EE516 : Homework 3

Gaurav Kalra

(Student ID: 2016 45 93)

gvkalra@kaist.ac.kr

Problem 1: Explain the following functions

lseek(), unlink(), fchmod(), fchown(), link(), symlink(), readlink(), fstat(), readdir(), getcwd()

[1] lseek()

Reposition read/write file offset

[2] unlink()

delete a name and possibly the file it refers to

[3] fchmod()

change permissions of a file

[4] fchown()

change ownership of a file

[5] link()

make a new name for a file (hard link)

[6] symlink()

make a new name for a file (soft link)

[7] readlink()

read value of a symbolic link

[8] fstat()

get file status

[9] readdir()

read a directory

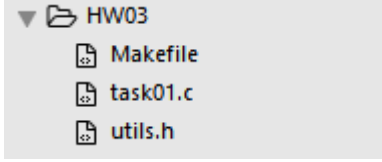
[10] getcwd()

get current working directory

References:

[1] http://man7.org/linux/man-pages/man2/lseek.2.html
[2] http://man7.org/linux/man-pages/man2/unlink.2.html
[3] http://man7.org/linux/man-pages/man2/fchmodat.2.html
[4] http://man7.org/linux/man-pages/man2/lchown.2.html
[5] http://man7.org/linux/man-pages/man2/link.2.html
[6] http://man7.org/linux/man-pages/man2/symlink.2.html
[7] http://man7.org/linux/man-pages/man2/readlink.2.html
[8] http://man7.org/linux/man-pages/man3/fstat.3p.html
[9] http://man7.org/linux/man-pages/man3/readdir.3.html
[10] http://man7.org/linux/man-pages/man2/getcwd.2.html

Problem 2: Make your own program using each red coloured functions




```c
cwd_path = getcwd(NULL, 0);
if (cwd_path == NULL) {
    err("getcwd() failed: [%s]", strerror(errno));
    goto error;
}

info("[10] getcwd() : %s", cwd_path);

info("\n##### Creating %s #####\n", FILENAME);
```

```c
static void
list_files(const char *dir_path)
{
    DIR *dir = NULL;
    struct dirent *r_dirent = NULL;

    /* open directory */
    dir = opendir(dir_path);
    if (dir == NULL) {
        err("opendir() failed: [%s]", strerror(errno));
        return;
    }

    /* list names & inode numbers */
    info("[9] readdir() => d_ino (d_name)");
    errno = 0;
    while ((r_dirent = readdir(dir)) != NULL) {
        info("\t %ld (%s)", r_dirent->d_ino, r_dirent->d_name);
    };
    if (errno != 0)
        err("readdir() failed: [%s]", strerror(errno));

    /* close directory */
    if (closedir(dir) != 0)
        err("closedir() failed: [%s]", strerror(errno));
}
```

```
root@gvkalra-desktop:/home/gvkalra/Desktop/EE516/HW03# ./task01
[10] getcwd() : /home/gvkalra/Desktop/EE516/HW03

##### Creating task01.dat #####

[9] readdir() => d_ino (d_name)
        3148596 (task01.dat)
        3146255 (task01)
        3146198 (.)
        3146259 (utils.h)
        3146257 (Makefile)
        3148595 (task01.c)
        2760540 (..)
[7] readlink() : success => /home/gvkalra/Desktop/EE516/HW03/task01.dat
[8] fstat() => /home/gvkalra/Desktop/EE516/HW03/task01.dat
        st_dev : 2066
        st_ino : 3148596
        st_mode : 33216
        st_nlink : 1
        st_uid : 0
        st_gid : 0
        st_rdev : 0
        st_size : 0
        st_blksize : 4096
        st_blocks : 0
        st_atime : 1478257091
        st_mtime : 1478257091
        st_ctime : 1478257091
```

```c
/* read file stat */
ret = fstat(fd, &sb);
if (ret != 0) {
    err("fstat() failed: [%s]", strerror(errno));
    return;
}

/* construct fd path from /proc */
ret = snprintf(proc_fd, sizeof(proc_fd), "/proc/self/fd/%u", fd);
if (ret >= sizeof(proc_fd)) {
    err("proc_fd buffer overflow");
    return;
}

/* read symlink */
bytes_read = readlink(proc_fd, link_value, sizeof(link_value));
if (bytes_read == -1) {
    err("readlink() failed: [%s]", strerror(errno));
    /* print UNKNOWN */
    (void)snprintf(link_value, sizeof(link_value), "UNKNOWN");
} else {
    /* NULL terminate */
    link_value[bytes_read] = '\0';
    info("[7] readlink() : success => %s", link_value);
}

info("[8] fstat() => %s", link_value);
info("\tst_dev : %lu", sb.st_dev);
info("\tst_ino : %lu", sb.st_ino);
info("\tst_mode : %d", sb.st_mode);
info("\tst_nlink : %lu", sb.st_nlink);
info("\tst_uid : %d", sb.st_uid);
info("\tst_gid : %d", sb.st_gid);
info("\tst_rdev : %lu", sb.st_rdev);
info("\tst_size : %ld", sb.st_size);
info("\tst_blksize : %ld", sb.st_blksize);
info("\tst_blocks : %ld", sb.st_blocks);
info("\tst_atime : %ld", sb.st_atime);
info("\tst_mtime : %ld", sb.st_mtime);
info("\tst_ctime : %ld", sb.st_ctime);
```

