

# Shadow File System

Patrick Coughlin - [ptcoughlin@gwmail.gwu.edu](mailto:ptcoughlin@gwmail.gwu.edu)

Cassius Ali - [pcali@gwmail.gwu.edu](mailto:pcali@gwmail.gwu.edu)

Richard Lancia - [rlancia7@gwmail.gwu.edu](mailto:rlancia7@gwmail.gwu.edu)

Alex Ertel - [atertel@gwu.edu](mailto:atertel@gwu.edu)

CSCI 3411 - Operating Systems  
Fall 2014

## **Introduction**

The objective of this project was to implement a file system in user-space using FUSE C. The basis of this file system would organize data contained within the shadow file in CentOS5 into directories, subdirectories, and then individual files. The overarching data structure is a tree implementation, where */shadow* is the root of the tree and contains subdirectories consisting of all the users listed within the shadow file. The actual shadow file is found within the directory */etc/shadow*.

Each user subdirectory has six files within it, containing the following information:

- User name
- Password Hash
- Number of days since the password was changed
- Number of days the user can change their password
- Number of days the user must change their password by
- Number of days the user will be warned about their password needing to be changed

## **Background Information and Design Plan**

FUSE C is a file-system implementation that resides entirely in user-space. The functions contained within FUSE C, and particularly the library *fuse.h*, transition briefly between user space and kernel space, but the FUSE API allows for students to design and deal only with user level function calls, making FUSE behave as a veritable microkernel.

To gain a solid background and understanding of the internal workings of FUSE C students actively researched all of the function calls contained within the *./hello* example file system implementation. After becoming acquainted with these processes, students based their shadow file system off of the general function calls within the example system.

Modifications were made to the back end data structures that structured the system to accommodate the information that the shadow file contained. Initializing the data structures required parsing functions, which stored each piece of information as individual variables, and then organized the variables into distinct nodes (one node per user, containing all of that user's information). The actual structure for such nodes can be found within the implementation section of the report. These nodes were then organized with one directory per user and six files per user directory using the functions *mkdir()* and *mknod()*, respectively.

## Software and Tools

- fuse-2.7.4
- CentOS5 - Linux Redhat Enterprise 4
- GitHub (for filesharing)

## Code Implementation

*shadow\_example.txt*

```
root:$1$jPzYRwZi$bfcWaH1tRVERi9Pr5jfx0.:13783:0:99999:7:::
bin:*:13783:0:99999:7:::
daemon:*:13783:0:99999:7:::
adm:*:13783:0:99999:7:::
lp:*:13783:0:99999:7:::
sync:*:13783:0:99999:7:::
shutdown:*:13783:0:99999:7:::
halt:*:13783:0:99999:7:::
mail:*:13783:0:99999:7:::
news:*:13783:0:99999:7:::
uucp:*:13783:0:99999:7:::
operator:*:13783:0:99999:7:::
games:*:13783:0:99999:7:::
gopher:*:13783:0:99999:7:::
ftp:*:13783:0:99999:7:::
nobody:*:13783:0:99999:7:::
rpm:!!:13783:0:99999:7:::
dbus:!!:13783:0:99999:7:::
mailnull:!!:13783:0:99999:7:::
```

Figure 1: *shadow\_example.txt* Snippet

This file lists information for 33 separate users, which would initialize 33 directories in */shadow*. The file is colon delimited and is separated with the parsing functions in *shadowParseTest.c*.

*shadowParseTest.c*

```
#define shadowFile "shadow_example.txt"

int lines = 0;

typedef struct shadowDataNode{
    char *user;
    char *pw_hash;
    int numDays;
    int daysCanChange;
    int daysMustChange;
    int daysWarn;
    struct shadowDataNode *next, *prev;
}shadowDataNode;
```

Figure 2: Struct Definition

The struct node in Figure 2 is used to store all the information per user listed in the shadow file.

```
shadowDataNode *parse(shadowDataNode *head) {
    char *temp_str;
    int i;
    shadowDataNode *x;
    tempUser *users = create();
    head = malloc(sizeof(shadowDataNode));
    x = head;
    for (i = 0; i < lines; i++) {
        temp_str = strtok(users[i].userinfo, ":");
        x->user = temp_str;

        temp_str = strtok(NULL, ":");
        x->pw_hash = temp_str;

        temp_str = strtok(NULL, ":");
        x->numDays = atoi(temp_str);

        temp_str = strtok(NULL, ":");
        x->daysCanChange = atoi(temp_str);

        temp_str = strtok(NULL, ":");
        x->daysMustChange = atoi(temp_str);

        temp_str = strtok(NULL, ":");
        x->daysWarn = atoi(temp_str);

        if (i != (lines-1)) {
            x->next = malloc(sizeof(shadowDataNode));
            x = x->next;
        }
    }
    free(users);
    return head;
}
```

Figure 3: Parse Function

The *parse()* function in Figure 3 sorts the shadow file information to the correct nodes (which are dynamically allocated in *create()*) and returns the *head* pointer, to be used in *shadow\_fs.c*.

