Busybox Overview

- We test a busybox by using CREST.
  - BusyBox is a one-in-all command-line utilities providing a fairly complete programming/debugging environment
  - It combines tiny versions of ~300 UNIX utilities into a single small executable program suite.
  - Among those 300 utilities, we focused to test the following 10 utilities
    - **`grep, vi, cut, expr, od , printf, tr, cp, ls, mv.`**
    - We selected these 10 utilities, because their behavior is easy to understand so that it is clear what variables should be declared as symbolic

Experiment overview

- Experimental environments
  - HW: Core(TM)2 E8400@3GHz, 4GB memory
  - OS: fc8 32bit
  - SW: CREST 0.1.1 32bit binary, Yices 1.0.28 32bit library

- Target program: **busybox 1.17.0**

- Strategies: 4 different strategies are used in our experiment.
  - **dfs**: explore path space by Depth-First Search
  - **cfg**: explore path space by Control-Flow Directed Search
  - **random**: explore path space by Random Branch Search
  - **random_input**: testing target program by randomly generating input
- In addition, a port-polio approach is applied (i.e., merging the test cases generated by all four above strategies).

Target description -- printf

- Description: print ARGUMENT(s) according to FORMAT, where FORMAT controls the output exactly as in C printf.
- Usage: printf FORMAT [ARGUMENT]...
- Example :
  - input: ./busybox printf ‘%s is coming' 'autumn‘
  - output: autumn is coming

Target program setting -- printf

- Experiment Setting :
  - Target utilities: **busybox printf**
  - Usage: **printf FORMAT [ARGUMENT]...**
  - Symbolic variables setting:
    1. Set **FORMAT** as symbolic value.
       - Type of FORMAT is string. Restrict **5** symbolic characters as input of FORMAT.
    2. Set **ARGUMENT** as symbolic value.
       - Type of ARGUMENT is array of string. Restrict ARGUMENT to **1** length, **10** symbolic characters for each string.
    3. Replace library function by source code: ***strchr().***
  - We perform experiments in the following approach:
    1. run experiment by various strategies.

Result -- printf

Experiment setting:

Iterations: 10,000

branches in printf.c : 144

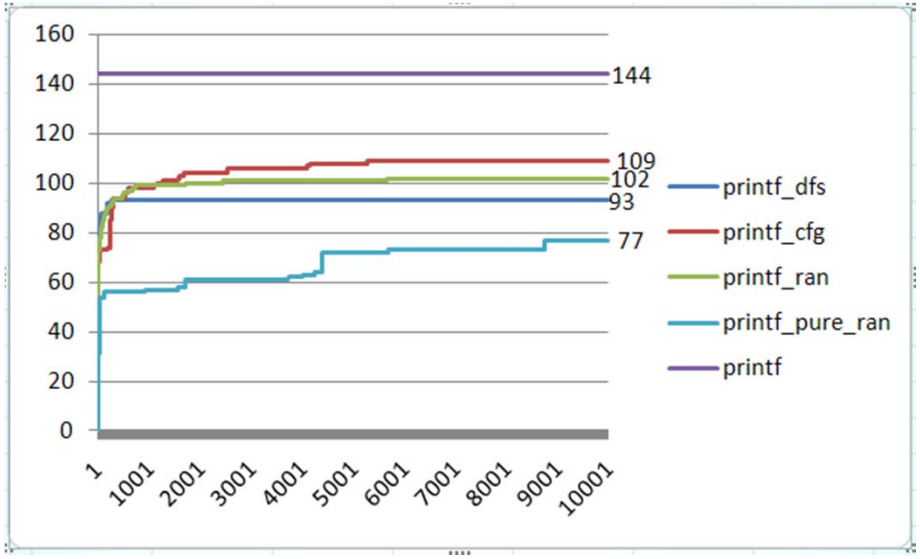
Execution command:

run_crest './busybox '%d123' 0123456789' 10000 -**dfs**

run_crest './busybox '%d123' 0123456789' 10000 -**cfg**

run_crest './busybox '%d123' 0123456789' 10000 -**random**

run_crest './busybox '%d123' 0123456789' 10000 -**random_input**



Strategy	Time cost (s)
Dfs	84
Cfg	41
Random	40
Pure_random	30

Symbolic setup in source code for printf

- Two main instruments in busybox printf.c.
  - Set 2 symbolic inputs: *FORMAT, ARGUMENT.*
  - Replace library function *strchr()* by source code.

```
1.  static void print_direc(char *format, unsigned fmt_length,
2.          int field_width, int precision,
3.          const char *argument)
4.  {
5.  //……
6.  #ifndef CREST
7.      have_width = strchr(format, '*');
8.  #else
9.      have_width = sym_strchr(format, '*');
10. #endif
11. //……
12. }
13. //……
14. int printf_main(int argc UNUSED_PARAM, char **argv)
15. {
16.     int conv_err;
17.     char *format;
18.     char **argv2;
19. //……
20.     format = argv[1];
21.     argv2 = argv + 2;
22.     int i;
23.     int argcc=strlen(format);
24. #ifdef CREST
25.     for( i=0 ; i<argcc ; i++){
26.         CREST_char(format[i]);
27.     }
28.     for(i= 0 ; i<10 ; i++){
29.         CREST_char(argv2[0][i]);
30.     }
31. #endif
32. //……
33. }

34. static char *sym_strchr(const char *str, char ch){

35.     while (*str && *str != ch)
36.         str++;

37.     if (*str == ch)
38.         return str;

39.     return(NULL);

40. }
```

