March 24, 2021

GTU Department of Computer Engineering
CSE344 - Spring 2021
Homework 1 Report

Akif Kartal
171044098

1 Problem Definition

The problem is to write an "advanced" file search program for POSIX compatible operating systems.

2 Solution

In order to write this program we need to **divide problem into modules**. These modules are following;

2.1 Argument Handling

In order to manage arguments in a proper way I used **argument struct** to keep given argument data as one block data. My argument struct is following.

```
typedef struct st
{
    int wFlag;
    int fFlag;
    int bFlag;
    int tFlag;
    int pFlag;
    int lFlag;
    char *wArg;
    char *fArg;
    char *bArg;
    char *tArg;
    char *pArg;
    char *lArg;
    int isFound;
    int count;

} args;
```

In this struct **count** represent number of optional arguments and **isFound** represent file is found or not. Also, in order to get argument's value **getopt()** library method was used.

2.2 Error Handling(Bonus)

In order to handle any error **stderr** used with **write system call** and **exit** system call was used.

- **Note:** In this part **fprintf library function was not used**. Instead write system call was used.

Following code shows an example of this;

```
void my_fprintf_with_stderr(const char *str){
    ssize_t size = strlen(str);
    if (size != write(STDERR_FILENO, str, size)) {
        perror("write system call error!");
        exit(EXIT_FAILURE);
    }

}
```

Usage:

```
if (closedir(dir))
{
    char *str = "Directory close error!!\n";
    my_fprintf_with_stderr(str);
    exit(EXIT_FAILURE);
}
```

2.3 Advanced Search Algorithm

2.3.1 Regex Handling

The filename argument can contain more than one "+" character. To handle this situation in an easy way somehow we need to keep position data of these regexs. In order to do this I created my own **LinkedList** data structure to keep both position and previous letter information. **My LinkedList** node is following;

```
//regex information node
typedef struct node_s
{
    int position;
    char preChr;
    struct node_s *next;

} node_t;
```

2.3.2 Checking Given Parameters

- File name check was made by using regex linkedlist.
- File size, file type, file permissions and file links was checked by using **stat calls**.

2.3.3 Recursive Search Algorithm

In this part we need to examine the files in the given target directory. In order to do this operation, we need to use **opendir** function. Calling the opendir returns a **DIR*** type, which we will use to access the directory contents. After that we need to use **readdir** function with DIR* varible. Each time we call readdir, it returns a pointer to a **struct dirent** instance corresponding to the next directory entry. The struct dirent that we get back from readdir has a field **d_name**, which contains the name of the directory entry. So, in order to continue recursively traverse all directories, we need to use this d_name file. After readdir call if directory is different than current and parent we will **concatenate "/" and d_name end of the current directory name** after that when we call traversal function with this new path, we are able to examine and traverse inner directories. While making this also we need to check current file it is searched file or not. In order to do that we need to use both d_name and current path to use in stat function. Filename will be used checking filename flag. Lastly, we need close directory by calling **closedir** function, passing the DIR* type, to end the directory listing operation. Following **small code pieces** show this process.

```
while ((entry = readdir(dir)) != NULL)
{
    //check directory is different than current and parent
    if (strcmp(entry->d_name, ".") != 0 && strcmp(entry->d_name, "..") != 0)
    {
        if (entry->d_type == DT_DIR)
        {
            checkGivenArguments(targetPath, givenArgs, entry->d_name);
            char currentPath[PATH_MAX];
            size_t size = strlen(targetPath);
            if (targetPath[size - 1] == '/')
                sprintf(currentPath, "%s%s", targetPath, entry->d_name);
            else
                sprintf(currentPath, "%s/%s", targetPath, entry->d_name);
            size_t pathLength = strlen(currentPath);
            traversePathRecursively(currentPath, givenArgs); //call function with child
        }
    }
}
//close directory
```

2.3.4 Drawing nicely formatted tree

If the searching file is found then directory tree will be drawn by using same recursive search algorithm with just a **minor** difference.

2.4 CTRL-C Handling

In order to give a message on CTRL-C interrupt, I used **sigaction** function from **signal.h** library. Also, I used a **global variable** to set if an interrupt has occur.

```
volatile __sig_atomic_t exitSignal = 0;
void exitHandler(int signal)
{
    if (signal == SIGINT)
    {
        exitSignal = 1;
    }
}
```

In each loop in code, signal flag was checked, on interrupt signal, resources was given back and exited elegantly.

3 Test Result

A simple test result is following;

Figure 1