*shadow\_fs.c*

```
int main(int argc, char *argv[])
{
    int res;
    res = shadow_init();
    if (res == -1) return -errno;
    return fuse_main(argc, argv, &shadow_oper, NULL);
}
```

Figure 4: *shadow\_fs* Main

The *main()* function in Figure 4 calls the *initialize()* function, which iterates through the linked list created in the parse function, and writes each node into files in a directory. Each system function returns a succeed/fail value, which then returns the error number if the process fails. These create paths for each directory and file, which could then be utilized when navigating the filesystem via Terminal.

```

static int shadow_init(void) {
    int res, i=0;
    char *path;
    char *attr_path = malloc(20*sizeof(char));
    char *str_int = malloc(10*sizeof(char));
    head = parse(head);
    shadowDataNode *x = head;

    for(i=0;i<lines;i++) { //lines is the length of the linked list
        strcpy(path, shadow_path);
        strcat(path, "/");
        strcat(path, "user_");
        strcat(path, itoa(i, str_int, 10));
        res = mkdir(path, S_IRWXU);
        if (res == -1) return -errno;
        strcat(path, "/");

        //Name
        strcpy(attr_path, path);
        strcat(attr_path, "Username");
        int fd = open(attr_path, O_CREAT | O_EXCL | O_RDWR);
        if (fd == -1) return -errno;
        res = pwrite(fd, x->user, sizeof(x->user), 0);
        close(fd);
        if (res == -1) return -errno;
    }
}

```

Figure 5: Initialize Function

Other functions to be implemented include those listed in the *shadow\_oper* struct in Figure 6, which serve as the APIs for the functions which make the system readable and writable by a user.

```

static struct fuse_operations shadow_oper = {
    .getattr    = shadow_getattr,
    .readdir    = shadow_readdir,
    .mknod      = shadow_mknod,
    .mkdir      = shadow_mkdir,
    .unlink     = shadow_unlink,
    .rmdir      = shadow_rmdir,
    .open       = shadow_open,
    .read       = shadow_read,
    .write      = shadow_write,
}

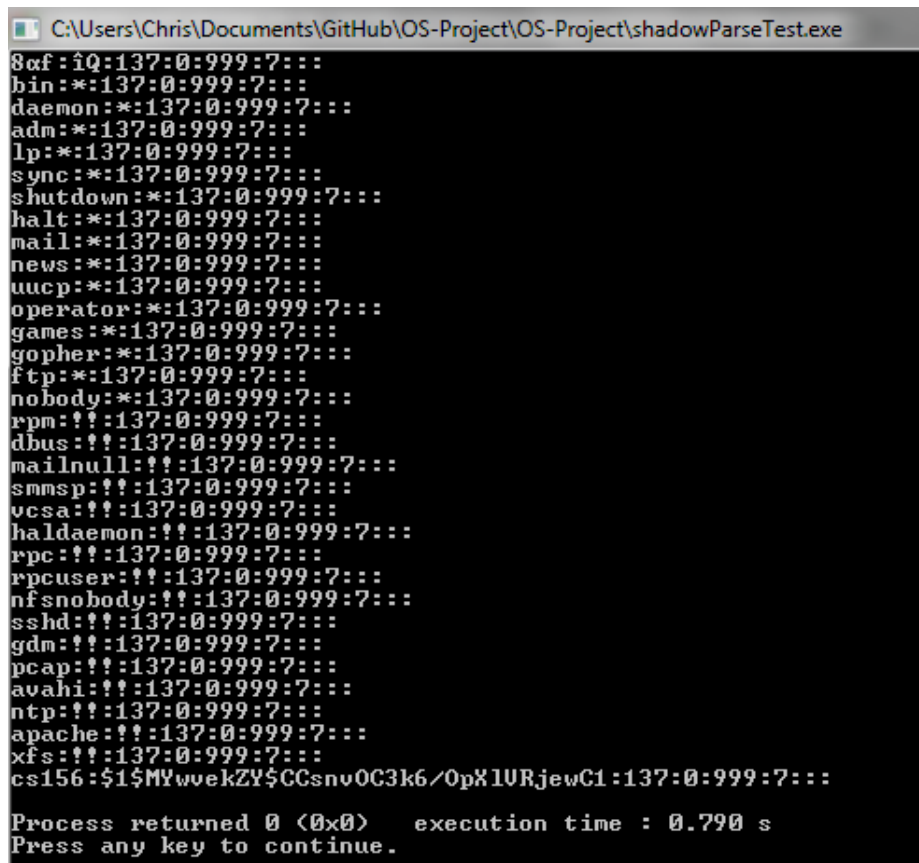
```