Target description -- grep

- Description: Search for PATTERN in FILEs (or stdin).
- Usage: grep [OPTIONS] PATTERN [FILE]
  - OPTIONS includes

  [-1nqvscFiHhf:Lorm:wA:B:C:EaI] (option followed by ":" means one argument is required.)
- Example :
  - "test_grep.dat" contains

```
define
enifed
what is defined?
def ine
```

  - input: `busybox grep "define" test_grep.dat`
  - output:

```
define
what is defined?
```

Options:
-H Add 'filename:' prefix
-h Do not add 'filename:' prefix
-n Add 'line_no:' prefix
-l Show only names of files that match
-L Show only names of files that don't match
-c Show only count of matching lines
-o Show only the matching part of line
-q Quiet. Return 0 if PATTERN is found, 1 otherwise
-v Select non-matching lines
-s Suppress open and read errors
-r Recurse
-i Ignore case
-w Match whole words only
-F PATTERN is a literal (not regexp)
-E PATTERN is an extended regexp
-m N Match up to N times per file
-A N Print N lines of trailing context
-B N Print N lines of leading context
-C N Same as '-A N -B N'
-f FILE Read pattern from file

Instrumentation for Concolic Testing

- Symbolic variable declaration
  - First, identify input variables
  - Second, declare these variables as symbolic variable by inserting CREST_<type>(var_name);
    - If necessary, additional constraints should be given to restrict symbolic variables to have valid ranges of values
      - if (!constraints on var_name) exit(0); shou
- Transform complex functions into simpler ones manually, if necessary
  - For example, bitwise operators (i.e. &, |, ~) cannot be handled by CREST
    - Bitwise operators are replaced by manually made functions containing loops to handle each bit of operands

Symbolic Variable Declaration for grep

- PATTERN was not declared as symbolic variables, since grep.c handles PATTERN using external binary libraries
  - CREST would not generate new test cases for symbolic PATTERN
  - We used a pattern “define”
- We use a concrete file “test_grep.dat” as a FILE parameter
- Set options as symbolic input (i.e. an array of symbolic character)
  - **23** different options can be given.
    - Specified options are represented by ***option_mask32,*** an ***uint32_t*** value, of which each bit field indicates one option is ON/OFF.
    - Function ***getopt32(char **argv, const char *applet_opts, …)*** is used to generate a bit array indicating specified options from command line input.
    - We added a ***bit mask*** is defined to replace ***option_mask32,*** whose type is array of integer. We replace bitwise operators by normal linear integer expressions through additional loops
- Set 4 parameters to options as symbolic variables
  - *Copt, max_matches, lines_before, lines_after.*
  - Option argument “***fopt***” is ignored, since it is hard to set file name as symbolic value. “***fopt***”, the parameter of option “-f” (read pattern form an exist file).

Instrumentation in grep.c

```
1. #define BITSIZE 23
2. int bitmask[23];
3. int bitopt[23];
4.
5. int grep_main (int argc UNUSED_PARAM, char
   **argv)
6. {
7.       getopt32(argv, OPTSTR_GREP,
8.         &pattern_head, &fopt,
9.         &max_matches, &lines_after,
10.         &lines_before,&Copt);
11.      #ifdef CREST
12.      CREST_int(max_matches);
13.      CREST_int(lines_after);
14.      CREST_int(lines_before);
15.      CREST_int(Copt);
16.      int i;
17.      for(i=BITSIZE-1 ; i>=0 ; i--){
18.            CREST_int(bitmask[i]);
19.      }
20.      #endif
21.      //……
22.      #ifndef CREST
23.      if (option_mask32 & OPT_m)
24.      #else
25.      if (bit_and(bitmask, itobs(OPT_m, bitopt)))
26.      #endif
27.      {
28.          //……
29.      }
30.      //……
31.}
```

2 Functions Added to Handle Bitwise Operators

- Two functions are added to replace “&”. They are bit_and(), itobs().
- int* itobs(unsigned long int n, int *bs) translates bit sequence value of an integer into int sequence value, and return this int sequence value.
- int bit_and(int *bita, int *bitb) compares bita and bitb, if both two cells in the same position of bita and bitb are not 0, then return 1. Otherwise, return 0.

```
1.  // bit and each cell of two arrays
2.  //bita is symbolic array, and bitb is
    constant
3.  // return 0 if the result of bit_and is 0,
4.  // return 1 otherwise
5.  int bit_and(int *bita, int *bitb)
6.  {
7.          int i;
8.          int flag=0;
9.          for(i=0 ; i<BITSIZE ; i++)
10.         {
11.                 if(bita[i]!=0 && bitb[i]==1){
12.                         flag=1;
13.                 }
14.         }
15.     return flag;
16. }
17.
18. //convert an unsigned long int variable
19. // into an array, each cell save value of
20. //one bit the variable
21. int* itobs(unsigned long int n, int *bs)
22. {
23.         int i;
24.         int *temp=bs;
25.         for(i = BITSIZE-1 ; i>=0 ; i--,
    n=n/2){
26.                 bs[i]=n%2;
27.         }
28.         return bs;
29. }
30.
31. #define BITSIZE 4
32. int bitmask[BITSIZE];
33. int bitopt[BITSIZE];
34.
```

Result of grep

Experiment 1:

Iterations: 10, 000

branches in grep.c : 178

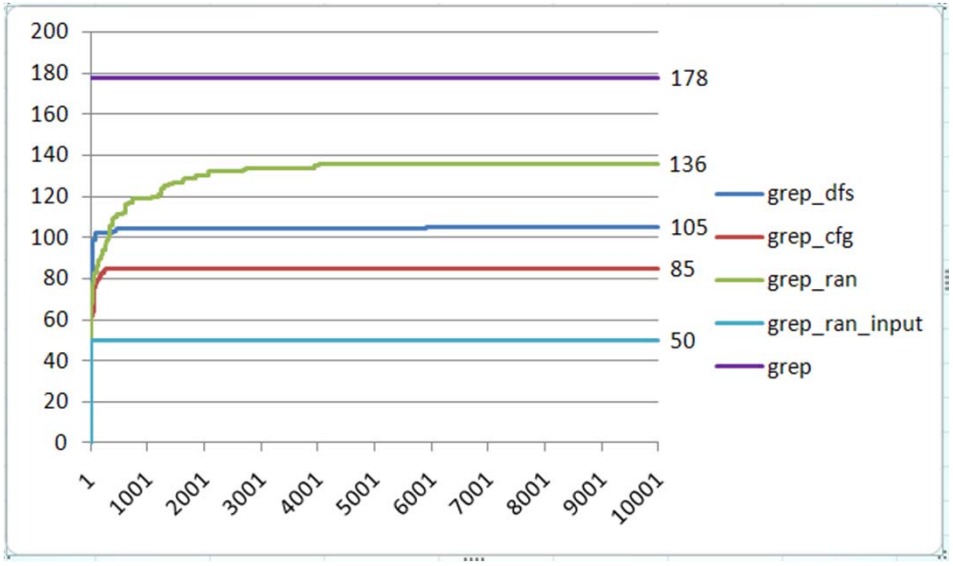
Execution Command:

run_crest './busybox grep "define" test_grep.dat' 10000 -**dfs**

run_crest './busybox grep "define" test_grep.dat' 10000 -**cfg**

run_crest './busybox grep "define" test_grep.dat' 10000 -**random**

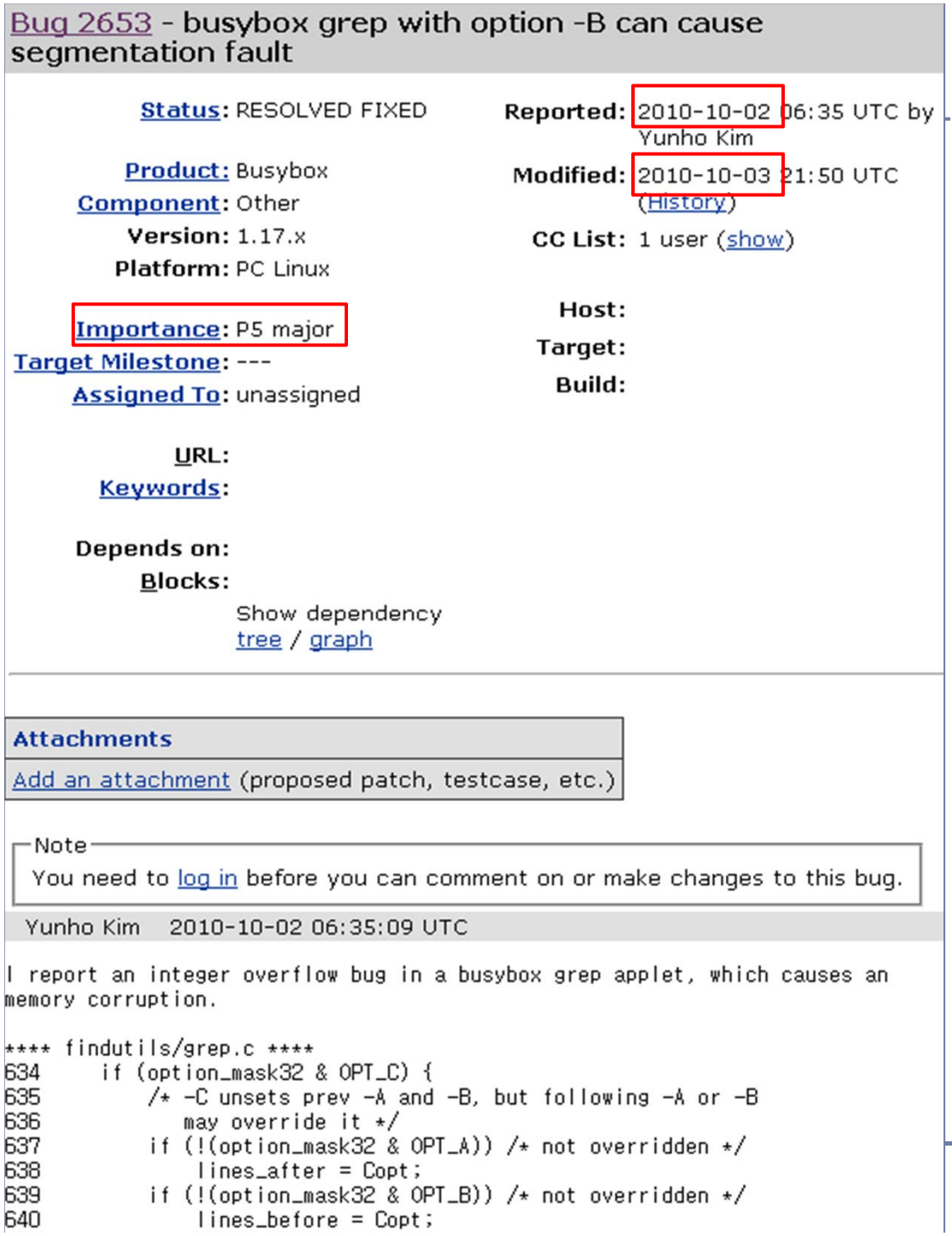
run_crest './busybox grep "define" test_grep.dat' 10000 -**random_input**



