file as well as any requested, associated information. If none of the **-d**, **-F**, or **-l** options are specified, or the **-H** or **-L** options are specified, for each operand that names a file of type symbolic link to a directory, *ls* shall write the names of files contained within the directory as well as any requested, associated information.

If no operands are specified, *ls* shall write the contents of the current directory. If more than one operand is specified, *ls* shall write non-directory operands first; it shall sort directory and non-directory operands separately according to the collating sequence in the current locale.

The *ls* utility shall detect infinite loops; that is, entering a previously visited directory that is an ancestor of the last file encountered. When it detects an infinite loop, *ls* shall write a diagnostic message to standard error and shall either recover its position in the hierarchy or terminate.

OPTIONS

The *ls* utility shall conform to the Base Definitions volume of IEEE Std 1003.1-2001, Section 12.2, Utility Syntax Guidelines.

The following options shall be supported:

-C Write multi-text-column output with entries sorted down the columns, according to the collating sequence. The number of text columns and the column separator characters are unspecified, but should be adapted to the nature of the output device.

-F Do not follow symbolic links named as operands unless the **-H** or **-L** options are specified. Write a slash ('/') immediately after each pathname that is a directory, an asterisk ('*') after each that is executable, a vertical bar ('|') after each that is a FIFO, and an at sign ('@') after each that is a symbolic link. For other file types, other symbols may be written.

4 Bugs Detected

1. Missing `@` symbol for a symbolic link file with `-F` option
2. Missing space between adjacent two columns with `-i` or `-b` options
3. The order of options is ignored
   - According to the ls specification, the last option should have a higher priority (i.e., `-C -1` and `-1 -C` are different)
4. Option `-n` does not show files in a long format
   - `-n` enforces to list files in a long format and print numeric UID and GID instead of user/group name

Examples for the 4 Bugs Detected

1. Missing '@' symbol for a symbolic link file with –F option

Output of Linux ls

```
$ ls -F t.lnk
t.lnk@
```

Output of Busybox ls (incorrect behavior)

```
$ ./busybox ls -F t.lnk
t.lnk
```

2. Missing space between adjacent two columns with –i or –b options

Output of Linux ls

```
$ ls -i ~user/12345 ~user/11111
154930324 /home/user/11111 🟥154930124
    /home/user/12345
```

Output of Busybox ls (incorrect behavior)

```
$ ./busybox ls -i ~user/12345 ~user/11111
    154930324 /home/user/11111154930124
    /home/user/12345
```

3. The order of options is ignored

Output of Linux ls

```
$ ls -1C
a.txt b.txt
```

Output of Busybox ls (incorrect behavior)

```
$ ./busybox ls -1C
a.txt
b.txt
```

4. –n does not show files in a long format

Output of Linux ls

```
$ ls –n a.txt
-rw-r--r-- 1 1000 1000 5833 Jun 24  2010 a.txt
```

Output of Busybox ls (incorrect behavior)

```
$ ./busybox ls –n a.txt
a.txt
```

Missing '@' symbol for symbolic link with –F option

- Busybox ls does not print a type marker '@' after a symbolic link file name, when -F is specified and a file name is specified in the command line.