Figure 6: Operations

## Testing and Results

Testing began with the parsing function, which could be executed in a normal C programming environment. To initially test the parsing functions, students created a temporary copy of the shadow file on their own machines. The parsing of the file was broken down into two main steps.

The first step was to take the shadow file, in its entirety, and hold it within a single dimensional array. This was accomplished by the *create()* function referenced in Figure 3, within the *parse()* function. The *create()* function used a *for()* loop and *fgets()* to parse through the shadow file line-by-line. Each line was contained within a temporary *myUsers* array. This array was then modified by the parsing function via *strtok()*. *strtok()* tokenized the colon delimited array and stored each piece of information into its respective variable within the *shadowDataNode* struct. After the parsing was completed, the function returned a head pointer whose contents was printed to the command line to ensure that the parsing was executed correctly, as can be seen in Figure 7.



```
C:\Users\Chris\Documents\GitHub\OS-Project\OS-Project\shadowParseTest.exe
8xf:iQ:137:0:999:7:::
bin:*:137:0:999:7:::
daemon:*:137:0:999:7:::
adm:*:137:0:999:7:::
lp:*:137:0:999:7:::
sync:*:137:0:999:7:::
shutdown:*:137:0:999:7:::
halt:*:137:0:999:7:::
mail:*:137:0:999:7:::
news:*:137:0:999:7:::
uucp:*:137:0:999:7:::
operator:*:137:0:999:7:::
games:*:137:0:999:7:::
gopher:*:137:0:999:7:::
ftp:*:137:0:999:7:::
nobody:*:137:0:999:7:::
rpm:!!:137:0:999:7:::
dbus:!!:137:0:999:7:::
mailnull:!!:137:0:999:7:::
smb:!!:137:0:999:7:::
vcsa:!!:137:0:999:7:::
haldaemon:!!:137:0:999:7:::
rpc:!!:137:0:999:7:::
rpcuser:!!:137:0:999:7:::
nfsnobody:!!:137:0:999:7:::
sshd:!!:137:0:999:7:::
gdm:!!:137:0:999:7:::
pcap:!!:137:0:999:7:::
avahi:!!:137:0:999:7:::
ntp:!!:137:0:999:7:::
apache:!!:137:0:999:7:::
xfs:!!:137:0:999:7:::
cs156:$1$MYwvekZY$CCsnuOC3k6/OpXlURjewC1:137:0:999:7:::
Process returned 0 (0x0)   execution time : 0.790 s
Press any key to continue.
```

Figure 7: Printed Parsing Results

After students debugged the parsing function, *parse()* was used within *shadow\_init()*. This function initializes the tree-structured file system, based on the contents stored within the linked list returned from *parse()*. *shadow\_init()* creates the directory structure by iterating through the linked list and concatenating the values stored within the *shadow\_path* to the *path*. This creates a hierarchical tree-like structure composed of all the information contained within the shadow file.

Unfortunately, due to an unknown error, the list could not be viewed with the *ls* command and therefore could not be navigated or edited. It was initially thought that the error was due to opening the incorrect path via *opendir()* in the *shadow\_readdir()* function, but nothing happened upon correcting that error.

## **Conclusion**

Since the code compiled correctly but failed to function correctly in Linux, it could not be tested and therefore cannot quite be considered a success. Theoretically, the code thus far should initialize all the information fields in the shadow file into a tree-based file system in Linux. Upon running the *ls* command, the system was not able to be viewed in Terminal. However, the parsing function correctly sorts all the information into usable variables, so that part of the code can be considered successful. Linking the shadow file variables to FUSE functions and system calls is where the difficulties arose.

With more experience in Linux, the errors could possibly be worked around and fixed. The first major compiling error was fixed with various solutions, including adding new C libraries and testing with different functions, such as *snprintf()* rather than *itoa()*. The segmentation faults were also quickly fixed by inspecting the memory allocations and frees multiple times.

Future aspirations for this project would include fixing the errors and at least being able to view everything in Terminal. Ideally, the user could navigate the tree and view all the information in the files.