

Mini Operating System

April 12, 2018

201501243 - Kajal Gosaliya
201501231 - Helly Patel

Overview

An operating system (OS) is the program that, after being initially loaded into the computer by a boot program, manages all the other programs(applications) in a computer. An operating system is a program that acts as an interface between the user and the computer hardware and controls the execution of all kinds of programs.

We have created a mini operating system using **C language** and **Assembly language**. In our operating system, users can interact directly with the operating system through a command line interface.

Key Functionalities:

1. **Kernel:** The kernel is a computer program that is the core of a computer's operating system. It is the first program loaded on startup. It basically enables any other programs to execute. As name suggests, kernel runs in the kernel mode of the operating system. When running in kernel mode, the CPU can execute every instruction in its instruction set and use every feature of the hardware.
2. **Interrupt Handling :** Interrupt handling handles different kind of hardware and software interrupts using Interrupt Service Routines(ISR). For each interrupt there has to

some interrupt handling routine which tells CPU what to do when the interrupt arrives. To implement these routines correctly, we use a data structure called Interrupt Descriptor Table (IDT). For each interrupt, there will be a entry in IDT, routing to the interrupt handler.

3. **Shell:** A shell is a special program(or a command line interface) that offers users an interface with the computer. In our mini version of shell we have implemented following commands which users can use:

Commands	Functionality
clear	Clears the screen
cmd	Launches a new recursive shell
color	Changes the colors of the font and background of shell
calc	Basic calculator to add, subtract, multiply or divide 2 numbers
echo	Reprint the given text
exit	Exits the current shell
sort	Sort given numbers in increasing or decreasing order

Implementation :

We have divided our code in four folders :

1. Include :

This folder contains all the header files which are required to run our source codes written in C. Each header file(excluding types.h) have corresponding C file in src folder. In header files, we declare functions which we will be required to use in our kernel or shell. For each function declared, we will have it's implementation in corresponding C file. The reason we do this is if we define a function in a header file, then that function will appear in each module that is compiled that includes that header file. That makes our OS error-prone! We did not require to create a separate C file for types.h file as it does not contains any function declarations. Following is the list of all header files and what they are supposed to do:

Header file	Functionality
idt.h	Used to create Interrupt Descriptor Table(IDT) using 2 struct. One for how every interrupt handler gets defined. And other one for pointer to the table or array of these handlers.
isr.h	Used to add handlers(ISRs) for 32 processor exceptions in IDT.
system.h	Used to read data from input ports and write some data to output ports
screen.h	Used to implement some functions for command line interface like updating cursor, clearing the line or screen, setting screen and font colors

kb.h	Checks if any key is pressed and which key is pressed and changes display(show character or pressing enter) on the command line accordingly. We have done it by converting scan codes of keys to ASCII codes
types.h	Used to define different data types like integer, float and string
util.h	Used for copying,setting and allocating memory
string.h	Used for implementing different functions on strings like counting string length, converting string to integer, comparing two strings
shell.h	Used to create the structure of our basic shell and implement different basic commands of shell like cmd, clear, calc,etc.

2. Src :

As mentioned earlier, in this folder, we have C files corresponding to header files of *include folder*. This C files implements the functions declared in corresponding header files. Additional to them we have three additional files: kernel.asm, kernel.c and link.ld .

Our kernel.asm file defines that our operating system is **32 bit OS**. It also defines some standard for loading various **x86 kernels** using a **boot loader**; called as **Multiboot specification**. GRUB will only load our kernel if it complies with the Multiboot spec.It also tells our operating system that from where to start or what to do initially when loaded. It directs OS to function defined in kernel.c . So our kernel.c file contains information what our OS will do when loaded first. It makes call to a function which gets our **interrupt descriptor table** created with details of handlers so that our OS can handle the interrupts. And then kernel.c opens the shell with which user can interact with our operating system and type commands in the shell. The **link.ld** file is used to link the **executables of kernel.asm and kernel.c** . It also defines some **memory reference** in main memory from where our operating system gets started. In other words, our OS runs in that memory space.

