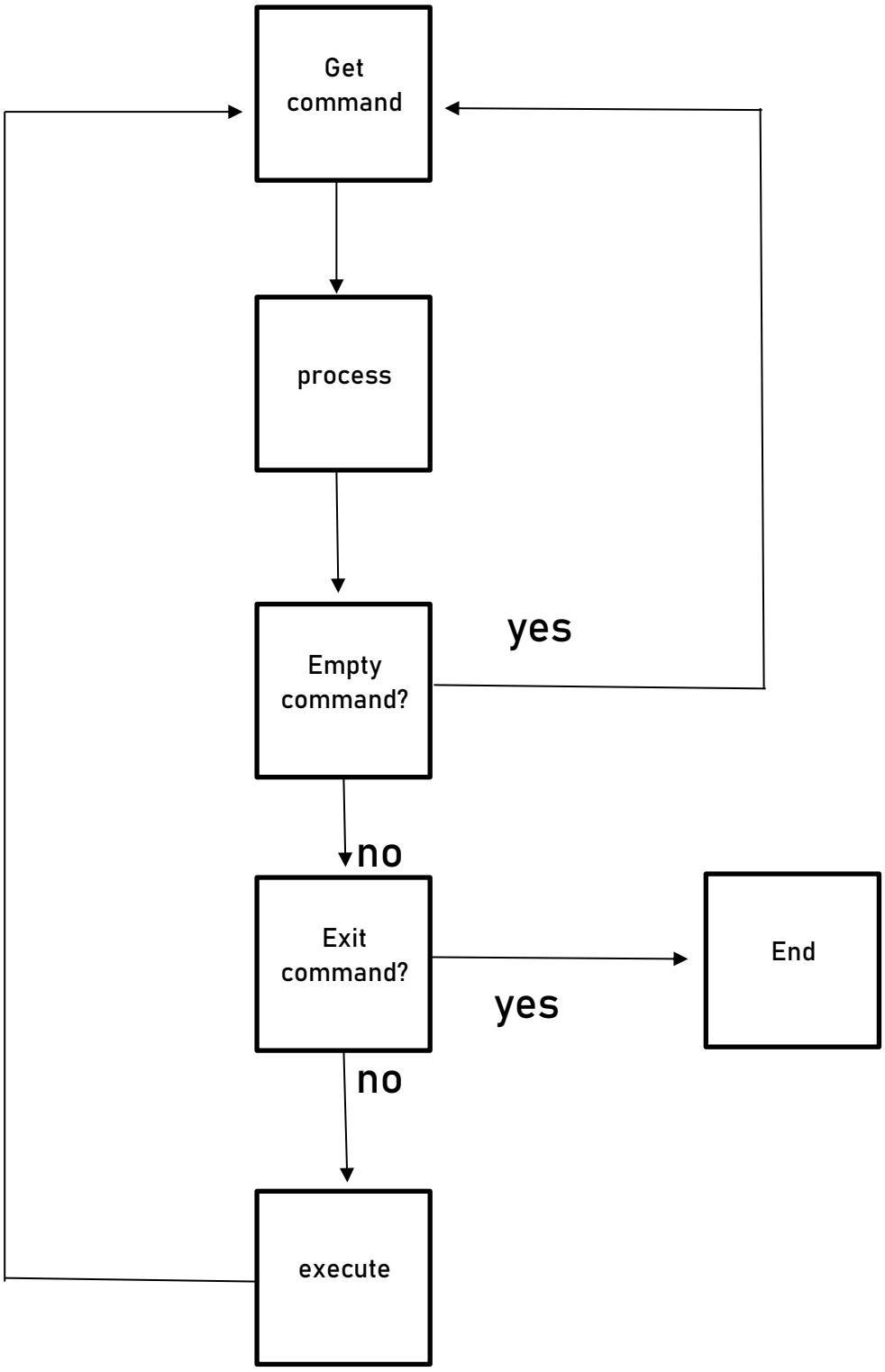
Simple Linux Shell

محمد طارق وجدي ١٨٠١١٥٠٨

محمد تهامي محمد ١١ (لائحة قديمة)

تغريد طه إبراهيم ١٨٠١٠٤٩٩

It's required to simulate linux shell. We must take the command and it's arguments (if found) and process them then execute it.



