Everlyn Leon

Abstract

Creating a bash shell in C using fork(), wait(), and execvp(), as well as having all the basic techniques such as accepting input and running system commands.

Introduction

On a Ubuntu Linux Virtual machine, a bash shell using C will be created to demonstrate the usage of fork(), wait(), and execvp(), in addition to common commands such as: ls -la, touch, cat, ps, top, nano, sudo, exit.

Summary of Results

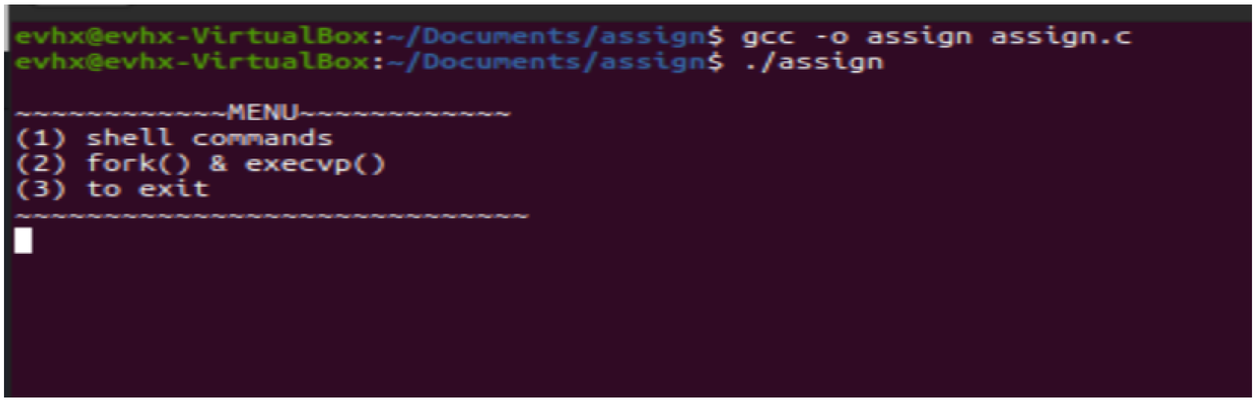
To execute the bash shell, we first need to use the GCC, which is a set of compilers and tools on Linux. To compile the file named 'assign.c', execute the command:

```
gcc -o assign assign.c
```

Then to run the executable file, use the command:

```
./assign
```

The bash shell will now prompt the user to make a selection.



```
evhx@evhx-VirtualBox:~/Documents/assign$ gcc -o assign assign.c
evhx@evhx-VirtualBox:~/Documents/assign$ ./assign

~~~~~~~~~~~~MENU~~~~~~~~~~~~
(1) shell commands
(2) fork() & execvp()
(3) to exit
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
```

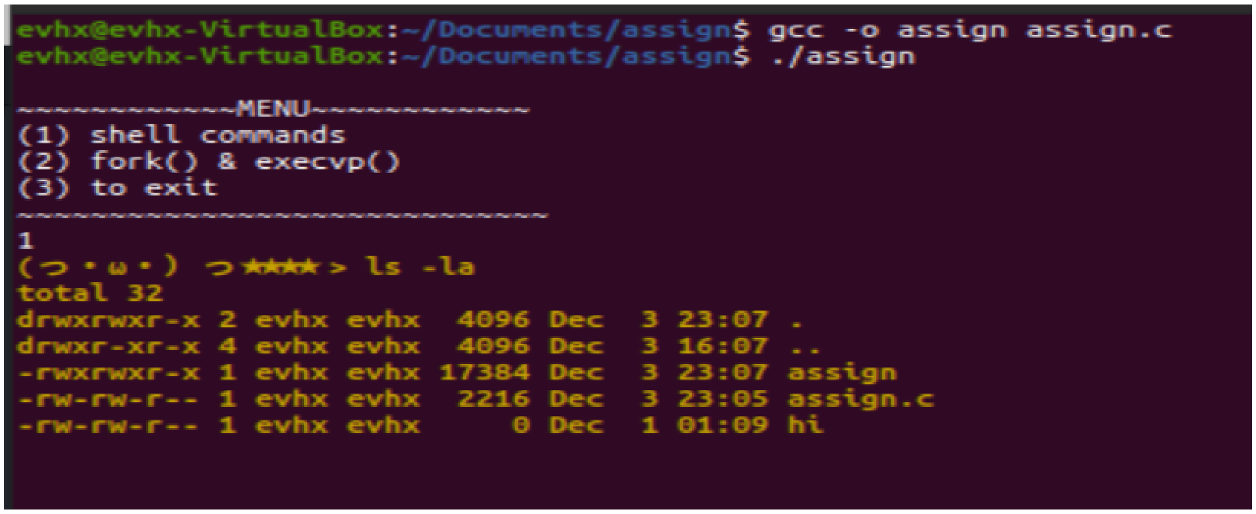
Now let’s begin testing the bash shell with the command:

ls -la

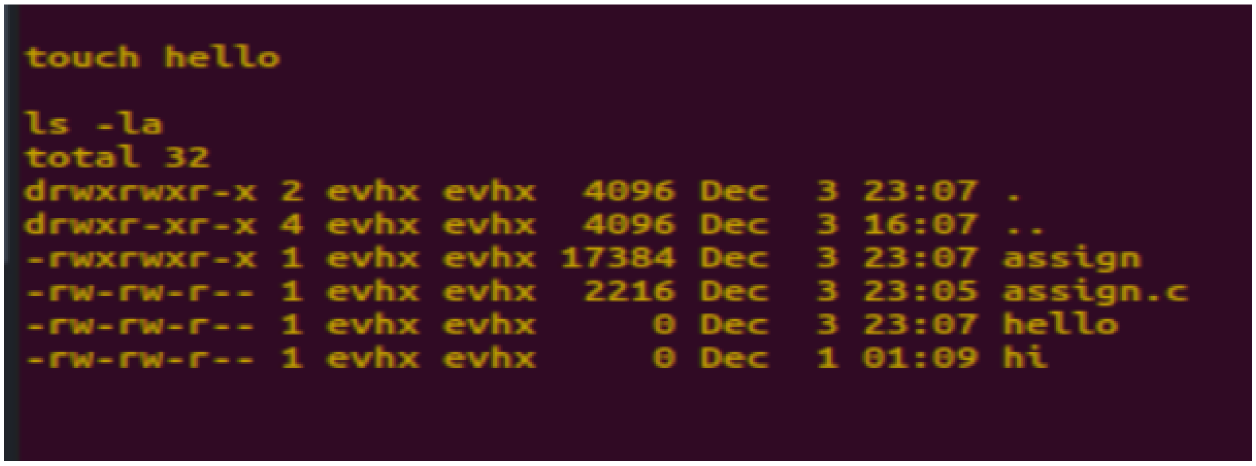
Which lists the files in the directory. Then let's test the touch command:

touch hello

Using the ‘ls -la’ command, you can see that the touch function created a text file named ‘hello’.