Strategy	Time cost (s)
Dfs	2758
Cfg	56
Random	85
Pure_random	45

Test Oracles

- In the busybox testing, we do not use any explicit test oracles
  - Test oracle is an orthogonal issue to test case generation
  - However, still violation of runtime conformance (i.e., no segmentation fault, no divide-by-zero, etc) can be checked
- Segmentation fault due to integer overflow detected at grep 2.0
  - This bug was detected by test cases generated using DFS
  - The bug causes segmentation fault when
    - -B 1073741824 (i.e. 2^32/4)
    - PATTERN should match line(s) after the 1st line
    - Text file should contain at least two lines
  - Bug scenario
    - Grep tries to dynamically allocate memory for buffering matched lines (-B option).
    - But due to integer overflow (# of line to buffer * sizeof(pointer)), memory is allocated in much less amount
    - Finally grep finally accesses illegal memory area



Bug 2653 - busybox grep with option -B can cause segmentation fault

Status: RESOLVED FIXED
Reported: 2010-10-02 06:35 UTC by Yunho Kim
Product: Busybox
Modified: 2010-10-03 21:50 UTC (History)
Component: Other
Version: 1.17.x
CC List: 1 user (show)
Platform: PC Linux
Importance: P5 major
Host:
Target Milestone: ---
Target:
Assigned To: unassigned
Build:
URL:
Keywords:
Depends on:
Blocks:
Show dependency tree / graph

Attachments
Add an attachment (proposed patch, testcase, etc.)

Note
You need to log in before you can comment on or make changes to this bug.

Yunho Kim 2010-10-02 06:35:09 UTC

```
I report an integer overflow bug in a busybox grep applet, which causes an
memory corruption.

**** findutils/grep.c ****
634     if (option_mask32 & OPT_C) {
635         /* -C unsets prev -A and -B, but following -A or -B
636            may override it */
637         if (!(option_mask32 & OPT_A)) /* not overridden */
638             lines_after = Copt;
639         if (!(option_mask32 & OPT_B)) /* not overridden */
640             lines_before = Copt;
```

- Bug patch was immediately made in 1 day, since this bug is critical one
  - Importance: P5 major
    - major loss of function
  - Busybox 1.18.x will have fix for this bug

Target description -- vi

- Description: Edit FILE

- Usage: vi [OPTIONS] [FILE] …
- Options:
  - -c Initial command to run ($EXINIT also available)
  - -R Read-only
  - -H Short help regarding available features
- Example :
  - input: cat read_vi.dat

    test for initial command
  - input: cat test_vi.dat

    this is the test for vi

    @#$%&*vi?
  - input: ./busybox vi -c ":read read_vi.dat" test_vi.dat
  - output:

    this is the test for vi
    test for initial command

    @#$%&*vi?

Symbolic Variable Declaration for vi

- We declared a key stroke by a user as a symbolic input character
  - Restrict user key input to **50** symbolic characters.
    - **We modified vi source code so that vi exits after testing 50th key stroke.**
- Set initial command as symbolic input (initial command is only used when option “-c” is specified).
  - Type of initial command is a string (i.e., an array of 17 characters)
- Replace ***4*** library functions with source code: ***strncmp(), strchr(), strcpy(), memchr()***.
- We used a concrete file “test_vi.dat”

Test Case Extraction

- Test case extraction
  - A test case consists of concrete values for symbolic variables declared by CREST_<type>(var).
  - We stored concrete values of symbolic variables in a test case file
  - One TC file is 67 bytes including 17 bytes of an initial command and 50 bytes of key strokes.
- Two modifications for extracting test case of vi.c
  - In coreuitls/vi.c, store all characters generated by CREST to a test case file -- tc.
  - In crest/src/run_crest/concolic_search.cc, provides an iteration number to make test case file has unique name (e.g. a test case generated at 1000'th iteration is named as ***tc_1000***)

```
1.  //……
2.  static FILE *finput;
3.  int vi_main(int argc, char **argv){
4.    //……
5.    finput=fopen("tc", "w");
6.    //……
7.  }
8.  //……
9.  static void edit_file(char *fn)
10. {
11.   //……
12.   #ifdef CREST
13.   for(i=0 ; i<strlen(initial_cmds[0]);i++)
14.   {
15.       CREST_char(initial_cmds[0][i]);
16.       putc(initial_cmds[0][i], finput);
17.   }
18.   #endif
19.   //……
20. }
21. //……
22. static int readit(void){
23.   //……
24.   #ifndef CREST
25.   c = read_key(STDIN_FILENO,
      readbuffer, /*timeout off:*/ -2);
26.   #else
27.   if(count<50){
28.           char ch;
29.           CREST_char(ch);
30.           putc(ch, finput);
31.           c=(int)ch;
32.           count++;
33.       }else {
34.           fclose(finput);
35.           exit(0);
36.       }
37.   #endif
38.   //……
39. }
```

vi.c

```
1.  void Search::LaunchProgram(const
    vector<value_t>& inputs) {
2.    WriteInputToFileOrDie("input", inputs);

3.    char name[10];
4.    char cmd[100];
5.    system("mkdir -p results/inputs");
6.    sprintf(name, "tc_%d", num_iters_);
7.    sprintf(cmd, "cp tc   results/inputs/%s",
    name);
8.    system(cmd);

9.    system(program_.c_str());
10. }
```