task01.dat is newly created.

Notice the following:

1. **st_mode** = 33216
2. **inode** = 3148596

```c
/* owner can read, write only */
if (fchmod(fd, S_IRUSR | S_IWUSR) != 0) {
    err("fchmod() failed: [%s]", strerror(errno));
    return;
} else {
    info("[3] fchmod() : success => S_IRUSR | S_IWUSR");
}
print_stat(fd);

/* restore permission */
if (fchmod(fd, S_IRWXU) != 0) {
    err("fchmod() failed: [%s]", strerror(errno));
    return;
} else {
    info("[3] fchmod() : success => restored");
}
print_stat(fd);
```

```
##### Permission Play #####

[3] fchmod() : success => S_IRUSR | S_IWUSR
[7] readlink() : success => /home/gvkalra/Desktop/EE516/HW03/task01.dat
[8] fstat() => /home/gvkalra/Desktop/EE516/HW03/task01.dat
        st_dev : 2066
        st_ino : 3148596
        st_mode : 33152
        st_nlink : 1
        st_uid : 0
        st_gid : 0
        st_rdev : 0
        st_size : 0
        st_blksize : 4096
        st_blocks : 0
        st_atime : 1478257091
        st_mtime : 1478257091
        st_ctime : 1478257091
[3] fchmod() : success => restored
[7] readlink() : success => /home/gvkalra/Desktop/EE516/HW03/task01.dat
[8] fstat() => /home/gvkalra/Desktop/EE516/HW03/task01.dat
        st_dev : 2066
        st_ino : 3148596
        st_mode : 33216
        st_nlink : 1
        st_uid : 0
        st_gid : 0
        st_rdev : 0
        st_size : 0
        st_blksize : 4096
        st_blocks : 0
        st_atime : 1478257091
        st_mtime : 1478257091
        st_ctime : 1478257091
```

Notice the change in **st_mode** by using fchmod(). It changes from 33216 to 33152.

```c
/* donate to first non-root user */
if (fchown(fd, 1000, 1000) != 0) {
    err("fchown() failed: [%s]", strerror(errno));
    return;
} else {
    info("[4] fchown() : success => donated to user");
}
print_stat(fd);

/* get back ownership */
if (fchown(fd, 0, 0) != 0) {
    err("fchown() failed: [%s]", strerror(errno));
    return;
} else {
    info("[4] fchown() : success => restored");
}
print_stat(fd);
```

```
[4] fchown() : success => donated to user
[7] readlink() : success => /home/gvkalra/Desktop/EE516/HW03/task01.dat
[8] fstat() => /home/gvkalra/Desktop/EE516/HW03/task01.dat
        st_dev : 2066
        st_ino : 3148596
        st_mode : 33216
        st_nlink : 1
        st_uid : 1000
        st_gid : 1000
        st_rdev : 0
        st_size : 0
        st_blksize : 4096
        st_blocks : 0
        st_atime : 1478257091
        st_mtime : 1478257091
        st_ctime : 1478257091
[4] fchown() : success => restored
[7] readlink() : success => /home/gvkalra/Desktop/EE516/HW03/task01.dat
[8] fstat() => /home/gvkalra/Desktop/EE516/HW03/task01.dat
        st_dev : 2066
        st_ino : 3148596
        st_mode : 33216
        st_nlink : 1
        st_uid : 0
        st_gid : 0
        st_rdev : 0
        st_size : 0
        st_blksize : 4096
        st_blocks : 0
        st_atime : 1478257091
        st_mtime : 1478257091
        st_ctime : 1478257091
```