1. Output of linux ls:

```
$ ls -F t.lnk
t.lnk@
```

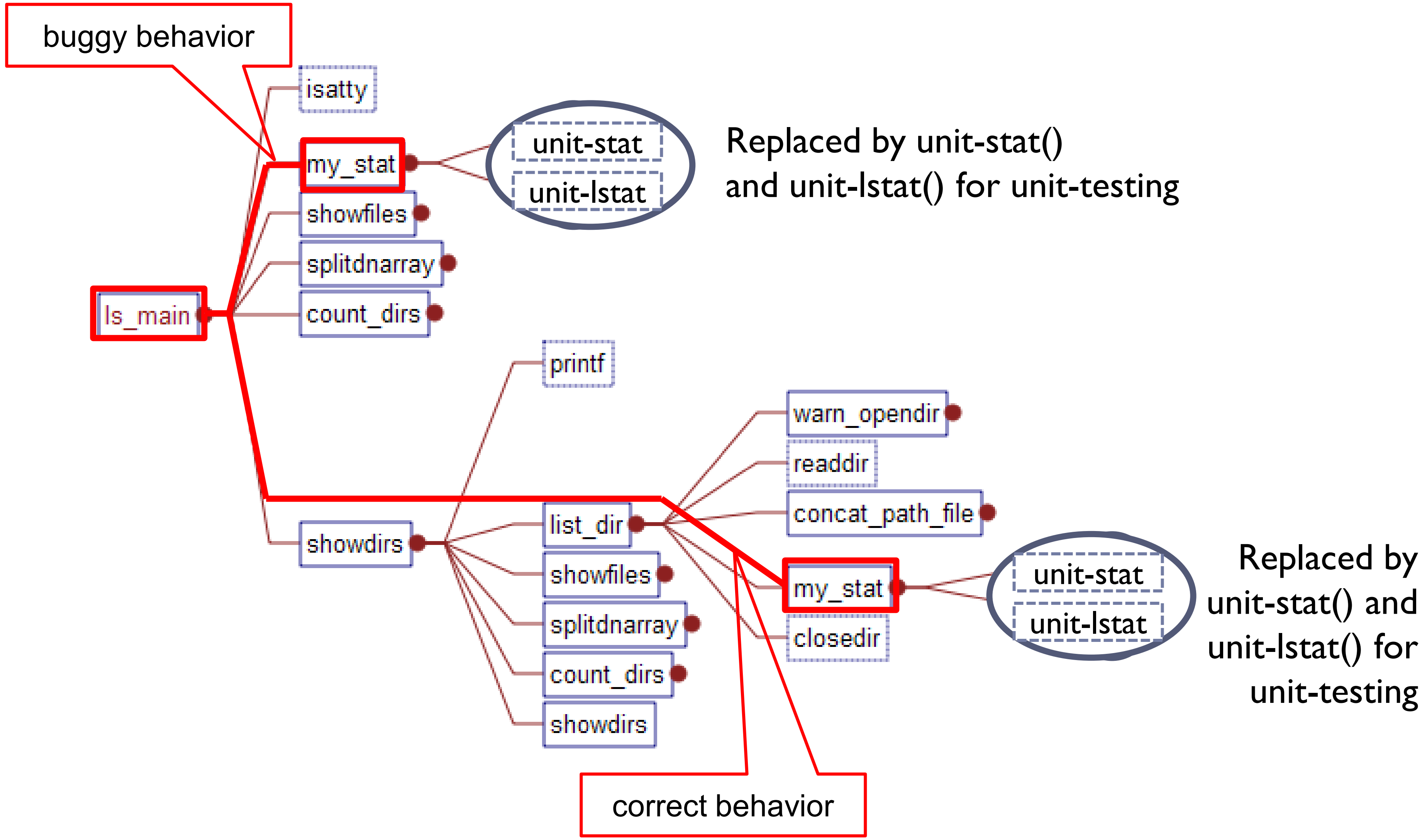
2. Output of Busybox ls (incorrect behavior):

```
$ ./busybox ls -F t.lnk
t.lnk
```

- `-F` means write a marker (/*|@=) for different type of files.
- `t.lnk` is a symbolic link, which links to file `t` in the directory ~`yang/`

We found that the bug was caused by the violation of a precondition of **`my_stat()`**