concolic_search.cc

4 Functions Added

```
1.  static int sym_strncmp (const char *first,const char *last,int
    count)
2.  {
3.     if (!count)
4.        return(0);
5.
6.     while (--count && *first && *first == *last){
7.          first++;
8.          last++;
9.      }
10.   return( *(unsigned char *)first - *(unsigned char *)last );
11.}
12.
13.static char *sym_strchr(const char *str, char ch){
14.
15.   while (*str && *str != ch)
16.        str++;
17.
18.   if (*str == ch)
19.        return str;
20.
21.   return(NULL);
22.
23.}
24.
```

```
25.static char *sym_strcpy(char *to, const char *from)
26.{
27.        char *save = to;
28.        for (; (*to = *from) != '₩0'; ++from, ++to);
29.        return(save);
30.}
31.
32.void *sym_memchr(const void* src, int c, size_t count)
33.{
34.      assert(src!=NULL);
35.      char *tempsrc=(char*)src;
36.      while(count&&*tempsrc!=(char)c)
37.      {
38.          count--;
39.          tempsrc++;
40.       }
41.       if(count!=0)
42.            return tempsrc;
43.       else
44.            return NULL;
45.}
```

Result of vi

Experiment 1:

Iterations: 10,000

Branches in vi.c : 1498

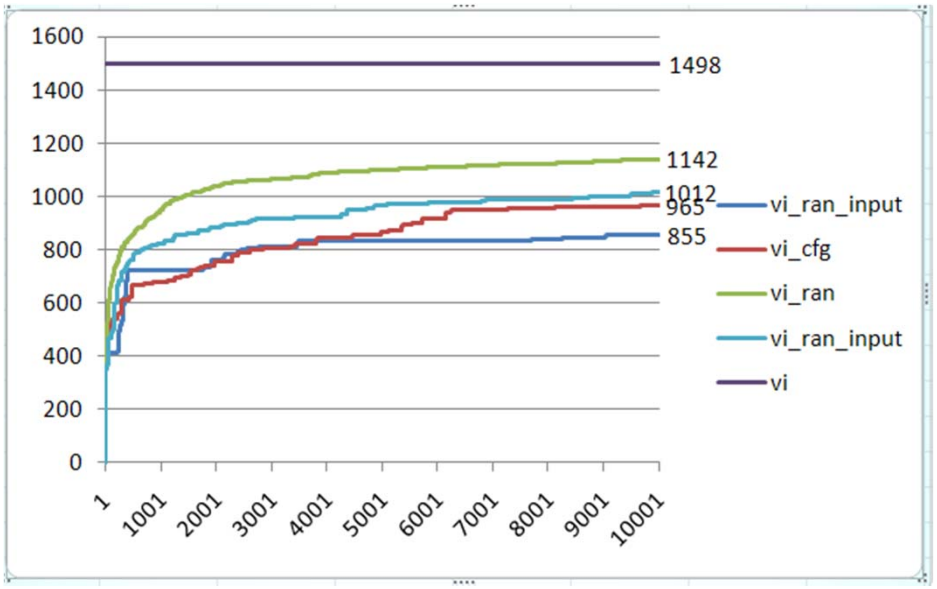
Execution Command:

run_crest './busybox -c ":read read_vi.dat" test_vi.dat' 10000 -**dfs**

run_crest './busybox -c ":read read_vi.dat" test_vi.dat' 10000 -**cfg**

run_crest './busybox -c ":read read_vi.dat" test_vi.dat' 10000 -**random**

run_crest './busybox -c ":read read_vi.dat" test_vi.dat' 10000 -**random_input**



Strategy	Time cost (s)
Dfs	1495
Cfg	1306
Random	723
Pure_random	463

Example of Generating/Feeding TCs Together

```
1. static FILE *finput;      //create a file stream
   to read from TC
2.
3. //create a buffer to store TC data
4. #if ENABLE_APPEND_CHAR
5. static struct tcbuf{
6.     unsigned char bitmask[BITSIZE];
7.     unsigned char modemask[MODESIZE];
8. }tcbuf;
9. #endif

10.int ls_main(int argc UNUSED_PARAM, char **argv)
11.{
12.    int i,j;
13.#if TC_FED //read TC data into buffer
14.    finput=fopen("results/ts/tc_1", "r");
15.    if(finput==NULL){
16.        fprintf(stderr, "Can not open TC file,
   please check TC path!\n");
17.        exit(0);
18.    }
19.    struct tcbuf tc1;
20.    fread(tc1.bitmask, sizeof(unsigned char),
   BITSIZE, finput);
21.    fread(tc1.modemask, sizeof(unsigned char),
   MODESIZE, finput);
22.    fclose(finput);
23.#endif
24.//……
25.#if CREST_TC_GEN
26.    for(i=0 ; i<BITSIZE ; i++){
27.        CREST_unsigned_char(bitmask[i]);
28.    }
29.    fwrite(bitmask, sizeof(unsigned char),
   BITSIZE, foutput);
30.#endif
31.#if TC_FED
32.    memcpy(bitmask, tc1.bitmask, BITSIZE);
33.    option_mask32=bstoi(bitmask, BITSIZE);
34.#endif

35.#if ENABLE_APPEND_CHAR
36.#if CREST_TC_GEN
37.//source code for tc generation
38.#endif
39.#if TC_FED  //execute append_char with buffer data
40.    unsigned int mode=bstoi(tc1.modemask,
   MODESIZE);
41.#endif
42.    char a = append_char(mode);
43.#endif
44.}
```

Unit-testing Busybox ls

- We performed unit-testing Busybox ls by using CREST
  - 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
  - Automated oracle approach
- However, it still required human expertise on Busybox ls code to define concrete assert statements from given high-level requirements

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.