Notice the change in **st_uid** and **st_gid** by using fchown(). It changes from 0:0 (root:root) to 1000:1000 (user:user)

Note:
1000 is the first non-root user created in ubuntu

Notice the **inode number** for task01.dat and it's soft (different) & hard links (same).

```
[5] link() : success => task01.dat-hardlink
[6] symlink : success => task01.dat-softlink
[9] readdir() => d_ino (d_name)
        3148596 (task01.dat)
        3146255 (task01)
        3148596 (task01.dat-hardlink)
        3146198 (.)
        3146259 (utils.h)
        3146257 (Makefile)
        3148597 (task01.dat-softlink)
        3148595 (task01.c)
        2760540 (..)

##### Seeking/Linking Play #####

[2] unlink() : success => task01.dat
[1] lseek() : success => task01.dat-hardlink
<links_play:235> open() failed: [No such file or directory]
[2] unlink() : success => task01.dat-hardlink
[2] unlink() : success => task01.dat-softlink
```

```c
/* hardlink FILENAME */
if (link(FILENAME, FILENAME HARDLINK_SUFFIX) != 0) {
    err("link() failed: [%s]", strerror(errno));
    goto error;
} else {
    info("[5] link() : success => %s", FILENAME HARDLINK_SUFFIX);
    hard_link = 1;
}

/* softlink FILENAME */
if (symlink(FILENAME, FILENAME SOFTLINK_SUFFIX) != 0) {
    err("symlink() failed: [%s]", strerror(errno));
    goto error;
} else {
    info("[6] symlink : success => %s", FILENAME SOFTLINK_SUFFIX);
    soft_link = 1;
}

/* list files */
list_files(cwd_path);
```

Notice that it is not possible to open() or lseek() softlink if task01.dat is removed.

However, hardlink is not affected by removal of task01.dat

```c
/* free file resources */
if (_fd != -1) {
    /* close file */
    if (close(_fd) == -1) {
        err("close() failed: [%s]", strerror(errno));
    }
    /* unlink */
    if (unlink(FILENAME)) {
        err("unlink() failed: [%s]", strerror(errno));
    } else {
        info("[2] unlink() : success => %s", FILENAME);
    }
}
```

```c
_fd = open(FILENAME HARDLINK_SUFFIX, O_RDONLY);
if (_fd == -1) {
    err("open() failed: [%s]", strerror(errno));
    /* skip seeking */
} else {
    if (lseek(_fd, 0, SEEK_END) == (off_t)-1) {
        err("lseek() failed: [%s]", strerror(errno));
    }
    info("[1] lseek() : success => %s", FILENAME HARDLINK_SUFFIX);
    close(_fd);
}

_fd = open(FILENAME SOFTLINK_SUFFIX, O_RDONLY);
if (_fd == -1) {
    err("open() failed: [%s]", strerror(errno));
    /* skip seeking */
} else {
    if (lseek(_fd, 0, SEEK_END) == (off_t)-1) {
        err("lseek() failed: [%s]", strerror(errno));
    }
    info("[1] lseek() : success => %s", FILENAME SOFTLINK_SUFFIX);
    close(_fd);
}
```

Problem 3: Explain the following functions
malloc(), calloc(), realloc(), free()

[1] malloc()

Allocates **un-initialized memory** for **given number of bytes**

[2] calloc()

Allocates **initialized (set to zero) memory** for **given number of elements of specified bytes each**

[3] realloc()

Changes size of memory block for given pointer to specified bytes

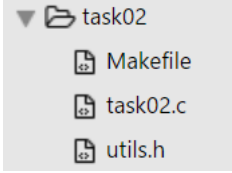
[4] free()

Frees memory allocated by [1], [2] or [3]

References:

http://man7.org/linux/man-pages/man3/malloc.3.html

Problem 4: Make your own program using malloc(), calloc(), realloc() & free()




```c
int main(int argc, const char *argv[])
{
    test_malloc();
    test_calloc();
    test_realloc();
    return 0;
}
```

```
TESTING : malloc()
        Requested : 128 bytes
        Allocated : 136 bytes
        0000: 00 00 00 00 00 00 00 00   00 00 00 00 00 00 00 00
        0010: 00 00 00 00 00 00 00 00   00 00 00 00 00 00 00 00
        0020: 00 00 00 00 00 00 00 00   00 00 00 00 00 00 00 00
        0030: 00 00 00 00 00 00 00 00   00 00 00 00 00 00 00 00
        0040: 00 00 00 00 00 00 00 00   00 00 00 00 00 00 00 00
        0050: 00 00 00 00 00 00 00 00   00 00 00 00 00 00 00 00
        0060: 00 00 00 00 00 00 00 00   00 00 00 00 00 00 00 00
        0070: 00 00 00 00 00 00 00 00   00 00 00 00 00 00 00 00
        0080:
```

