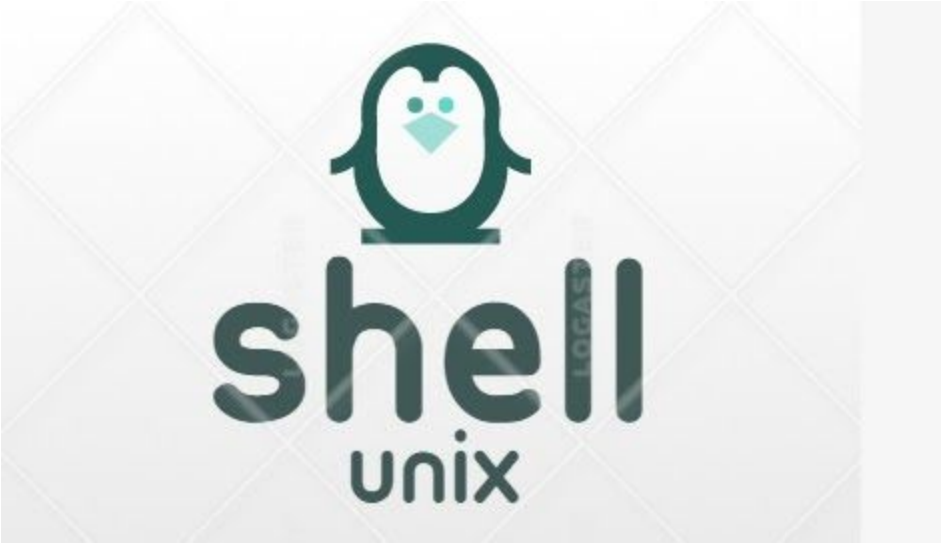
Operating Systems

Simple Shell

By: Hossam Elkordi n: 24



Problem Statement:

It is required to implement a Unix shell program that executes the internal shell command (exit) and shell commands with and without arguments in addition to the execution of background processes using (&) in the end of the command.

Major Functions:

```
void newCommand();
void getCommand(char input[]);
int split(char command[1000], char* list[], int* background);
void normalExec(char* list[]);
void backExec(char* list[]);
void sigHandler(int signal);
void appendFile();
```

getCommand(char input[]):

Takes user's input and stores it in the (input) array.

split(char command[1000], char* list[], int* background):

Splits the command entered by the user around a single wight space in the (list) array and adds a (NULL) element at the end. Also, this function checks if the command ends with (&) or not to determine whether the user wants the process to be executed in the background or not and sets the integer value (background) by 1 if it is a background process and sets it by 0 otherwise. This function returns the number of words in the (list) array (i.e index of the first NULL element).

```c
int split(char command[1000], char* list[], int* background){
    int listLen = 0;
    char* delim = " ";
    char* token = strtok(command, delim);
    while (token != NULL){
        list[listLen++] = token;
        token = strtok( s: NULL, delim);
    }

    if (strcmp(list[listLen - 1], "&") == 0){
        list[--listLen] = NULL;
        (*background) = 1;
    } else if (list[listLen - 1][strlen( s: list[listLen - 1]) - 1] == '&'){
        list[listLen - 1][strlen( s: list[listLen - 1]) - 1] = '\0';
        (*background) = 1;
        list[listLen] = NULL;
    }else {
        list[listLen] = NULL;
    }
    return listLen;
}
```

normalExec(char* list[]) / backExec(char* list[]):

Both functions fork a new process and check for the id returned.

```c
void normalExec(char* list[]){
    int id = fork();
    if (id == 0){
        if(execvp( file: list[0], list) == -1){
            perror( s: "");
        }
    } else if (id == -1){
        printf( format: "Error occurred while forking.\n");
    }else if (id > 0){
```

The difference between them is the way the parent process works.

