Name: Raunak Thakur
Roll Number: 2021484

Assumptions:
Throughout the program, for any command, any and all flags should be the very first instructions of their respective argument (However I have handled it so that the order doesn't matter)

Main working:
I have made separate functions for fork-exec type commands and pthread-system type commands, however, at a birds-eye level, they differ in name alone.

So, As soon as the program begins, I store the present working directory globally and begin the shell loop:

```
int main(int argc, char* argv[]){
    _PROGRAM_DIRECTORY = getPWD();
    shell_loop();
```

```
void shell_loop(){
    while(1) {
        char * x = getPWD();
        printf("\033[0;31m<\033[0;36m%s\033[0;31m>\033[0m ",basename(x));
        free(x);
        char **segment = getInput();
        if(segment == NULL){continue;}
        int ptr = 0;
        char* s0 = delim( input: segment[0]);
        //char* s0 = segment[0];
        if (segment[0]==NULL) {}
        else if (strcmp(s0, "exit") == 0) {free(s0); break; }
        else if(strcmp(s0,"cd")==0){ changeDir(segment);}
        else if(strcmp(s0,"pwd")==0){
            char * cwd = getPWD();
            printf("%s\n",cwd);
            free(cwd);
        }
        else if(strcmp(s0,"echo")==0){ echo(segment);}
        else if(strcmp(s0,"mkdir")==0){mkdir1(segment);}
        else if(strcmp(s0,"date")==0){date(segment);}
        else if(strcmp(s0,"ls")==0){ls(segment);}
        else if(strcmp(s0,"rm")==0){rm(segment);}
        else if(strcmp(s0,"cat")==0){cat(segment);}
        else{printf("Segment[0] is %s!\n",segment[0]);printf("Command Not Found!\n");}
        free(s0);

    }
```

This then subdivides the work into if-else statements calling different functions for different commands and execution types (fork vs threads).

Commands:

External: I've used both fork nor threads for these, here is a list:

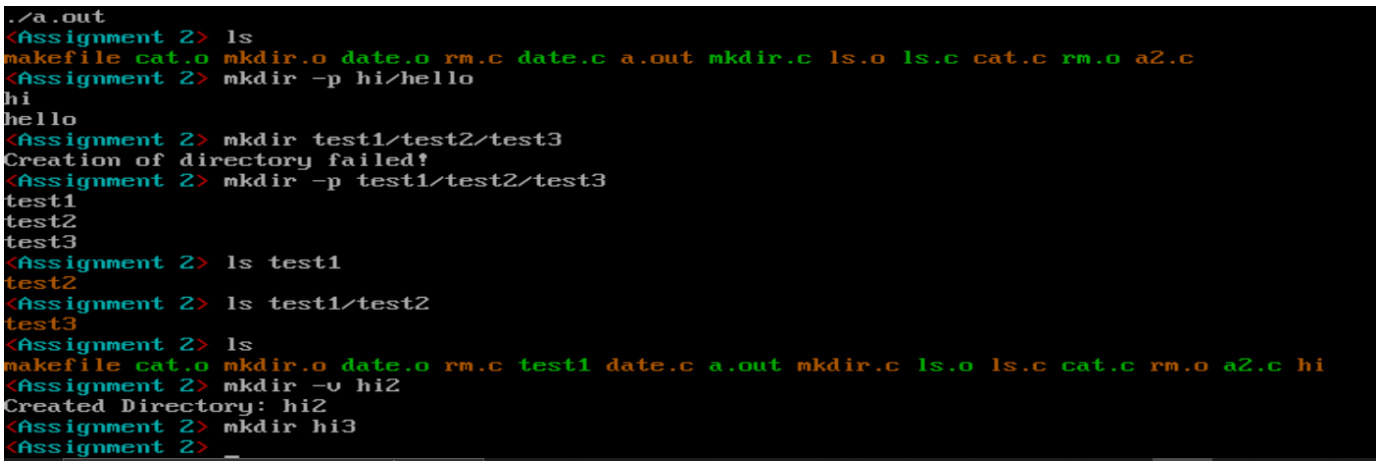
- ls / ls&t:
  - Flags:
    1) **-a**: includes directories and files in the listing that start with a '.'
    2) **-m:** Displays folder contents comma separated (as opposed to simple Space separation normally)
  - Edge Cases:
    a) If a user tries to use space, it'll be concatenated on and treated as plain text.
    b) '\' is handled like in echo
    c) Multiple flags are handled, the user can use -a and -m together

```
<Assignment 2> ls -a
makefile cat.o mkdir.o hi .txt .. date.o rm.c hihi date.c a.out mkdir.c ls.o ls.c cat.c rm.o a2.c temp.txt hi . .temp hi hi
<Assignment 2> ls -m
makefile, cat.o, mkdir.o, hi .txt, date.o, rm.c, hihi, date.c, a.out, mkdir.c, ls.o, ls.c, cat.c, rm.o, a2.c, temp.txt, hi, hi h
i
<Assignment 2> ls -a -m
makefile, cat.o, mkdir.o, hi .txt, .., date.o, rm.c, hihi, date.c, a.out, mkdir.c, ls.o, ls.c, cat.c, rm.o, a2.c, temp.txt, hi,
., .temp, hi hi
<Assignment 2> ls
makefile cat.o mkdir.o hi .txt date.o rm.c hihi date.c a.out mkdir.c ls.o ls.c cat.c rm.o a2.c temp.txt hi hi hi
```