```
evhx@evhx-VirtualBox:~/Documents/assign$ gcc -o assign assign.c
evhx@evhx-VirtualBox:~/Documents/assign$ ./assign

~~~~~~~~~~~~MENU~~~~~~~~~~~~
(1) shell commands
(2) fork() & execvp()
(3) to exit
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
1
(っ•ω•) っ★★★★> ls -la
total 32
drwxrwxr-x 2 evhx evhx  4096 Dec  3 23:07 .
drwxr-xr-x 4 evhx evhx  4096 Dec  3 16:07 ..
-rwxrwxr-x 1 evhx evhx 17384 Dec  3 23:07 assign
-rw-rw-r-- 1 evhx evhx  2216 Dec  3 23:05 assign.c
-rw-rw-r-- 1 evhx evhx     0 Dec  1 01:09 hi
```



```
touch hello

ls -la
total 32
drwxrwxr-x 2 evhx evhx  4096 Dec  3 23:07 .
drwxr-xr-x 4 evhx evhx  4096 Dec  3 16:07 ..
-rwxrwxr-x 1 evhx evhx 17384 Dec  3 23:07 assign
-rw-rw-r-- 1 evhx evhx  2216 Dec  3 23:05 assign.c
-rw-rw-r-- 1 evhx evhx     0 Dec  3 23:07 hello
-rw-rw-r-- 1 evhx evhx     0 Dec  1 01:09 hi
```

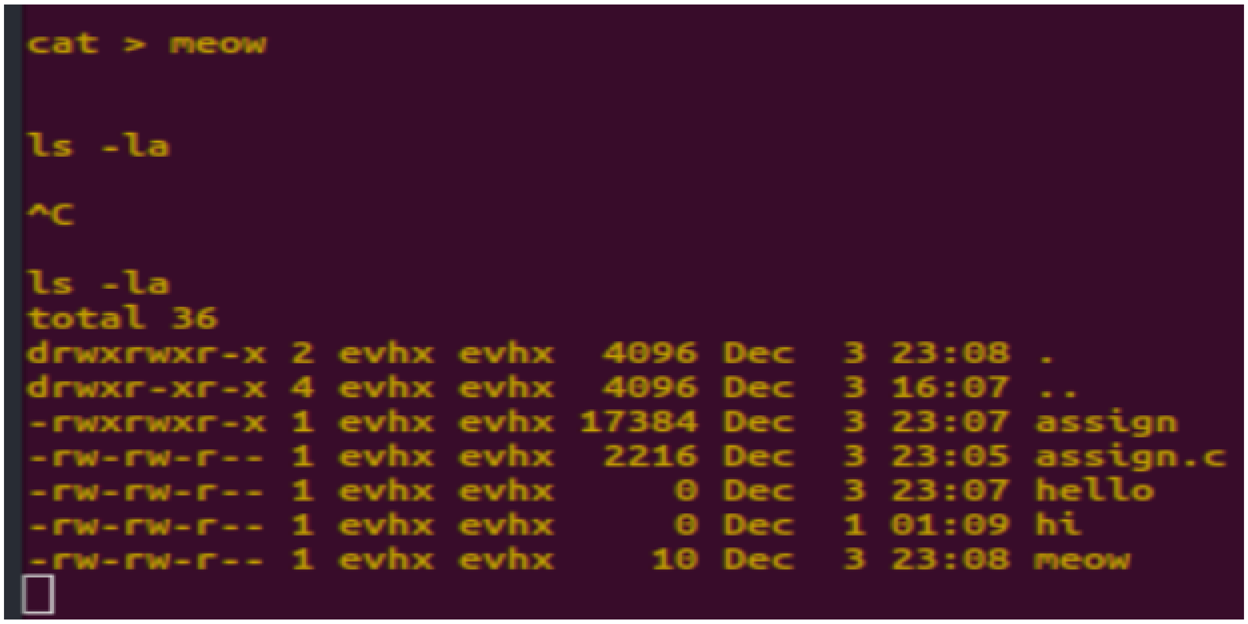
To concatenate into a file, use the command:

cat > [file name]

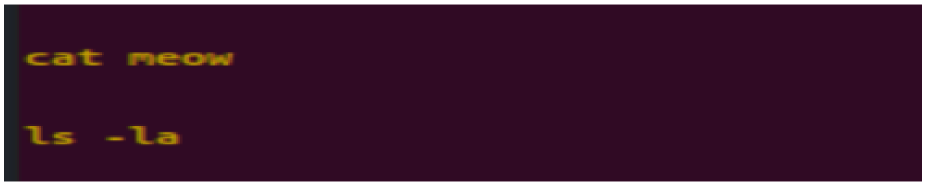
I accidentally typed ‘ls -la’ in the text file, which will then be located in the file called ‘meow’. To exit out of the concatenation, use ‘ctrl+C’. Using the ‘ls -la’ command, you can then see the file ‘meow’ was created.

To view the processes, use the command:

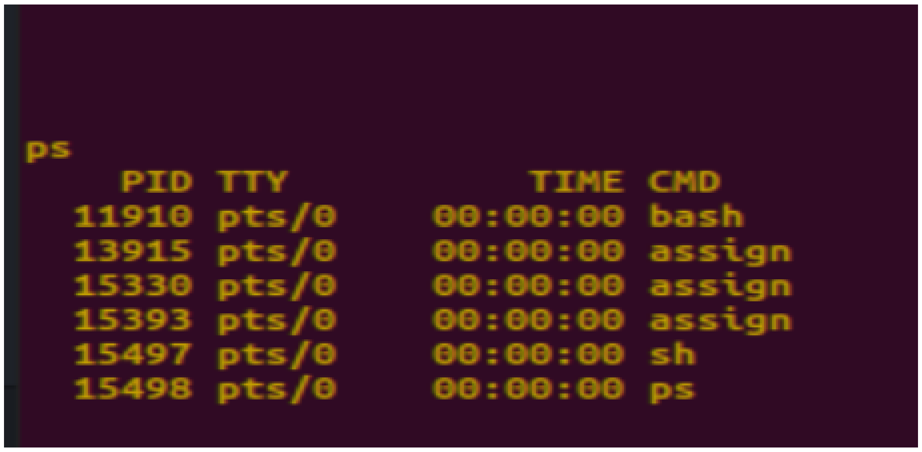
ps



```
cat > meow


ls -la

^C

ls -la
total 36
drwxrwxr-x 2 evhx evhx  4096 Dec  3 23:08 .
drwxr-xr-x 4 evhx evhx  4096 Dec  3 16:07 ..
-rwxrwxr-x 1 evhx evhx 17384 Dec  3 23:07 assign
-rw-rw-r-- 1 evhx evhx  2216 Dec  3 23:05 assign.c
-rw-rw-r-- 1 evhx evhx     0 Dec  3 23:07 hello
-rw-rw-r-- 1 evhx evhx     0 Dec  1 01:09 hi
-rw-rw-r-- 1 evhx evhx    10 Dec  3 23:08 meow
```