Run samples:

```
>>>ls
builtInFunctions.c  log.txt  main.c   shell.h
builtInFunctions.h  main     shell.c
>>>pwd
/home/taghreed/os/linux shell
>>>ls
builtInFunctions.c  log.txt  main.c   shell.h
builtInFunctions.h  main     shell.c
>>>ls -a
.   builtInFunctions.c  log.txt  main.c   shell.h
..  builtInFunctions.h  main     shell.c
>>>ls -1
builtInFunctions.c
builtInFunctions.h
log.txt
main
main.c
shell.c
shell.h
>>>
>>>
>>>sleep 3
>>>
>>>
```

Here, we trying commands without argument (ls, pwd), commands with arguments (ls -a, ls -1, sleep 3,) and empty commands.

```
>>>lss
Error executing...
: No such file or directory
>>>exit
taghreed@taghreed-VirtualBox:~/os/linux shell$
```

Here, trying wrong command and exit command.

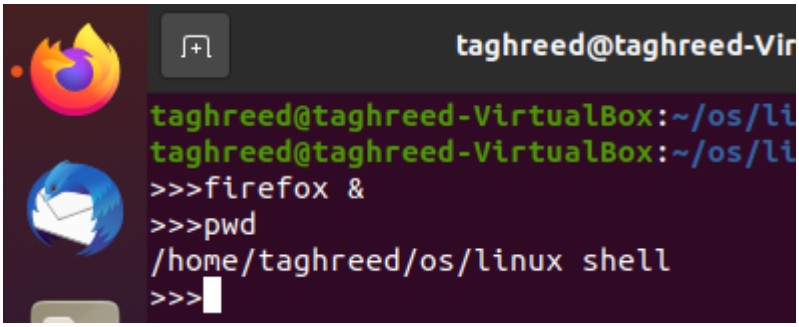
```
>>>ls
builtInFunctions.c  builtInFunctions.h  main  main.c  shell.c  shell.h
>>>ls
builtInFunctions.c  log.txt  main.c   shell.h
builtInFunctions.h  main     shell.c
>>>pwd
/home/taghreed/os/linux shell
>>>ls -a
.   builtInFunctions.c  log.txt  main.c   shell.h
..  builtInFunctions.h  main     shell.c
>>>exit
taghreed@taghreed-VirtualBox:~/os/linux shell$
```

```
child proccess terminated.
child proccess terminated.
child proccess terminated.
child proccess terminated.
```

Here, showing logging file.

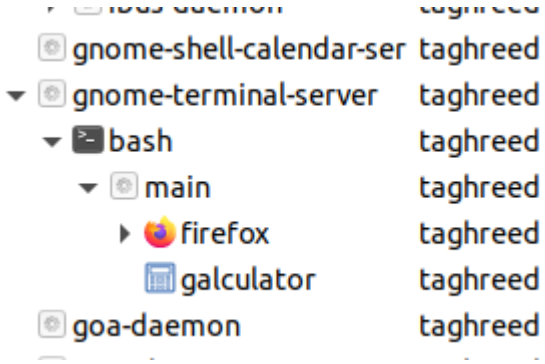
```
>>>firefox
ls
pwd
ls -a
>>>builtInFunctions.c  log.txt  main.c   shell.h
builtInFunctions.h  main     shell.c
>>>/home/taghreed/os/linux shell
>>>.   builtInFunctions.c       log.txt  main.c   shell.h
..  builtInFunctions.h  main     shell.c
>>>
```

Without (&) new commands cannot execute until we closed firefox.



With (&) new commands executed until we still using firefox.

```
>>>firefox &
>>>galculator &
>>>
```



Firefox and calculator are child processes to the main process.

Full code:

main.c file:

```c
#include "shell.h"

int main(){
    signal(SIGCHLD, LogHandle); //attach signal with logging file
    getJobDone();
    return 0;
}
```

Shell.h file:

```c
#ifndef _SHELL_C_INCLUDED_
#include "builtInFunctions.h"
#include <stdlib.h>
#include <stdio.h>
#include <unistd.h>
#include <string.h>
#include <stdbool.h>
#include <sys/wait.h>
#include <signal.h>

#define TOKEN_BUFFER_SIZE 64
#define TOKEN_DELIMTER " \t\r\n\a"

void getJobDone();
char* readLine();
char** readArgs(char* line);
bool launchShell(char** args);
bool executeShell(char** args);
void LogHandle(int sig);

#include "shell.c"
#define  _SHELL_C_INCLUDED_
#endif
```