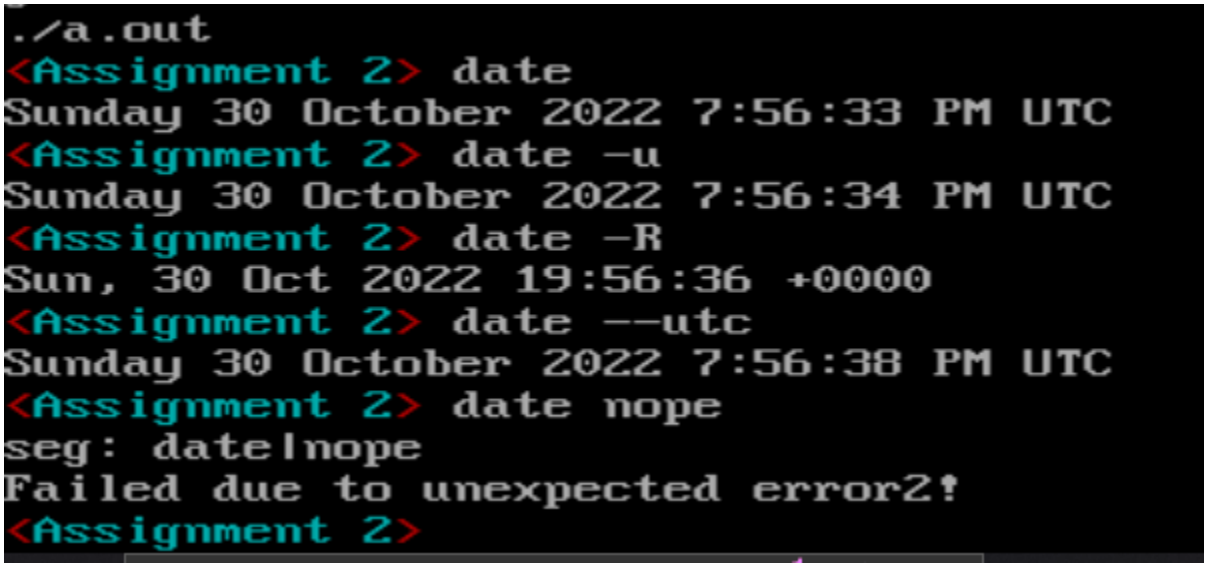
- rm / rm&t:
  - Flags:
    1) **-f:** Forces the deletion, no error message is printed even if the file isn't found
    2) **-v:** Announces successful deletion
  - Edge Cases:
    a) If a user tries to use space, it'll be concatenated on and treated as plain text.
    b) '\' is handled like in echo
    c) multiple flags are handled, the user can use -f and -v together

```
./a.out
<Assignment 2> rm -f THISDOESNOTEXIST
<Assignment 2> ls
makefile cat.o mkdir.o hi .txt date.o rm.c hihi date.c a.out mkdir.c ls.o ls.c cat.c rm.o a2.c temp.txt hi hi hi
<Assignment 2> rm -f THISDOESNOTEXIST
<Assignment 2> rm THISDOESNOTEXIST
File does not exist!
<Assignment 2> rm -v THISDOESNOTEXIST
File does not exist!
<Assignment 2> rm -v -f THISDOESNOTEXIST
<Assignment 2> rm -v -f hi hi
File deleted Successfully!
<Assignment 2> rm temp.txt
<Assignment 2> exit
```