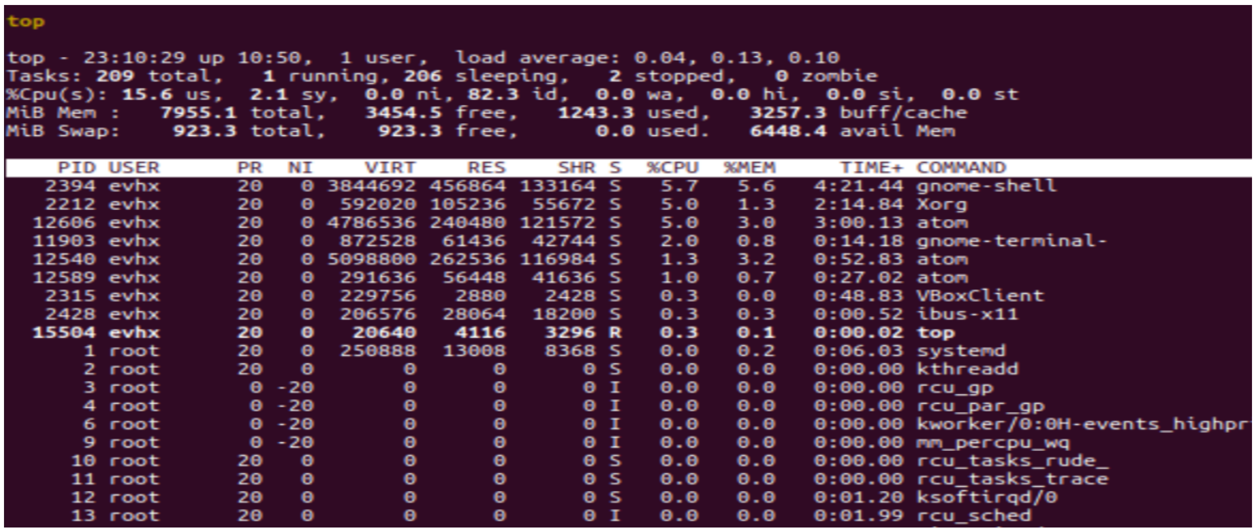
```
cat meow

ls -la
```



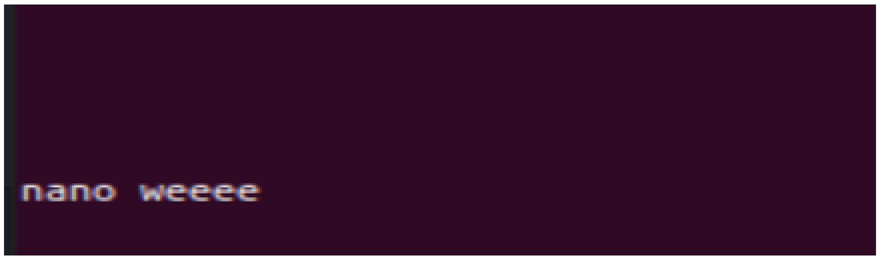
```
ps
    PID TTY          TIME CMD
  11910 pts/0    00:00:00 bash
  13915 pts/0    00:00:00 assign
  15330 pts/0    00:00:00 assign
  15393 pts/0    00:00:00 assign
  15497 pts/0    00:00:00 sh
  15498 pts/0    00:00:00 ps
```

To view CPU information and more details about the processes running, use the command:
top

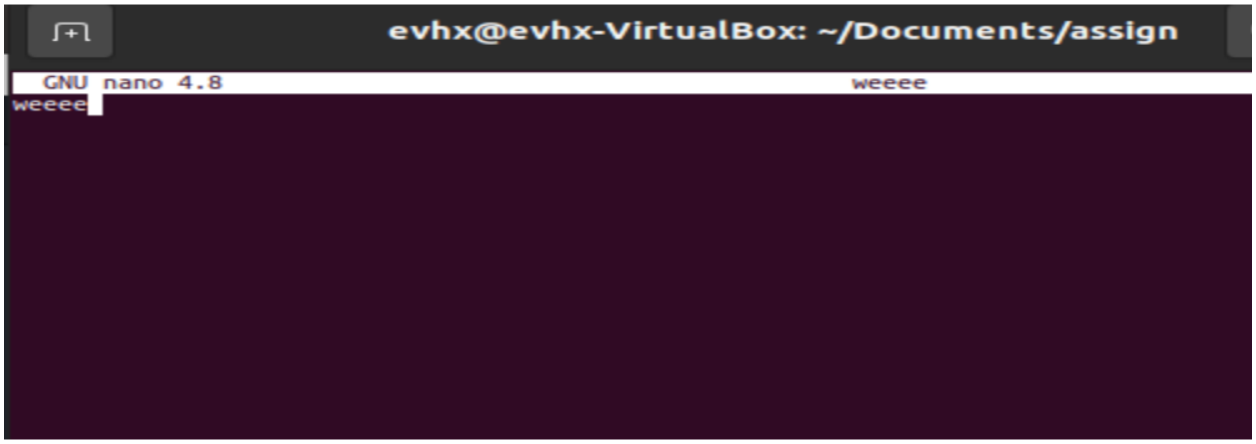
To write text into a text document using a text editor, use the command:
nano



```
top

top - 23:10:29 up 10:50,  1 user,  load average: 0.04, 0.13, 0.10
Tasks: 209 total,   1 running, 206 sleeping,   2 stopped,   0 zombie
%Cpu(s): 15.6 us,  2.1 sy,  0.0 ni, 82.3 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
MiB Mem :   7955.1 total,   3454.5 free,   1243.3 used,   3257.3 buff/cache
MiB Swap:    923.3 total,    923.3 free,      0.0 used.   6448.4 avail Mem

    PID USER      PR  NI    VIRT    RES    SHR S  %CPU  %MEM     TIME+ COMMAND
   2394 evhx      20   0 3844692 456864 133164 S   5.7   5.6   4:21.44 gnome-shell
   2212 evhx      20   0  592020 105236  55672 S   5.0   1.3   2:14.84 Xorg
  12606 evhx      20   0 4786536 240480 121572 S   5.0   3.0   3:00.13 atom
  11903 evhx      20   0  872528  61436  42744 S   2.0   0.8   0:14.18 gnome-terminal-
  12540 evhx      20   0 5098800 262536 116984 S   1.3   3.2   0:52.83 atom
  12589 evhx      20   0  291636  56448  41636 S   1.0   0.7   0:27.02 atom
   2315 evhx      20   0  229756   2880   2428 S   0.3   0.0   0:48.83 VBoxClient
   2428 evhx      20   0  206576  28064  18200 S   0.3   0.3   0:00.52 ibus-x11
  15504 evhx      20   0   20640   4116   3296 R   0.3   0.1   0:00.02 top
      1 root      20   0  250888  13008   8368 S   0.0   0.2   0:06.03 systemd
      2 root      20   0       0      0      0 S   0.0   0.0   0:00.00 kthreadd
      3 root       0 -20       0      0      0 I   0.0   0.0   0:00.00 rcu_gp
      4 root       0 -20       0      0      0 I   0.0   0.0   0:00.00 rcu_par_gp
      6 root       0 -20       0      0      0 I   0.0   0.0   0:00.00 kworker/0:0H-events_highpr
      9 root       0 -20       0      0      0 I   0.0   0.0   0:00.00 mm_percpu_wq
     10 root      20   0       0      0      0 S   0.0   0.0   0:00.00 rcu_tasks_rude_
     11 root      20   0       0      0      0 S   0.0   0.0   0:00.00 rcu_tasks_trace
     12 root      20   0       0      0      0 S   0.0   0.0   0:01.20 ksoftirqd/0
     13 root      20   0       0      0      0 I   0.0   0.0   0:01.99 rcu_sched
```



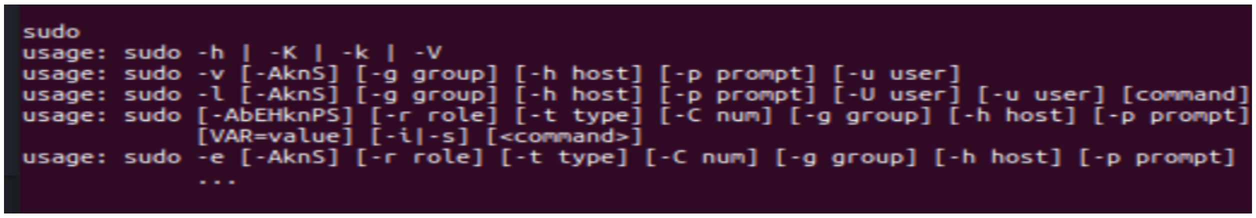
```
nano weeee
```



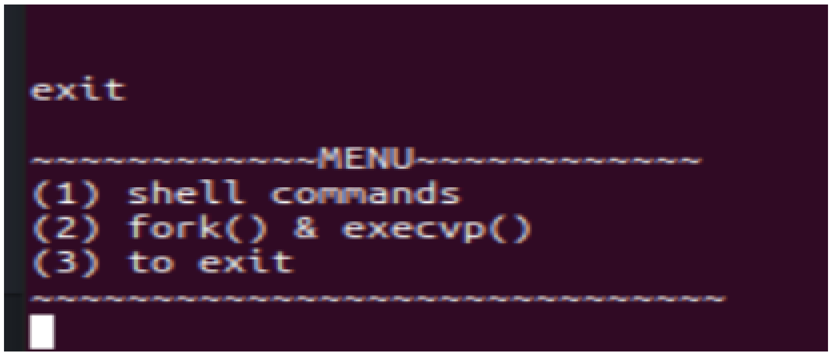
```
evhx@evhx-VirtualBox: ~/Documents/assign
  GNU nano 4.8                                   weeee
weeee
```