- ## normalExec:

The parent process waits the child to terminates:

```
}else if (id > 0){
    // Parent -> Wait for its children
    int status;
    wait( stat_loc: &status);
    appendFile();
    if (WIFEXITED(status)){
        int statcode = WEXITSTATUS(status);
        if(statcode != 0){
            perror( s: "No Child.\n");
        }
    }
}
```

- ## backExec:

The parent doesn't wait for the child to terminate. But when the child terminates it gets notified by SIGCHLD signal by the mean of the sigaction struct in (signal.h) header file and the sigHandler function.

```
struct sigaction saction;
saction.sa_handler = sigHandler;
saction.sa_flags = SA_RESTART;
```

```
}else{
    sigaction(SIGCHLD, &saction, oact: NULL);
}
```

sigHandler(int signal):

This function gets notified when a background process is terminated by using the (waitpid) function and the (WNOHANG -> wait no hang) signal.

```
void sigHandler(int signal){
    int status ,id = waitpid( pid: -1, &status, WNOHANG);
    if (id > 0){
        appendFile();
    }
}
```

appendFile():

This function gets called whenever a child process is terminated and writes a message in a file called (logFile) that exists in the project folder.

```
void appendFile(){
    FILE *log_file = fopen( filename: "logFile", modes: "a");
    fprintf(log_file,  format: "Child process was terminated.\n");
    fclose(log_file);
}
```

main():

The program starts here by initializing the needed variables and flags and sets them to the default values. Then it takes input from the user and splits it then checks it is an internal command from (exit or cd) to execute them. If they command wasn't internal it calls normalExec or backExec depending on the (background) flag.

```c
int main() {
    char command[1000];
    char* list[500];
    int listLen = 0, background = 0;
    FILE *log_file = fopen( filename: "logFile", modes: "w");
    printf( format: "Simple Shell created by Hossam Elkordi.\n\n");

    while (1){
        memset(command, c: NULL, sizeof(command));
        memset(list, c: NULL, sizeof(list));
        getCommand(command);
        listLen = split(command, list, &background);
        if (listLen == 1 && strcmp(list[0], "exit") == 0){
            exit( status: 0);
        } else if (listLen == 2 && strcmp(list[0], "cd") == 0){
            chdir( path: list[1]);
        } else {
            if(background == 1){
                background = 0;
                backExec(list);
            } else {
                normalExec(list);
            }
        }
    }
    return 0;
}
```

Sample Runs:

```
hossam@Geek:~/CLionProjects/SimpleShell$ ./main
Simple Shell created by Hossam Elkordi.

hossam:~$ pwd
/home/hossam/CLionProjects/SimpleShell
hossam:~$ ls -l
total 28
drwxrwxr-x 4 hossam hossam  4096 Nov  7 16:14 cmake-build-debug
-rw-rw-r-- 1 hossam hossam   121 Nov  1 22:12 CMakeLists.txt
-rwxrwxr-x 1 hossam hossam 14120 Nov  7 16:00 main
-rw-rw-r-- 1 hossam hossam  3446 Nov  7 16:14 main.c
hossam:~$ cd /home/hossam
hossam:~$ ls
CLionProjects  Documents  examples.desktop  Pictures  SimpleShellLog  Videos
Desktop        Downloads  Music             Public    Templates
hossam:~$ mkdir file
hossam:~$ ls
CLionProjects  Documents  examples.desktop  Music     Public          Templates
Desktop        Downloads  file              Pictures  SimpleShellLog  Videos
hossam:~$ mv file newFile
hossam:~$ ls
CLionProjects  Documents  examples.desktop  newFile   Public          Templates
Desktop        Downloads  Music             Pictures  SimpleShellLog  Videos
```

```
hossam:~$ cd newFile
hossam:~$ ls
file
hossam:~$ rm file
hossam:~$ ls
hossam:~$ cd ..
hossam:~$ touch file
hossam:~$ ls
CLionProjects  Downloads         main     Pictures        Templates
Desktop        examples.desktop  Music    Public          Videos
Documents      file              newFile  SimpleShellLog
hossam:~$ cp file newFile/file
hossam:~$ cd newFile
hossam:~$ ls
file
hossam:~$ mv file newFileName
hossam:~$ ls
newFileName
hossam:~$ rm newFileName
hossam:~$ ls
hossam:~$ cd ..
hossam:~$ ls
CLionProjects  Downloads         main     Pictures        Templates
Desktop        examples.desktop  Music    Public          Videos
Documents      file              newFile  SimpleShellLog
hossam:~$ rm -R newFile
hossam:~$ ls
CLionProjects  Downloads         main      Public          Videos
Desktop        examples.desktop  Music     SimpleShellLog
Documents      file              Pictures  Templates
hossam:~$ rm file
hossam:~$ ls
CLionProjects  Documents  examples.desktop  Music     Public          Templates
Desktop        Downloads  main              Pictures  SimpleShellLog  Videos
hossam:~$ exit
```

KSysguard screenshots:

Opening (Zoom) from the shell: