CSE 344 System Programming

HW1

Alp Emir Bilek *Student Id:161044049*

Directory Traverser: Reads a path from user via terminal and some parameters.Then constructs a directory tree for requested file.

- Getopt() used for getting and parsing the parameters from keyboard. -W parameter is mandatory.If it is NULL gives an error and terminates the program.Also another parameter needed beside -W parameter(also terminates the program unless when one more parameter isn't given.).

```
while((opt = getopt(argc, argv, ":w:f:b:t:l:p:")) != -1)
{
    switch(opt)
    {
        case 'f':
            flag_control++;
            file_name=optarg;
            break;
        case 'b':
            flag_control++;
            file_size=atoi(optarg);
            break;
        case 't':
            flag_control++;
            file_type=optarg[0];
            break;
        case 'p':
            flag_control++;
            permissions=optarg;
            break;
        case 'l':
            flag_control++;
            links=atoi(optarg);
            break;
        case 'w':
            directory=optarg;
            break;
        case '?':
            err_flag++;
            break;
    }
}
if(strcmp(directory,"")==0){
    perror("Directory cannot be empty");
}
else if(flag_control<1){
    perror("You gotta enter one more parameter");
}
else if(err_flag>0){
    perror("Unknown option");
```

- -w parameter can be given as full path or just the directory name.(Ex:/home/alp/Desktop or Desktop)This part understands the which input type it is.Then calls the recursive functions and then if the given input file is found,prints the path(full path.It means if you search on Desktop it prints the whole directories on Desktop).

```
if(strstr(directory,"/")){
    printf("%s\n",directory);
    FileFinder(directory,file_name,file_size,file_type,permissions,links,1);
}
else{
    printf("%s\n",directory);
    DirectoryFinder("/home",directory);
    FileFinder(global_string,file_name,file_size,file_type,permissions,links,1
}
if(countflag>0){
    int i=0;
    for(i=0;i<MAINCOUNT;i++){
        if(flag[i]==1){
            printf(ANSI_COLOR_BLUE "%s" ANSI_COLOR_RESET  ,print[i]);
        }
        else{
            printf("%s",print[i]);
        }
    }
}
else{
    perror("Not Found!!");
}
```

- FileFinder():Takes inputs as parameter(directory,name,size,type,permission,link). Opens a directory with given directory and checks is there a file which is satisfies criterias and applies this processes to whole child directories.If it finds the requested file,stores the name in a global array and then prints those names after searching.

```
void FileFinder(char *directory,char *name,int size,char type,char *permission,int link,int counter)
{
    char path[1000];
    struct dirent *dp;
    DIR *dir = opendir(directory);
    if (!dir)
        return;
    while ((dp = readdir(dir)) != NULL)
    {
        if (strcmp(dp->d_name, ".") != 0 && strcmp(dp->d_name, "..") != 0)
        {
            strcpy(path, directory);
            strcat(path, "/");
            strcat(path, dp->d_name);
            struct stat filestat;
            stat(path,&filestat);
                if(dp->d_type==DT_DIR){
                    if(parameterValidator(name,dp->d_name,size,filestat.st_size,type,filestat.st_mode,permission,link,filestat.st_nlink)){
                        printer(counter);
                        strcpy(print[MAINCOUNT],dp->d_name);
                        flag[MAINCOUNT]=1;
                        MAINCOUNT++;
                        strcpy(print[MAINCOUNT],"\n");
                        MAINCOUNT++;
                        countflag++;
                        flag[MAINCOUNT]=1;

                    }
                    else{
                        printer(counter);
                        strcpy(print[MAINCOUNT],dp->d_name);
                        flag[MAINCOUNT]=0;
                        MAINCOUNT++;
                        strcpy(print[MAINCOUNT],"\n");
                        MAINCOUNT++;
                        flag[MAINCOUNT]=0;
                        FileFinder(path,name,size,type,permission,link,counter+1);
                    }
                }
                else {
                    if(parameterValidator(name,dp->d_name,size,filestat.st_size,type,filestat.st_mode,permission,link,filestat.st_nlink)){
                        printer(counter);
                        strcpy(print[MAINCOUNT],dp->d_name);
                        flag[MAINCOUNT]=1;
                        MAINCOUNT++;
                        strcpy(print[MAINCOUNT],"\n");
                        MAINCOUNT++;
                        countflag++;
                        flag[MAINCOUNT]=1;
                    }
```