Four 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

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:

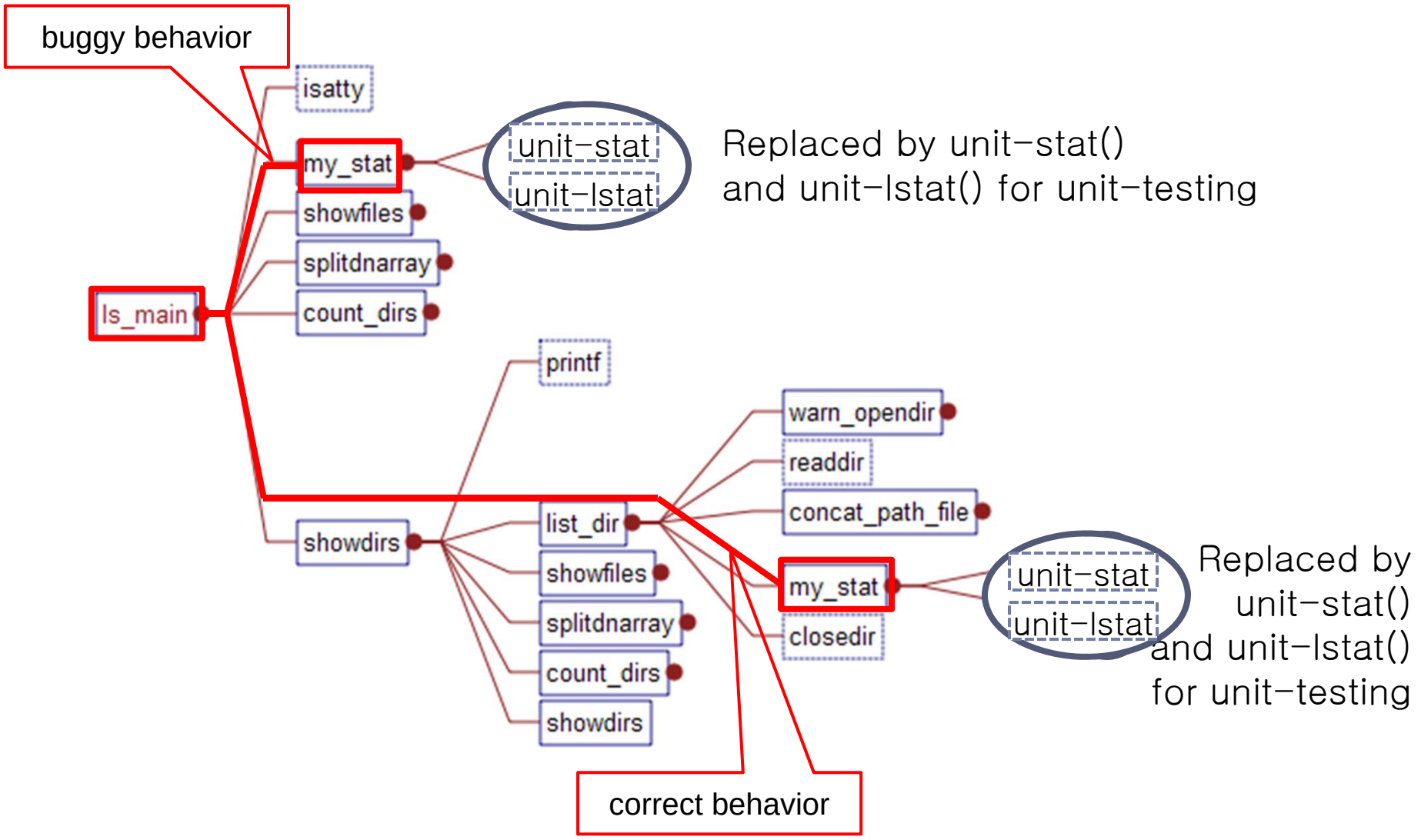
```
$ ls -F t.lnk
t.lnk@
```

  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 due to the violation of a precondition of **`my_stat()`**

Calls Graph of Busybox ls



ls_main

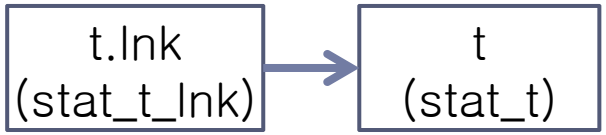
```
int ls_main(int argc UNUSED_PARAM,
            char **argv)
```

- Purpose:
  1. `ls_main` obtains specified options and file/dir names from command line input.
  2. Calculate runtime features by specified options.
     1) `unsigned int` **`opt`** is defined to represent all options given in the command line arguments
        - `opt = getopt32(argv, ls_options, &tabstops, &terminal_width)`
     2) `unsigned int` **`all_fmt`** is defined to represent all runtime features based on the given command line options and compile time option **`opt_flags`**`[]`
        - `all_fmt=`fun (`opt, opt_flags`)
  3. Print file/dir entries of a file system whose names and corresponding options are given in the command line input by calling sub-functions