To see the list of available usages with the command ‘sudo’, just execute:
sudo

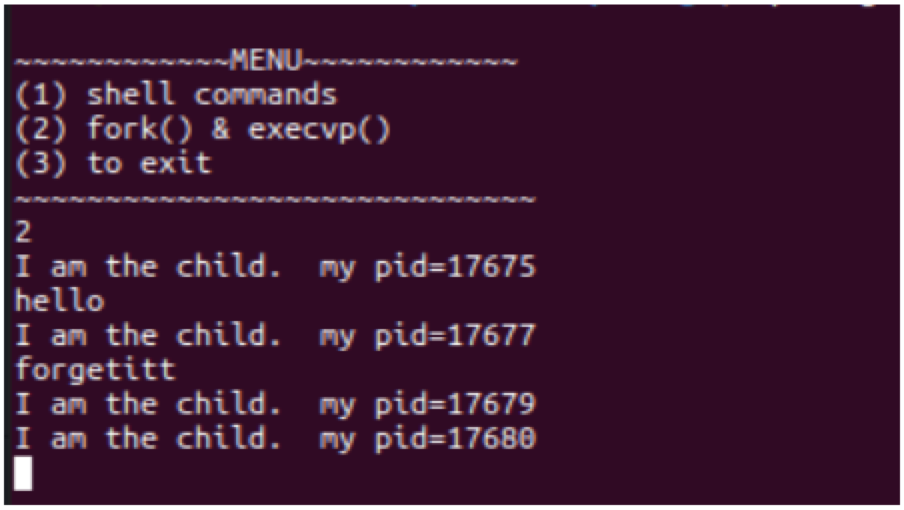
To exit from using the shell commands simply type:
exit.



```
sudo
usage: sudo -h | -K | -k | -V
usage: sudo -v [-AknS] [-g group] [-h host] [-p prompt] [-u user]
usage: sudo -l [-AknS] [-g group] [-h host] [-p prompt] [-U user] [-u user] [command]
usage: sudo [-AbEHknPS] [-r role] [-t type] [-C num] [-g group] [-h host] [-p prompt]
            [VAR=value] [-i|-s] [<command>]
usage: sudo -e [-AknS] [-r role] [-t type] [-C num] [-g group] [-h host] [-p prompt]
            ...
```