Source file	Basic Function and their Implementation
idt.c	set_idt_get() : defines attributes for entry of IDT. set_idt() : define start address and sizeof IDT.
isr.h	isr_install() : set attributes for all the entries of IDT. isr0() isr31() : exception handling and interrupt for 32 processor exceptions.
kb.c	readStr() : It will check whether a key is pressed or not using input port and if pressed then read it's value, display it on screen and store it in a string.
kernel.c	kmain() : it is the function that called from kernel.asm to start execution of our kernel. It set the interrupt descriptor table, ISRs and launches shell using launchShell function.
screen.c	print() : It prints given string on screen character by character. clearLine() : clears the given lines. clearScreen() : clears the whole screen. updateCursor() : updates the position of cursor using CRT control register. scrollUp() : Basically scroll the windows up by moving the line up and removing line at the bottom.
shell.c	launchShell() : Read user's commands and call function according it.
string .c	strEq() : check whether two strings are equal or not. strLength() : calculate the length of string character by character. int_to_string() : convert from int to string. str_to_int() : convert from string to int
system.c	inportb() : reads data from input ports outportb() : writes data to output ports

util.c	memory_copy() : copy chunk of a memory from one place to another. memory_set() : set value of memory chunk. malloc() : take number of bytes, define a string with given size and return it's address to allocate memory.
---------------	---

3. Obj :

Folder obj contains all object files of all the C codes and assembly codes. These object files are generated by compiling those C and assembly codes. We have used **gcc** and **nasm** to compile C and assembly codes respectively.

4. Boot :

Linker combines object files with and links them with link.ld and creates a binary file(**kernel.bin**) which is located in boot folder. Emulator **qemu** uses this kernel.bin file to emulate our final operating system.

5. makefile :

Makefile is to run the necessary commands required to run our operating system.

Conclusion :

We have implemented basic and mini version of operating system successfully. We have implemented some basic functionalities like kernel, interrupt handling and shell which any other operating system will require. We have created these functionalities with maximum care so that it does not take much space in the memory and runs smoothly with speed. For example, in defining struct, we do not allow compiler to add padding such that it acquires extra space. Also, we have just declared functions in header files and implemented them in other source files.

Scope :

Although we have created mini version of shell, but with implementation basic commands like clear, cmd, echo and mini application type calc we can always implement other complex commands and run other bigger applications. As we have implemented basics requirements for any operating system, we can further develop complex features like memory management, file management, job control and scheduling, device management using different device drivers, etc.

Screenshots :

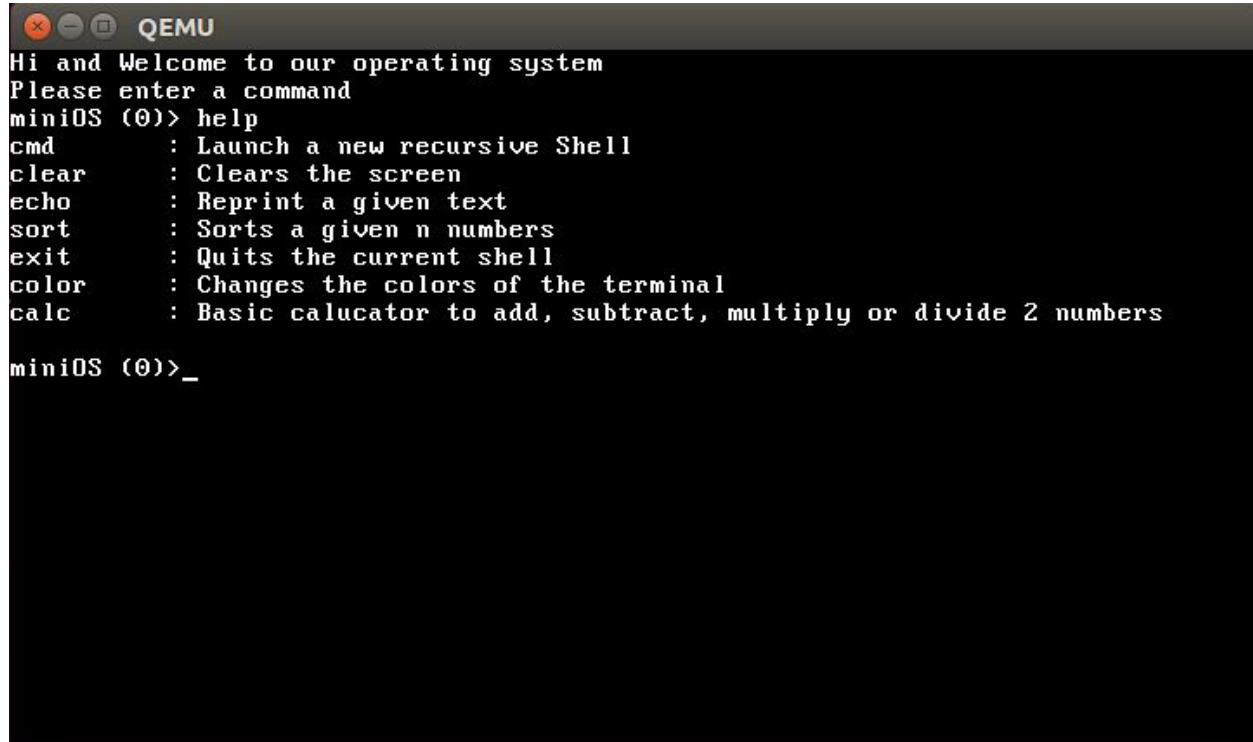
- ❖ We have used [QEMU emulator](#) for running this miniOS.