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.
- Pre condition:
  1. `len (fullname) >= len(name)`
  2. 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: llist 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



my_stat

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

- Post condition:
  1. 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 */};
```

Assertions in my_stat

```
1.  static struct dnode *my_stat(const char *fullname,
    const char *name, int force_follow)
2.  {
3.  #ifdef ASSERTION
4.  assert(strlen(fullname) >= strlen(name));
5.  #endif
6.      struct stat dstat;
7.      struct dnode *cur;
8.      IF_SELINUX(security_context_t sid = NULL;)
9.  #ifdef ASSERTION
10./* If any of -d, -F, or -l options is given, and -L
11. * option  is not given,  ls should print out  the status
12. * of the symbolic  link file. I.e.,
13. * ((d || F || l) && !L) -> !FOLLOW_SYM_LNK
14. */
15.unsigned char follow_symlink =
16.             (all_fmt & FOLLOW_LINKS) || force_follow;
17.assert(!((opt_mask[2] || opt_mask[17] || opt_mask[4])
18.          && !opt_mask[19]) || !follow_symlink);
19.#endif
20.     if (follow_symlink) { /*get file stat of link itself*/
21.//……
22.#if !CREST
23.         if (stat(fullname, &dstat))
24.#else
25.         if (unit_stat(fullname, &dstat))
26.#endif
27.         {
28.             bb_simple_perror_msg(fullname);
29.             exit_code = EXIT_FAILURE;
30.             return 0;
31.         }
32.     } else { /*get file stat of real file which sym_lnk
    linked to*/
33.//……
34.#if !CREST
35.         if (lstat(fullname, &dstat))
36.#else
37.         if (unit_lstat(fullname, &dstat))
38.#endif
39.         {
40.             bb_simple_perror_msg(fullname);
41.             exit_code = EXIT_FAILURE;
42.             return 0;
43.         }
44.     }
44.     cur = xmalloc(sizeof(*cur));
45.     cur->fullname = fullname;
46.     cur->name = name;
47.     cur->dstat = dstat;
48.     IF_SELINUX(cur->sid = sid;)
49.#ifdef ASSERTION
50.assert(strcmp(fullname, cur->fullname)==0);
51.assert(strcmp(name, cur->name)==0);
52.#endif
53.     return cur;
54.}
```

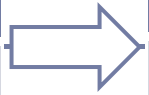
Symbolic Environment Setting

- Symbolic variables:
  - Command line options
  - Target file status
    - we partially simulate status of a file (`struct stat dstat`) in a file system by a symbolic value
- Test 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)`
- Macro re-definition:
  - SYM_S_ISLNK() is replaced by S_ISLNK()
    - Note that CREST cannot handle bit operator directly.

Symbolic Variables

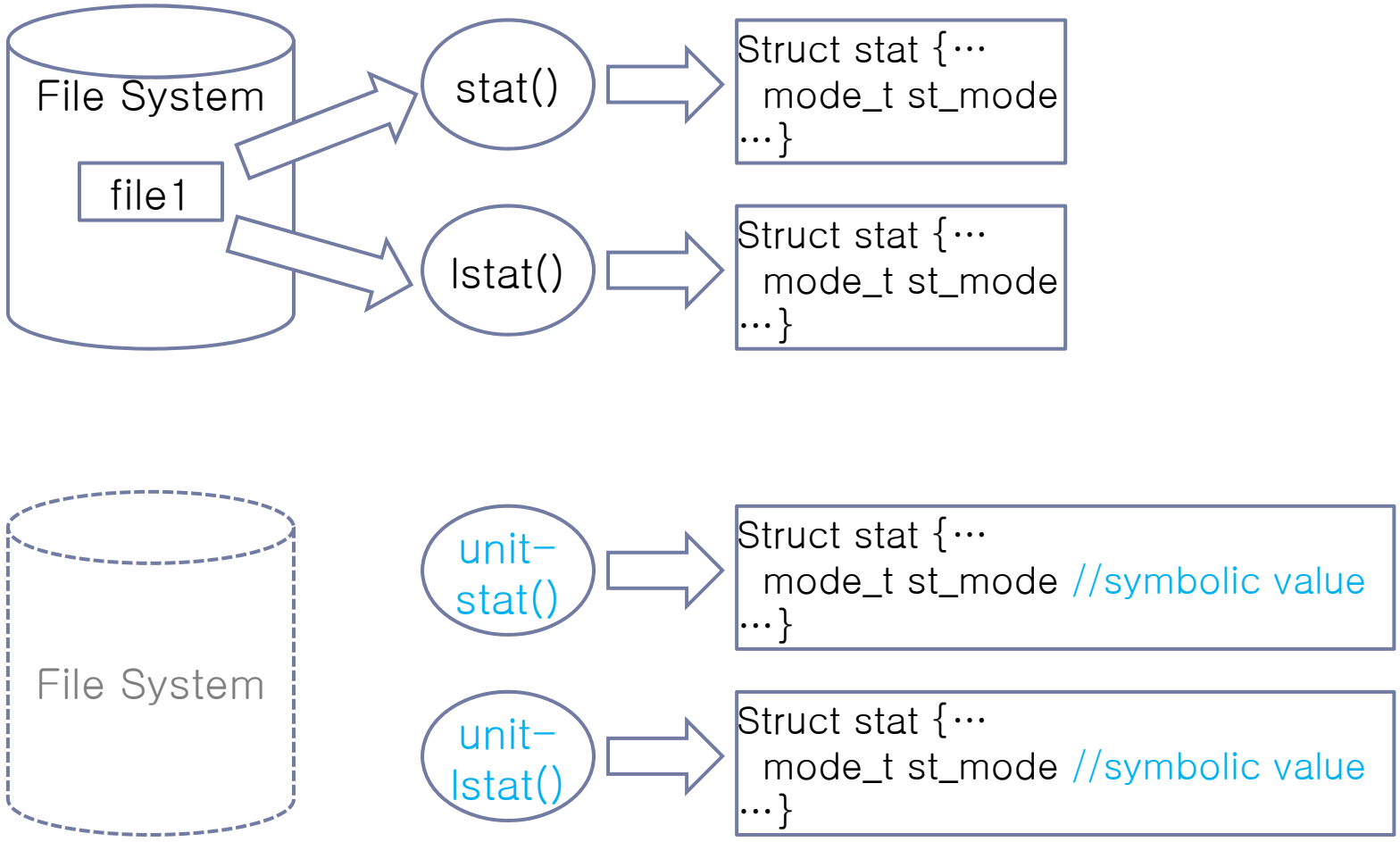
- Symbolic options
  - Replacing `unsigned int opt` with a symbolic value.
    - `opt = getopt32(argv, ……);`
    - `unsigned char opt_mask[22]` is defined to replace `opt`
      - each element in `opt_mask[]` specifies a symbolic value for each bit of `opt`
- Symbolic file status
  - Replacing `mode_t dstat.st_mode` with a symbolic value
    - `st_mode` stores information on a file type, file permission, etc. in each bit.
      - `sizeof(st_mode)=sizeof(unsigned int)`
    - `unsigned char modemask[32]` is defined to replace `st_mode`
      - each element in `modemask[]` specifies a symbolic value for each bit of `st_mode`

Bit 0	But 1	Bit 2	…	Bit 21
1	0	1	…	0



array[0]	array[1]	Array[2]	…	array[21]
1	0	1	…	0

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



Test Driver for Symbolic Command Line Options