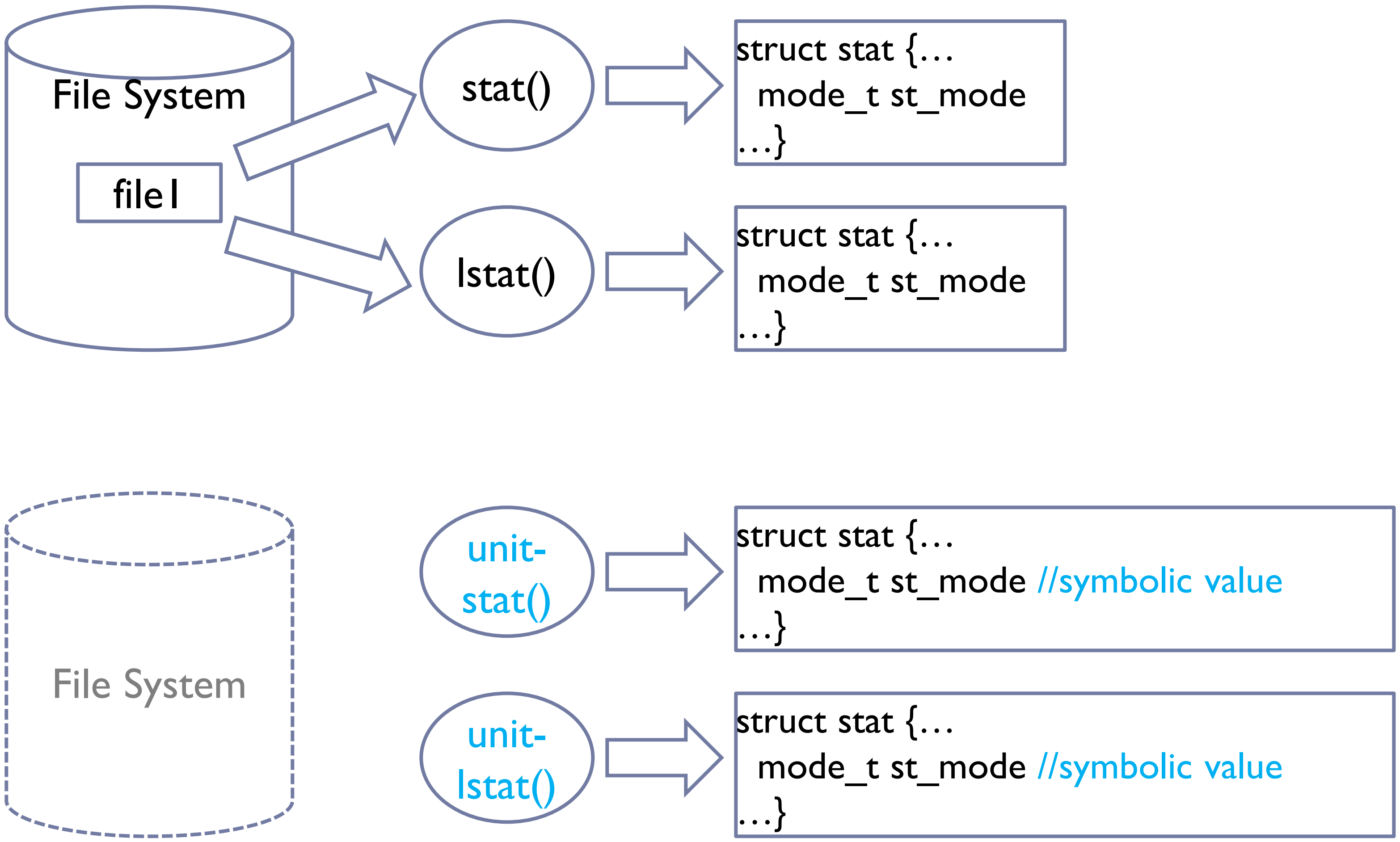
Calls Graph of Busybox ls



Replaced by unit-stat() and unit-lstat() for unit-testing

Replaced by unit-stat() and unit-lstat() for unit-testing

Stub Function – unit-stat(), unit-lstat()



Symbolic Environment Setting

- Symbolic variables:
  - Command line options
    - Replacing `unsigned int opt` with a symbolic value.
      - `opt = getopt32(argv, ……);`
  - Target file status
    - we partially simulate status of a file (`struct stat dstat`) in a file system by a symbolic value

- Symbolic stubs:
  - stat() and lstat() are replaced by unit-stat() and unit-lstat() for generating symbolic file status
    - `stat(const char *path, struct stat *buf)`
    - `lstat(const char *path, struct stat *buf)`

Testing target function: `my_stat`

```
static struct dnode *my_stat(const char
*fullname, const char *name, int force_follow)
```