Make command in Terminal

```
kajal@kajal:/media/kajal/New Volume/Studies Stuff/Semester 6/Operating Systems/miniOS$ make
nasm -f elf32 -o obj/kasm.o src/kernel.asm
gcc -m32 -c -ffreestanding src/screen.c -o obj/screen.o
gcc -m32 -c -ffreestanding src/string.c -o obj/string.o
gcc -m32 -c -ffreestanding src/system.c -o obj/system.o
gcc -m32 -c -ffreestanding src/util.c -o obj/util.o
src/util.c: In function 'int_to_ascii':
```

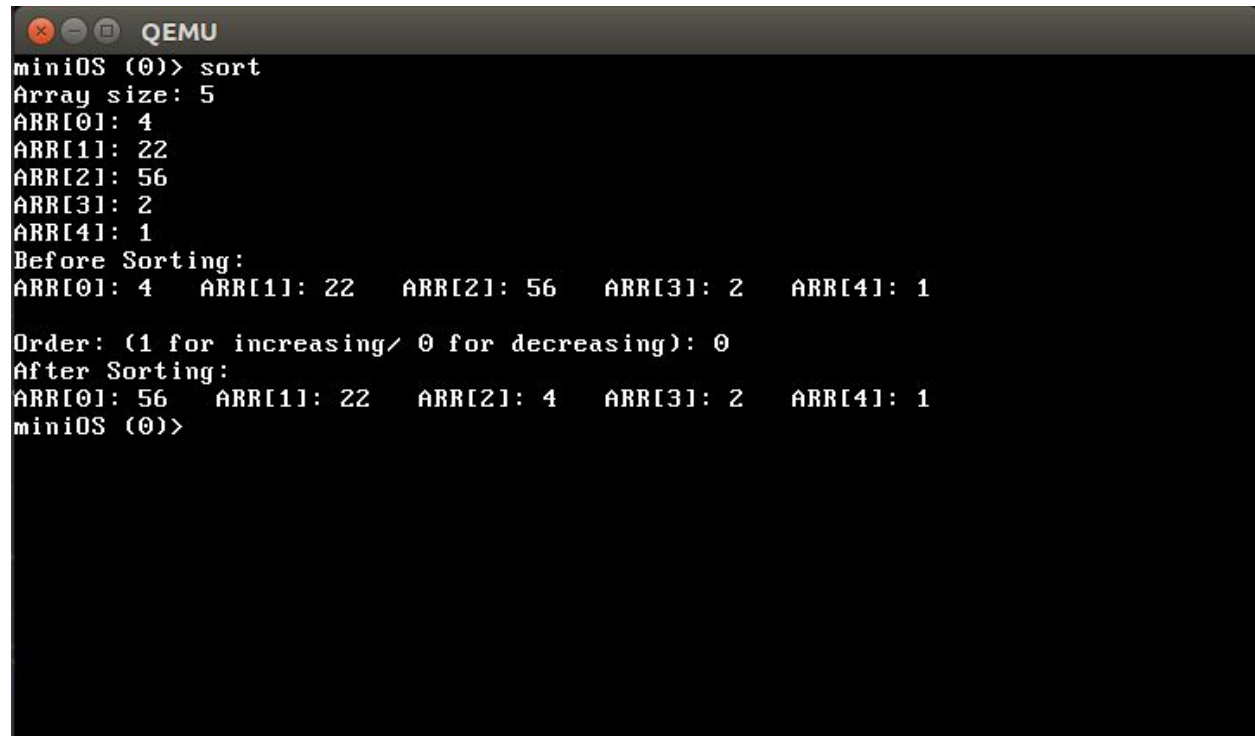
```
kajal@kajal:/media/kajal/New Volume/Studies Stuff/Semester 6/Operating Systems/miniOS$ make
mkdir boot/ -p
ld -m elf_i386 -T src/link.ld -o boot/kernel.bin obj/kasm.o obj/kc.o obj/idt.o obj/isr.o obj/kb.o obj/screen.o obj/string.o obj/system.o obj/util.o obj/shell.o
qemu-system-i386 -kernel boot/kernel.bin
```