```
50.int ls_main(int argc UNUSED_PARAM, char **argv)
51.{
52.//……
53./* process options */
54.    IF_FEATURE_LS_COLOR(applet_long_options
55.     = ls_longopts;)
56.#if ENABLE_FEATURE_AUTOWIDTH
57.    opt_complementary="T+:w+"; /* -T N, -w N */
58.    opt = getopt32(argv, ls_options, &tabstops,
59.          & terminal_width, IF_FEATURE_LS_COLOR
60.          (&color_opt));
61.#else
62.    opt = getopt32(argv, ls_options
63.           IF_FEATURE_LS_COLOR(&color_opt));
64.#endif
65.
66.#ifdef CREST
67.for(i=0 ; i<OPTSIZE ; i++){
68.    CREST_unsigned_char(opt_mask[i]);
69.}
70.opt=bstoi(opt_mask, OPTSIZE);
71.option_mask32=opt;
72.#endif

73.//START of calculating all_fmt value
74.    for (i = 0; opt_flags[i] != (1U<<31); i++) {
75.#ifdef CREST
76.        if(opt_mask[i]!=0)
77.#else
78.        if (opt & (1 << i))
79.#endif
80.        {
81.            unsigned flags = opt_flags[i];
82.            //refresh value when trigger is on
83.            if (flags & LIST_MASK_TRIGGER) //0
84.                all_fmt &= ~LIST_MASK;
85.//……
86.            all_fmt |= flags;
87.        }
88.    }
89.//processing all_fmt for some special cases
90.    if (argv[1])
91.        all_fmt |= DISP_DIRNAME; /* 2 or more
    items? label directories */
92.//END of calculating all_fmt value
93.//calling internal functions
94.}
```

Test Driver for Symbolic File Status

```
1.  #define OPTSIZE 22
2.  #define MODESIZE 32
3.  unsigned char opt_mask[OPTSIZE];
4.  unsigned char modemask[2][MODESIZE];
5.  static char file_no=0;
6.
7.  #define SYM_S_ISLNK(mode) ((mode)[12]==0 &&
    (mode)[13]==1 && (mode)[14]==0 && (mode)[15]==1)

8.  //simulating file status by symbolic value
9.  static char gen_sym_file_stat(struct stat *buf){
10.     int i;
11.     for(i=0 ; i<MODESIZE ; i++){ //file type
12.         CREST_unsigned_char(modemask[file_no][i]);
13.     }
14.     return 1;
15.}
16.
17.static int unit_stat(const char *path, struct stat *buf){
18.     unsigned char local_mode[32];
19.     char ret;
20.     CREST_char(ret);
21.     if(ret==(char)0){
22.         if(gen_sym_file_stat(buf)){
23.             memcpy(local_mode, modemask[file_no], 32);
24.             if(SYM_S_ISLNK(local_mode)){ //link
25.                 local_mode[13]=0; //change to reg file
26.             }
27.             buf->st_mode = bstoi(local_mode, MODESIZE);
28.         }
29.         file_no++;
30.         return 0;
31.     }else
32.         return -1;
33.}
34.
35.static int unit_lstat(const char *path, struct stat *buf){
36.     unsigned char local_mode[32];
37.     char ret;
38.     CREST_char(ret);
39.     if(ret==(char)0){
40.         if(gen_sym_file_stat(buf)){
41.             memcpy(local_mode, modemask[file_no],
    MODESIZE);
42.             buf->st_mode = bstoi(local_mode, MODESIZE);
43.         }
44.         file_no++;
45.         return 0;
46.     }else
47.         return -1;
48.}
```