Notice that allocated size is > requested size. This depends on implementation of malloc() or any other memory allocator. However, as a programmer, we should only utilize requested size of memory. Also observe that although hex dump of allocated memory is initialized to 'zero' in this case, it may not hold true always. Reference: http://stackoverflow.com/a/8029624

```
TESTING : calloc()
        Requested : 2 blocks * 65 bytes = 130 bytes
        Allocated : 136 bytes
        0000: 00 00 00 00 00 00 00 00   00 00 00 00 00 00 00 00
        0010: 00 00 00 00 00 00 00 00   00 00 00 00 00 00 00 00
        0020: 00 00 00 00 00 00 00 00   00 00 00 00 00 00 00 00
        0030: 00 00 00 00 00 00 00 00   00 00 00 00 00 00 00 00
        0040: 00 00 00 00 00 00 00 00   00 00 00 00 00 00 00 00
        0050: 00 00 00 00 00 00 00 00   00 00 00 00 00 00 00 00
        0060: 00 00 00 00 00 00 00 00   00 00 00 00 00 00 00 00
        0070: 00 00 00 00 00 00 00 00   00 00 00 00 00 00 00 00
        0080: 00 00
```

In case of calloc(), allocated memory is always initialized to 'zero'.

```
TESTING : realloc()
        Requested : 128 bytes
        Allocated : 136 bytes
        0000: 00 00 00 00 00 00 00 00   00 00 00 00 00 00 00 00
        0010: 00 00 00 00 00 00 00 00   00 00 00 00 00 00 00 00
        0020: 00 00 00 00 00 00 00 00   00 00 00 00 00 00 00 00
        0030: 00 00 00 00 00 00 00 00   00 00 00 00 00 00 00 00
        0040: 00 00 00 00 00 00 00 00   00 00 00 00 00 00 00 00
        0050: 00 00 00 00 00 00 00 00   00 00 00 00 00 00 00 00
        0060: 00 00 00 00 00 00 00 00   00 00 00 00 00 00 00 00
        0070: 00 00 00 00 00 00 00 00   00 00 00 00 00 00 00 00
        0080:
```

```
Increasing to : 150 bytes
Allocated : 152 bytes
0000: 00 00 00 00 00 00 00 00   00 00 00 00 00 00 00 00
0010: 00 00 00 00 00 00 00 00   00 00 00 00 00 00 00 00
0020: 00 00 00 00 00 00 00 00   00 00 00 00 00 00 00 00
0030: 00 00 00 00 00 00 00 00   00 00 00 00 00 00 00 00
0040: 00 00 00 00 00 00 00 00   00 00 00 00 00 00 00 00
0050: 00 00 00 00 00 00 00 00   00 00 00 00 00 00 00 00
0060: 00 00 00 00 00 00 00 00   00 00 00 00 00 00 00 00
0070: 00 00 00 00 00 00 00 00   00 00 00 00 00 00 00 00
0080: 00 00 00 00 00 00 00 00   61 0B 02 00 00 00 00 00
0090: 00 00 00 00 00 00
```

```
Decreasing to : 128 bytes
Allocated : 152 bytes
0000: 00 00 00 00 00 00 00 00   00 00 00 00 00 00 00 00
0010: 00 00 00 00 00 00 00 00   00 00 00 00 00 00 00 00
0020: 00 00 00 00 00 00 00 00   00 00 00 00 00 00 00 00
0030: 00 00 00 00 00 00 00 00   00 00 00 00 00 00 00 00
0040: 00 00 00 00 00 00 00 00   00 00 00 00 00 00 00 00
0050: 00 00 00 00 00 00 00 00   00 00 00 00 00 00 00 00
0060: 00 00 00 00 00 00 00 00   00 00 00 00 00 00 00 00
0070: 00 00 00 00 00 00 00 00   00 00 00 00 00 00 00 00
0080:
```

Notice that realloc() doesn't initialize memory. It simply increases or decreases an already allocated memory.