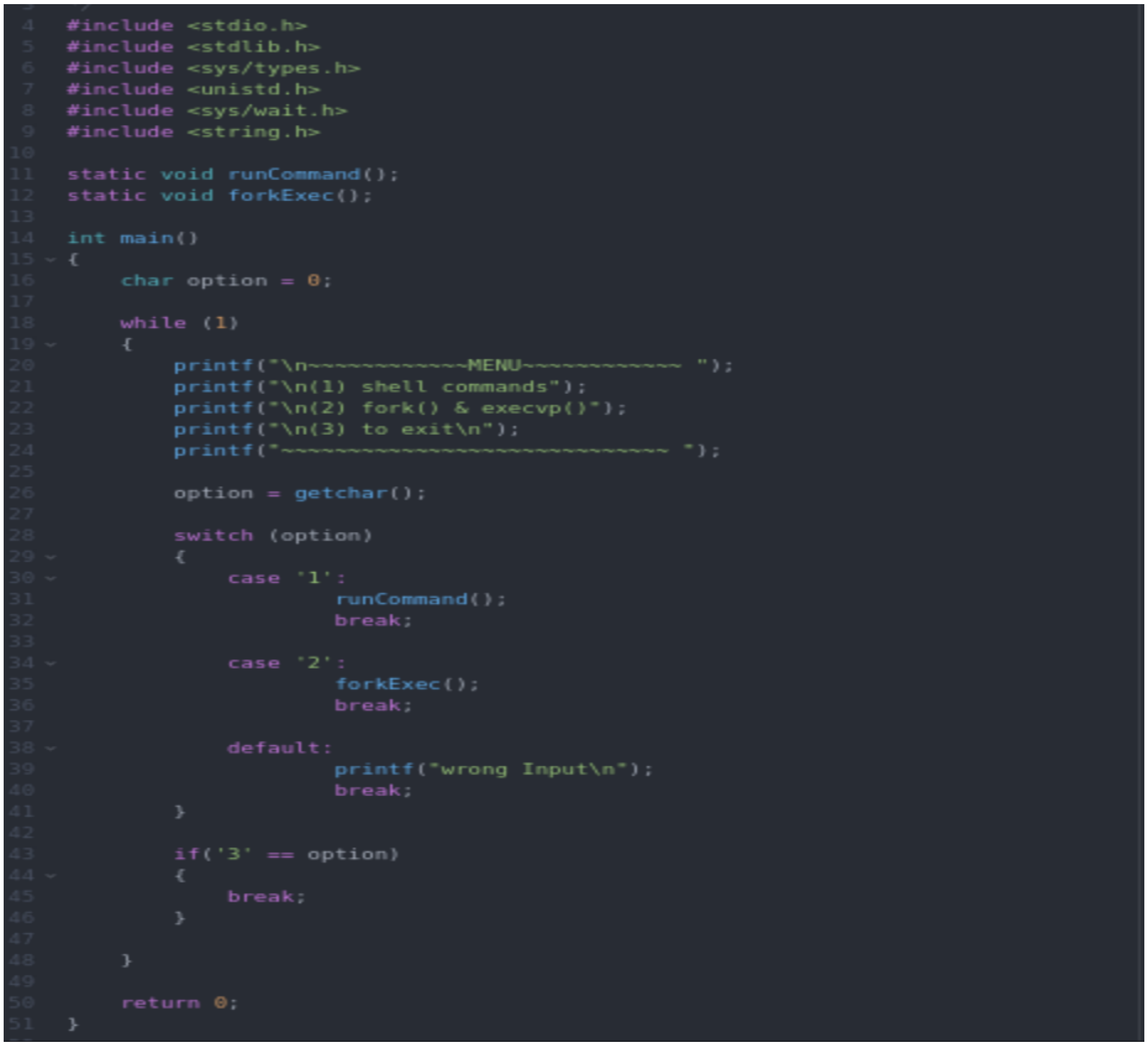


```
exit

~~~~~~~~~~~~~~MENU~~~~~~~~~~~~~~
(1) shell commands
(2) fork() & execvp()
(3) to exit
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
```

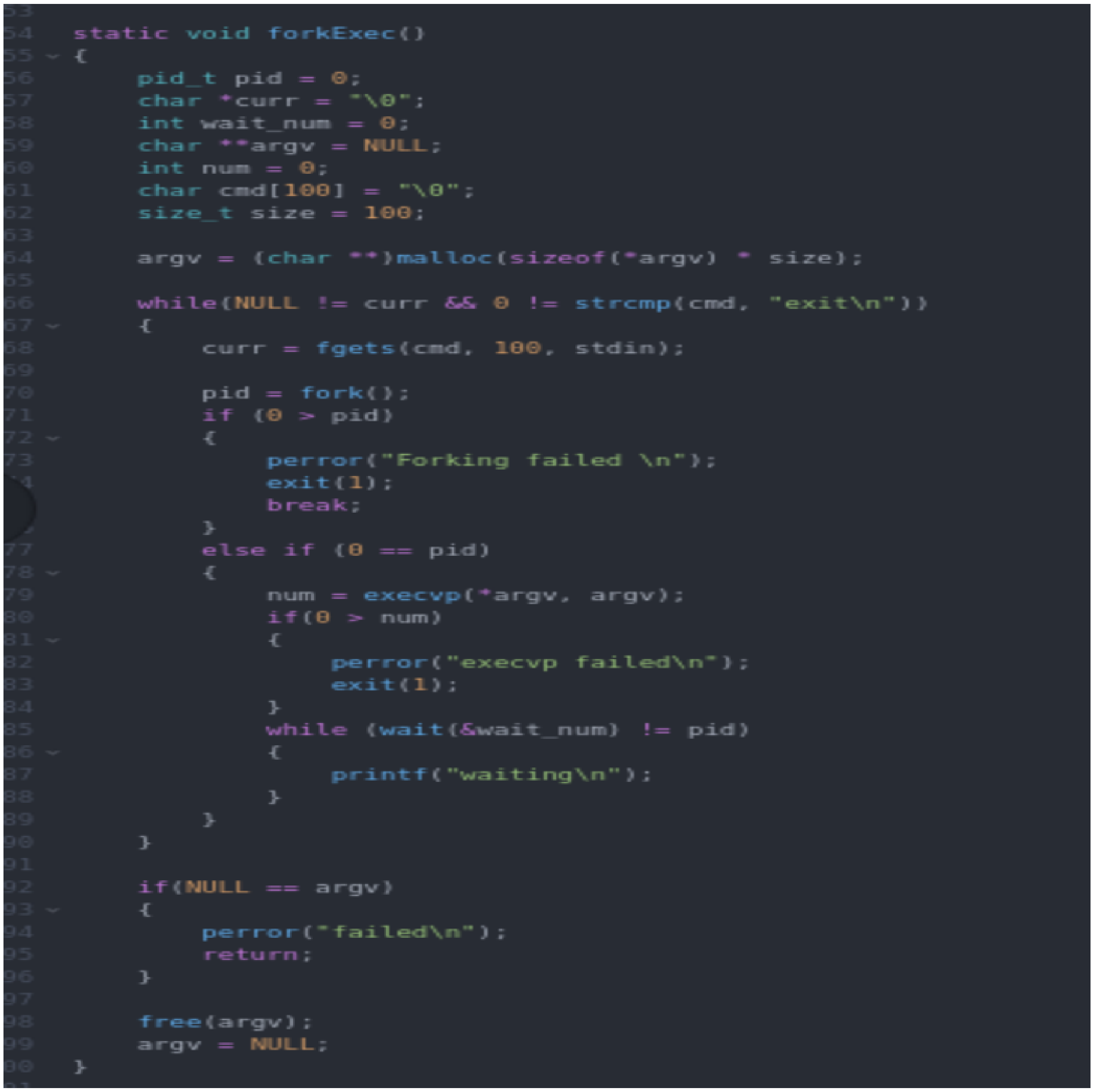