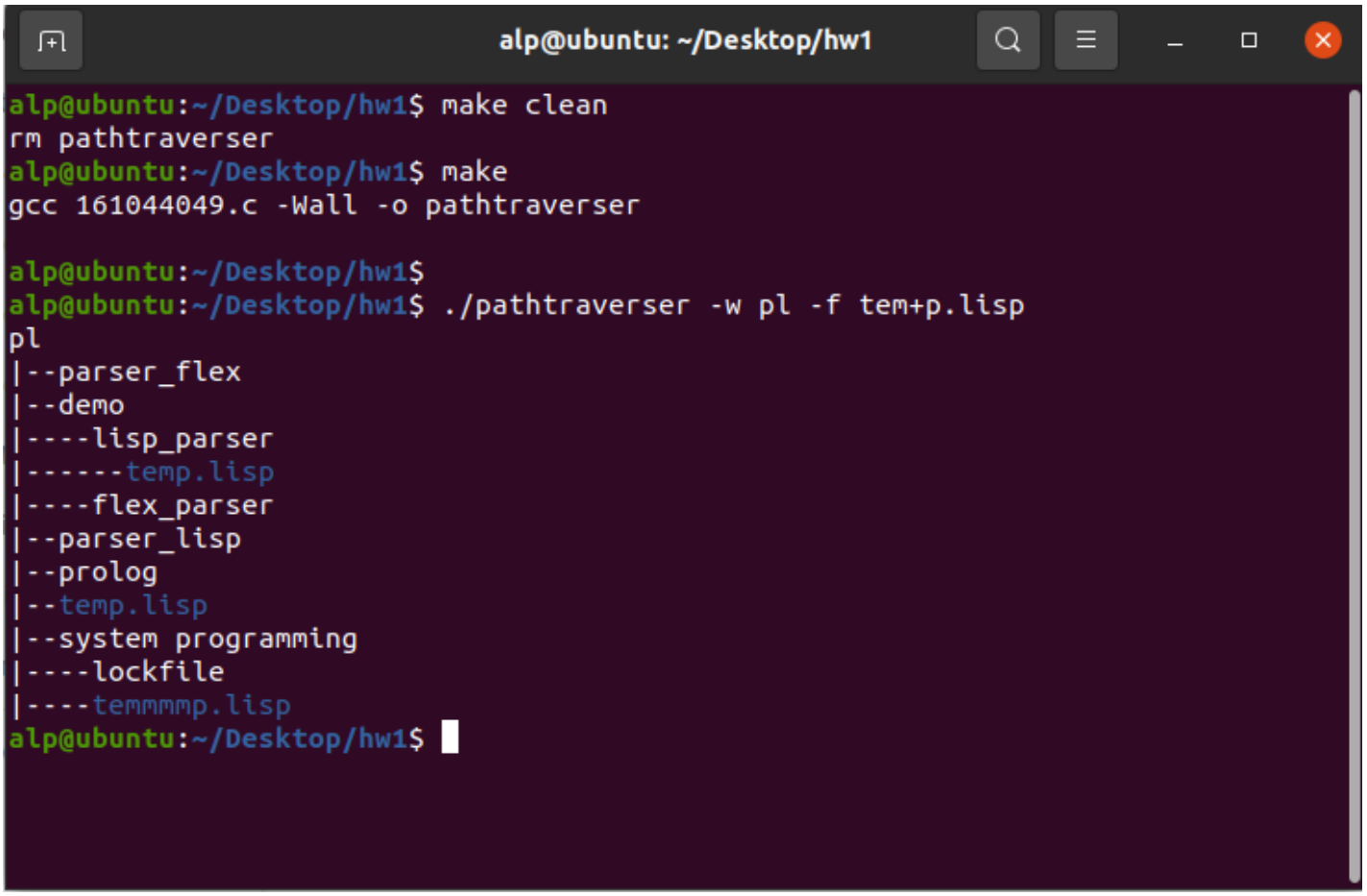
```
bool parameterValidator(char *input_name,char *file_name,int input_size,int size,char input_type,mode_t Type,char *input_permission,int input_link,int link){
    bool name_flag,size_flag,type_flag,perm_flag,link_flag;
    char Perm[10];
    if(strcmp(input_name,"")==0){
        name_flag=true;
    }
    else {
        name_flag=strcmp(input_name,file_name)==0;
        if(strstr(input_name,"+")){
            name_flag=regex(input_name,file_name);
        }
    }

    if(input_size==0){
        size_flag=true;
    }
    else{
        size_flag= input_size==size;
    }

    if(input_type == 'q'){
        type_flag=true;
    }
    else{
        type_flag = (S_ISREG(Type) && input_type == 'f')  || (S_ISDIR(Type) && input_type == 'd')  ||  (S_ISCHR(Type) && input_type == 'c') ||
            (S_ISBLK(Type) && input_type == 'b')  || (S_ISFIFO(Type) && input_type == 'p')  ||  (S_ISSOCK(Type) && input_type == 's')  ||  (S_ISLNK(Type) && input_type == 'l');
    }

    if(strcmp(input_permission,"")==0){
        perm_flag=true;
    }
    else {
        Perm[0]=(Type & S_IRUSR) ? 'r' : '-';
        Perm[1]=(Type & S_IWUSR) ? 'w' : '-';
        Perm[2]=(Type & S_IXUSR) ? 'x' : '-';
        Perm[3]=(Type & S_IRGRP) ? 'r' : '-';
        Perm[4]=(Type & S_IWGRP) ? 'w' : '-';
        Perm[5]=(Type & S_IXGRP) ? 'x' : '-';
        Perm[6]=(Type & S_IROTH) ? 'r' : '-';
        Perm[7]=(Type & S_IWOTH) ? 'w' : '-';
        Perm[8]=(Type & S_IXOTH) ? 'x' : '-';
        perm_flag=strcmp(input_permission,Perm)==0;
    }

    if(input_link==0){
        link_flag=true;
    }
    else{
        link_flag= input_link==link;
    }
    return name_flag && size_flag && type_flag && perm_flag && link_flag;
```