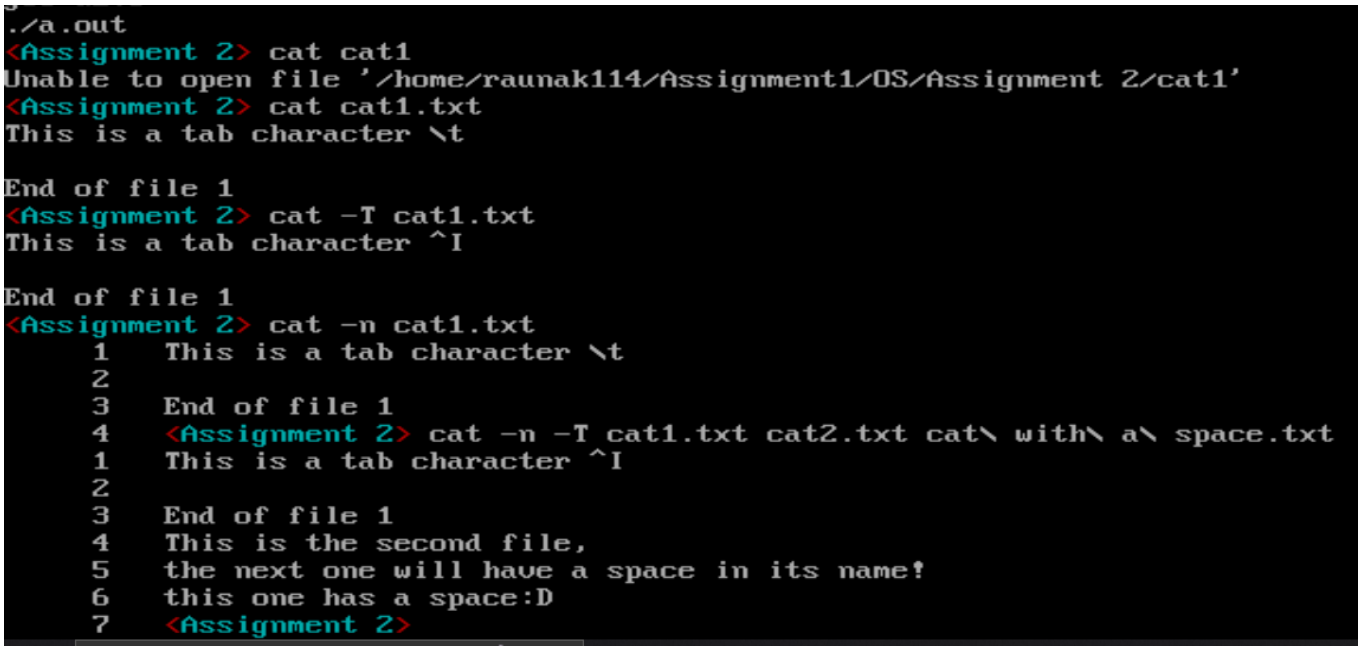
- mkdir / mkdir&t :
    - Flags:
        1) **-p:** Makes all required parent directories, and forces the creation of given directory even if the path currently doesn't exist, it'll make every folder required along the way.
        2) **-v**: Displays a success message
    - Edge Cases:
        a) Supports space separated folder creation, if a user enters spaces, it is simply concatenated.
        b) If an empty input is given, it throws an error prompt and exits back into the shell loop.



- date /date&t:
    - Flags:
        1) **-u:** Prints out date in UTC format
        2) **-r:** Prints out date in RFC5322 format
    - Edge Cases:
        a) Handles invalid date inputs and throws an error message if it fails
        b) Gives the user feedback as to what part of the date command was invalid, ‘Seg = <invalid part>‘