```
~~~~~~~~~~~~MENU~~~~~~~~~~~~
(1) shell commands
(2) fork() & execvp()
(3) to exit
~~~~~~~~~~~~~~~~~~~~~~~~~~~~
2
I am the child.  my pid=17675
hello
I am the child.  my pid=17677
forgetitt
I am the child.  my pid=17679
I am the child.  my pid=17680
```

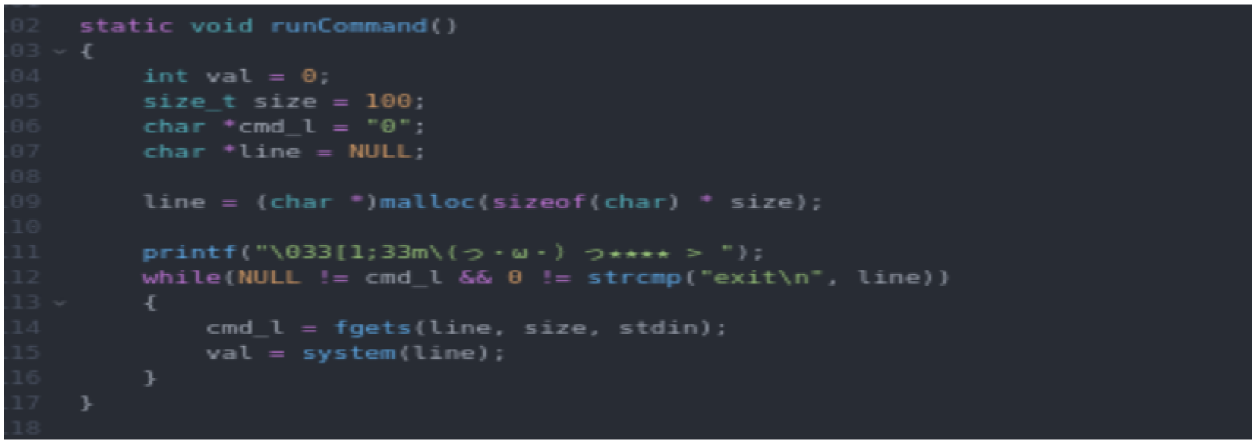
Program Created



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

static void runCommand();
static void forkExec();

int main()
{
    char option = 0;

    while (1)
    {
        printf("\n~~~~~~~~~~~~MENU~~~~~~~~~~~~ ");
        printf("\n(1) shell commands");
        printf("\n(2) fork() & execvp()");
        printf("\n(3) to exit\n");
        printf("~~~~~~~~~~~~~~~~~~~~~~~~~~~~ ");

        option = getchar();

        switch (option)
        {
            case '1':
                    runCommand();
                    break;

            case '2':
                    forkExec();
                    break;

            default:
                    printf("wrong Input\n");
                    break;
        }

        if('3' == option)
        {
            break;
        }

    }

    return 0;
}
```



```c
static void forkExec()
{
    pid_t pid = 0;
    char *curr = "\0";
    int wait_num = 0;
    char **argv = NULL;
    int num = 0;
    char cmd[100] = "\0";
    size_t size = 100;

    argv = (char **)malloc(sizeof(*argv) * size);

    while(NULL != curr && 0 != strcmp(cmd, "exit\n"))
    {
        curr = fgets(cmd, 100, stdin);

        pid = fork();
        if (0 > pid)
        {
            perror("Forking failed \n");
            exit(1);
            break;
        }
        else if (0 == pid)
        {
            num = execvp(*argv, argv);
            if(0 > num)
            {
                perror("execvp failed\n");
                exit(1);
            }
            while (wait(&wait_num) != pid)
            {
                printf("waiting\n");
            }
        }
    }

    if(NULL == argv)
    {
        perror("failed\n");
        return;
    }

    free(argv);
    argv = NULL;
}
```



```c
static void runCommand()
{
    int val = 0;
    size_t size = 100;
    char *cmd_l = "0";
    char *line = NULL;

    line = (char *)malloc(sizeof(char) * size);

    printf("\033[1;33m\(っ･ω･) っ★★★★ > ");
    while(NULL != cmd_l && 0 != strcmp("exit\n", line))
    {
        cmd_l = fgets(line, size, stdin);
        val = system(line);
    }
}
```

Conclusion

Creating a bash shell is not so difficult and allows for customizability, especially with using functions such as fork(), wait(), and execvp().