- parameterValidator():This function checks the given inputs satisfies the current file or not. In this function we have 5 flags(name,size,type,permission,link).The logic is same on all flags.Firstly we are checking is it null or not because it can be null.If it is null the flag value will be true because it means user doesn't care the current flag.If it is not null,we compare with the current file stats and according to this examination flag will become true or false.At the end of the function we AND the whole flags and return the result as file is valid or not.

- regex():This function used for if given file name contains '+' .The first process will be split the input name via sscanf().Then checked the equality between input name's head and tail with current file name's head and tail.If it is equal,check the middle with the previous character from '+'.Even it's true then return true.

```
bool regex(char *input_name,char *file_name){
    int i;
    for(i=0;i<strlen(input_name);i++){
        input_name[i]=tolower(input_name[i]);
    }
    for(i=0;i<strlen(file_name);i++){
        file_name[i]=tolower(file_name[i]);
    }
    char *last=(char*)calloc(1024,sizeof(char));
    char *first=(char*)calloc(1024,sizeof(char));
    char *dotpart=(char*)calloc(1024,sizeof(char));
    char c;
    int j=0,counterf=0,counterl=0,counterMain=0;
    if(strstr(input_name,".")){
        sscanf(input_name,"%[a-zA-Z0-9]+%[a-zA-Z0-9]  .%[a-zA-Z0-9]",first,last,dotpart);
        strcat(last,".");
        strcat(last,dotpart);
    }
    else{
        sscanf(input_name,"%[a-zA-Z0-9]+%[a-zA-Z0-9]",first,last);
    }
    c=first[strlen(first)-1];
    for(i=0;i<strlen(first);i++){
        if(file_name[i]==first[i]){
            counterf++;
        }
    }
    for(i=strlen(file_name)-strlen(last),j=0;i<strlen(file_name);i++,j++){
        if(file_name[i]==last[j]){
            counterl++;
        }
    }
    if(counterf==strlen(first) && counterl==strlen(last)){
        for(i=strlen(first);i<strlen(file_name)-strlen(last);i++){
            if(file_name[i]==c){
                counterMain++;
            }
        }
        if((strlen(file_name)-strlen(last)-strlen(first))==counterMain){
            free(last);
            free(first);
            free(dotpart);
            return true;
        }
    }
```

- -W parameter stars from /home as a default value(if user didn't enter full path.Ex:Desktop).Because if we start from root("/") there are so many "permission denied" file on the root it can occur some errors.Also it takes too much time because it traverse the whole computer's file.So you SHOULD NOT start searching from root directory. For example;


alp@ubuntu: ~/Desktop/hw1
alp@ubuntu:~/Desktop/hw1$ make clean
rm pathtraverser
alp@ubuntu:~/Desktop/hw1$ make
gcc 161044049.c -Wall -o pathtraverser
alp@ubuntu:~/Desktop/hw1$
alp@ubuntu:~/Desktop/hw1$ ./pathtraverser -w pl -f tem+p.lisp
pl
|--parser_flex
|--demo
|----lisp_parser
|------temp.lisp
|----flex_parser
|--parser_lisp
|--prolog
|--temp.lisp
|--system programming
|----lockfile
|----temmmmp.lisp
alp@ubuntu:~/Desktop/hw1$


alp@ubuntu:~/Desktop/hw1$ ./pathtraverser -w /home/alp/Desktop/pl -f tem+p.lisp
/home/alp/Desktop/pl
|--parser_flex
|--demo
|----lisp_parser
|------temp.lisp
|----flex_parser
|--parser_lisp
|--prolog
|--temp.lisp
|--system programming
|----lockfile
|----temmmmp.lisp
alp@ubuntu:~/Desktop/hw1$