- Test oracles (cont.):
  1. `len (fullname) >= len(name)`
  2. When `fullname` is a real file name, the following condition should be satisfied:

     `(cur!=NULL && cur->fullname==fullname && cur->name==name)`

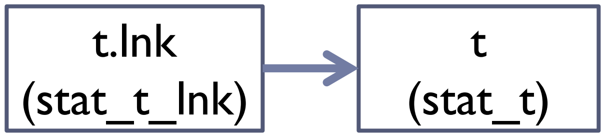
```
struct dnode {
  const char *name;
  const char *fullname;
  /* point at the next node */
  struct dnode *next;
  smallint fname_allocated;
  /* the file stat info */
  struct stat dstat;
}
```

```
struct stat {
dev_t     st_dev;     /* ID of device containing file */
ino_t     st_ino;     /* inode number */
mode_t    st_mode;    /* protection */
nlink_t   st_nlink;   /* number of hard links */
uid_t     st_uid;     /* user ID of owner */
gid_t     st_gid;     /* group ID of owner */
dev_t     st_rdev;    /* device ID (if special file) */
off_t     st_size;    /* total size, in bytes */
blksize_t st_blksize; /* blocksize for filesystem I/O */
blkcnt_t  st_blocks;  /* number of blocks allocated */
time_t    st_atime;   /* time of last access */
time_t    st_mtime;   /* time of last modification */
time_t    st_ctime;   /* time of last status change */};
```

Testing target function: `my_stat`

```
static struct dnode *my_stat(const char
  *fullname, const char *name, int force_follow)
```

- Purpose:
  1. `my_stat` gets file status by `fullname`, and store file status in **`struct dnode *cur`** which is returned by `my_stat`
  2. If a file/dir entry corresponding to `fullname` is available in the file system, `cur->stat` should stores the corresponding file info. Otherwise, NULL is turned.
- Test oracles:
  1. If any of -d, -F, or -l options is given, and -L option is not given, `follow_symlink` should be false
     - `((-d || -F || -l) && !-L) -> !follow_symlink`
     - -d: list directory entries instead of contents, and do not dereference symbolic links
     - -F: append indicator (one of */=>@|) to entries
     - -l: use a long listing format
     - -L: when showing file information for a symbolic link, show information for the file the link references rather than for the link itself



Assertions in my_stat

With -F without -L, the last parameter of my_stat() becomes true, when my_stat is called directly from ls_main()

```
static struct dnode *my_stat(const char *fullname, const char
    *name, int force_follow)
{
#ifdef ASSERTION
assert(strlen(fullname) >= strlen(name));
#endif
    struct stat dstat;
    struct dnode *cur;
    IF_SELINUX(security_context_t sid = NULL;)
#ifdef ASSERTION
/* If any of -d, -F, or -l options is given, and -L
 * option  is not given,  ls should print out  the status
 * of the symbolic  link file. I.e.,
 * ((d || F || l) && !L) -> !FOLLOW_SYM_LNK
 */
unsigned char follow_symlink =
                (all_fmt & FOLLOW_LINKS) || force_follow;
assert(!((opt_mask[2] || opt_mask[17] || opt_mask[4])
            && !opt_mask[19]) || !follow_symlink);
#endif
    if (follow_symlink) { /*get file stat of link itself*/
//……
#if !CROWN
        if (stat(fullname, &dstat))
#else
        if (unit_stat(fullname, &dstat))
#endif
        {
            bb_simple_perror_msg(fullname);
            exit_code = EXIT_FAILURE;
            return 0;
        }
    } else { /*get file stat of real file which sym_lnk linked to*/
//……
#if !CROWN
        if (lstat(fullname, &dstat))
#else
        if (unit_lstat(fullname, &dstat))
#endif
        {
            bb_simple_perror_msg(fullname);
            exit_code = EXIT_FAILURE;
            return 0;
        }
    }
    cur = xmalloc(sizeof(*cur));
    cur->fullname = fullname;
    cur->name = name;
    cur->dstat = dstat;
    IF_SELINUX(cur->sid = sid;)
    return cur;
}
```