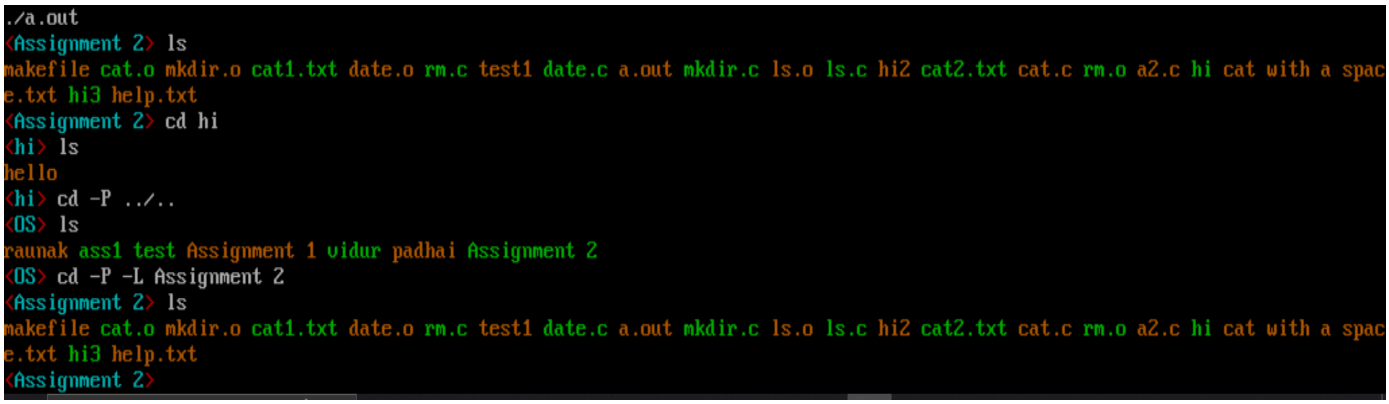


- cat / cat&t:
    - Flags:
        1) **-T**: Whenever the program encounters a '\t', it outputs '^I'
        2) **-n:** Displays the line number of the output
    - Edge Cases:
        a) Since this is the only command that allows multiple inputs, for it to support files containing spaces, there is a provision for the user, that is, to use '\' before the space bar, so hi\ hi2\ .txt would represent 'hi hi2 .txt'.
        b) Multiple Flags are supported
        c) '\' is handled like echo

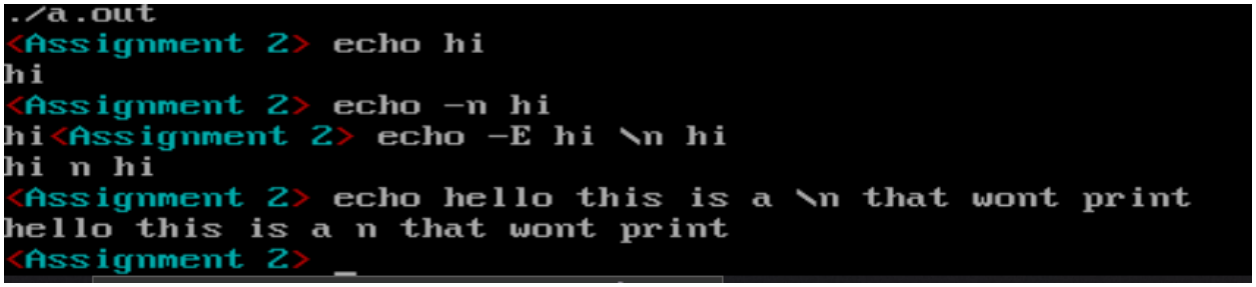


Internal Commands:

- Cd:
    - Flags:
        1) **-L:** Force symbolic links to be followed
        2) **-P:** Use the physical directory structure
    - Edge Cases:
        a) Supports directories with spaces directly with no special clauses
        b) '\' is handled like in echo

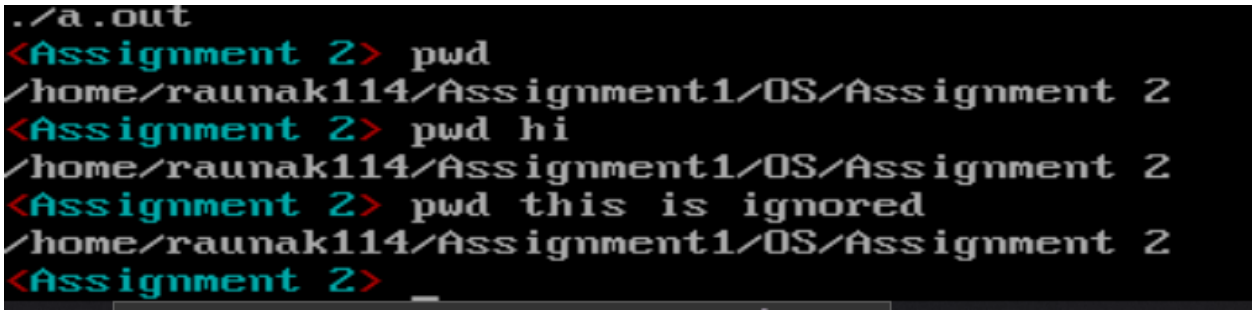


- Echo:
    - Flags:
        1) **-n:** Do not append a new line after the message
        2) **-E:** Explicitly suppress escape sequence characters
    - Edge Cases:
        a) If '\' is used, meaning a string with a '\' is entered, the \ prints whatever comes after it, and disappears itself. This handles escape sequences as well
        b) If $$ is input, it prints out the process ID of the shell



PWD: Prints the current working directory
Both available flags are irrelevant for the program as the shell automatically handles -L automatically

Edge case: If the user inputs any other arguments in pwd, it ignores it, as it is of no use or consequence



Exit: Terminates the shell