Shell.c file:

```c
void getJobDone(){
    char *line;
    char **args;
    bool status;

    do {
        printf(">>>");
        line = readLine();  //get command and arguments
        args = readArgs(line);   //put it in suitable form for execution
        status = executeShell(args);  //execute command

        free(line);
        free(args);
    } while(status);  //repeat till exit
}
```

```c
char* readLine(){
    //get command from user
    char *line = NULL;
    size_t buffersize = 0;

    if(getline(&line, &buffersize, stdin) == -1){
        // checking for error in getting the command
        if(feof(stdin)){
            exit(EXIT_SUCCESS);
        }else{
            perror("Error in Reading Line...\n");
            exit(EXIT_FAILURE);
        }
    }
    return line;
}
```

```c
char** readArgs(char* line){
    // putting input in the form of [command , arguments , NULL]
    int buffersize = TOKEN_BUFFER_SIZE;
    int position = 0;
    char **tokens = malloc(buffersize*sizeof(char*));
    char *token;

    if(!token){
        fprintf(stderr, "Allocate Error\n");
        exit(EXIT_FAILURE);}

    token = strtok(line, TOKEN_DELIMTER);
    while(token != NULL){
        tokens[position] = token;
        ++position;
        if(position >= buffersize){//check if we need more space
        buffersize += TOKEN_BUFFER_SIZE;
        tokens = realloc(tokens, buffersize*sizeof(char*));
            if(!tokens){
                fprintf(stderr, "Allocate Error...\n");
                exit(EXIT_FAILURE);}}
        token = strtok(NULL, TOKEN_DELIMTER);}
    tokens[position] = NULL;
    return tokens;
}

bool executeShell(char** args){
    if(args[0] == NULL)//if empty command start again and get new command
        return true;
    //check for (cd,help,exit) and execute it
    for(int i=0; i<getBuiltInNums(); ++i){
        if(strcmp(args[0], builtInChar[i]) == 0)
            return (*builtInFunctions[i])(args);
    }

    return launchShell(args);
}
```

```
bool launchShell(char** args){
    pid_t pid, wpid;
    int status, i = 0, toWait = true;
    const char* AND_OPERATOR = "&";
    // check for & in the command
    while(args[i] != NULL){
        if(strcmp(args[i], AND_OPERATOR) == 0){
            args[i] = NULL;
            toWait = false;}
        ++i;}

    pid = fork();
    if(pid == 0){
        if(execvp(args[0], args) == -1){
            perror("Error executing...\n");
            exit(EXIT_FAILURE);}}
    else if(pid < 0){perror("Error in processes...\n");}
    else{
        if(toWait){
            do{
                wpid = waitpid(pid, &status, WUNTRACED);
            } while(!WIFEXITED(status) && !WIFSIGNALED(status));}}

    return true;
}

void LogHandle(int sig){
    //logging file
    FILE *pFile;
    pFile = fopen("log.txt", "a+");
    if(pFile==NULL) perror("Error opening file.");
    else fprintf(pFile, "child proccess terminated.\n");
    fclose(pFile);
}
```

builtInFunctions.h file:

```
#ifndef _BUILTINFUNCTIONS_C_INCLUDED_

#include <stdio.h>
#include <unistd.h>
#include <stdbool.h>

bool checkCD(char** args);
bool checkHelp(char** args);
bool checkExit(char** args);
int getBuiltInNums();

char *builtInChar[] = {
  "cd",
  "help",
  "exit"
};
bool (*builtInFunctions[]) (char **) = {
  &checkCD,
  &checkHelp,
  &checkExit
};

#include "builtInFunctions.c"
#define _BUILTINFUNCTIONS_C_INCLUDED_
#endif
```

builtInFunctions.c file:

```
bool checkCD(char** args){
    //dealing with cd
    if(args[1] == NULL)
        fprintf(stderr, "Expected command after cd...\n");
    else{
        if(chdir(args[1]) != 0)
            perror("Error in Changing Directory...\n");
    }

    return true;
}

bool checkHelp(char** args){//dealing with help
    printf("***welcom to help***\nlist of commands supported:
\n>cd\n>ls\n>exit\n>all other general commands available in UNIX
shell\n>improper space handling\n");
    return true;
}

bool checkExit(char** args){//dealing with exit
    return false;
}

int getBuiltInNums(){
    return sizeof(builtInChar) / sizeof(char*);
}
```