using execve system call + 1 arg like pwd + must take tool path

use by
`I am a simple shell, enter your command $ /usr/bin/ls`

```
#include <stdio.h>
#include <stdlib.h>
#include <sys/wait.h>
#include <unistd.h>
#include <string.h>

#define BUF_SIZE 100
int main()
{
    char buf[BUF_SIZE];

    while(1) {
    printf("I am a simple shell, enter your command $ ");
    fgets(buf, BUF_SIZE, stdin);
    buf[strlen(buf) - 1] = 0;
    if (strlen(buf) == 0)
        continue;


    pid_t pid = fork();
    if (pid > 0) {
        int status;
        printf("PARENT: my pid = %d, my child pid = %d\n", getpid(),
           pid);
        wait(&status);
        printf("PARENT: my pid = %d, my child status = %d\n", getpid(),
           WEXITSTATUS(status));
    } else if (pid == 0) {
        printf("CHILD: my pid = %d, my parent pid = %d\n", getpid(),
           getppid());
        char *newargv[] = { buf, NULL };
        char *newenvp[] = { NULL };
        execve(buf, newargv, newenvp);
        printf
        ("Exec failed, kernel is not the mode of executing programs\n");
        exit(-1);
    } else {
        printf("PARENT: failed to fork\n");
    }
    }
    return 0;
}
```

using wrapper execlp here 1 arg but take only tool name not path and it searches for u

`Enter command to run: ls`

```c
#include <string.h>
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/wait.h>

int main() {
    char buf[256];
    while(1){
        printf("Enter command to run: ");
    if (fgets(buf, sizeof(buf), stdin) == NULL) {
           perror("fgets failed");
       return -1;
    }
    // Remove newline
    buf[strlen(buf) - 1] = 0;
        if (strlen(buf) == 0)
        continue;
    pid_t pid = fork();

    if (pid > 0) {
        // Parent process
        int status;
        wait(&status);  // wait for child
        printf("PARENT: my pid = %d, my child status = %d\n",
               getpid(), WEXITSTATUS(status));

    } else if (pid == 0) {
        // Child process
        printf("CHILD: my pid = %d, my parent pid = %d\n", getpid(),
getppid());

        // Use execlp: first arg = file to run, followed by argv list, ending
with NULL
        execlp(buf, buf, (char *)NULL);
        // Only reached if exec fails
        perror("Exec failed");
        exit(-1);
    } else {
        // Fork failed
        perror("PARENT: failed to fork");
        return -1;
    }
    }
    return 0;
}
```