Help Function



```
QEMU
Hi and Welcome to our operating system
Please enter a command
miniOS (0)> help
cmd      : Launch a new recursive Shell
clear    : Clears the screen
echo     : Reprint a given text
sort     : Sorts a given n numbers
exit     : Quits the current shell
color    : Changes the colors of the terminal
calc     : Basic calucator to add, subtract, multiply or divide 2 numbers
miniOS (0)>_
```

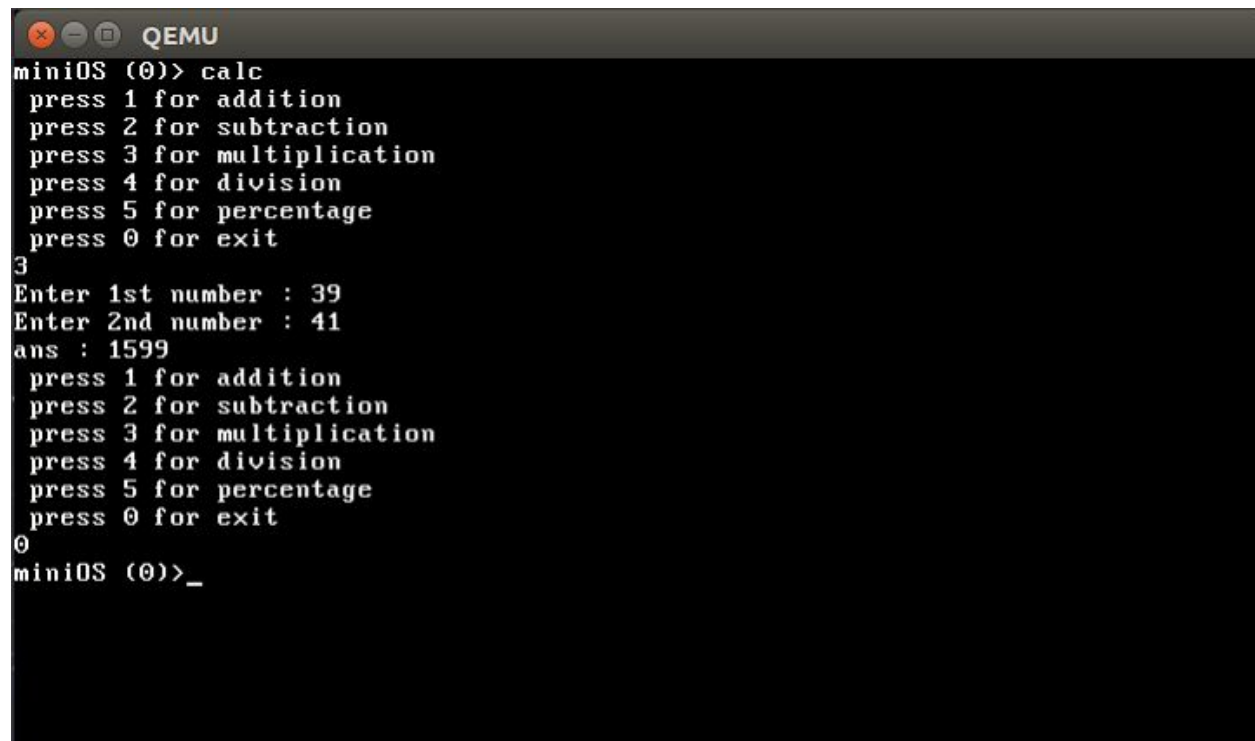

Sort Function



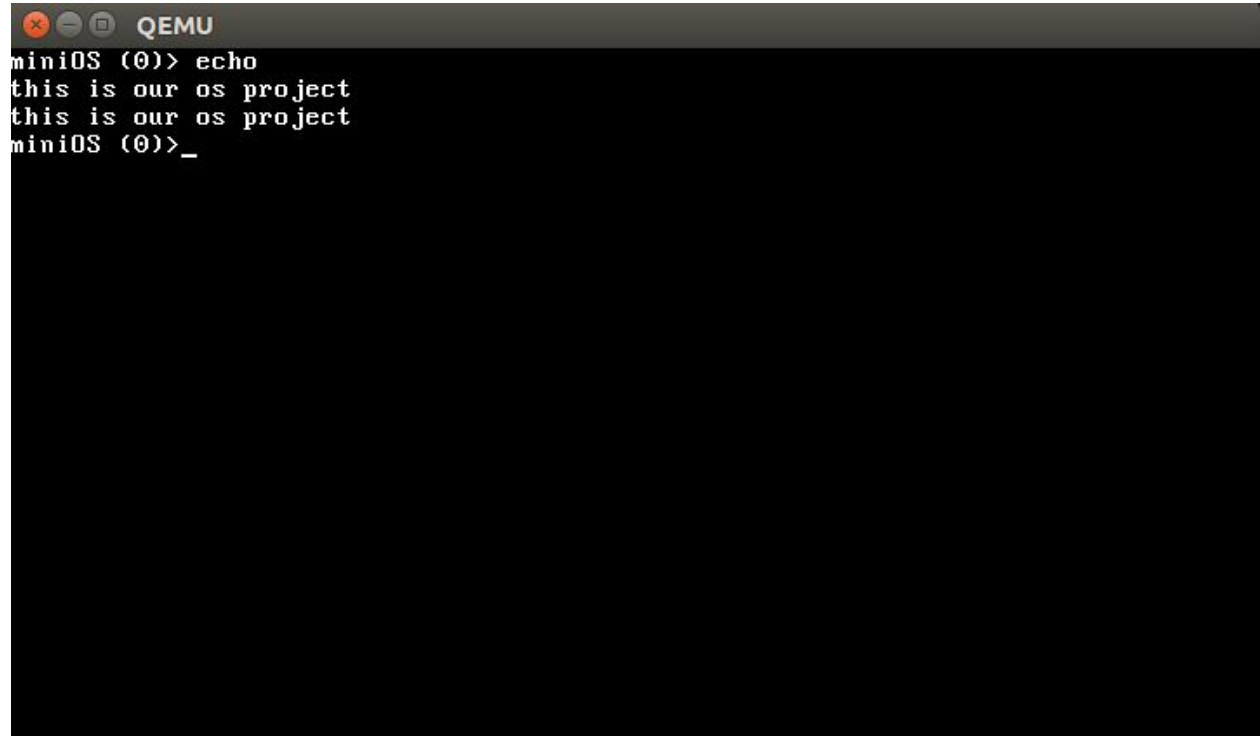
```
miniOS (0)> sort
Array size: 5
ARR[0]: 4
ARR[1]: 22
ARR[2]: 56
ARR[3]: 2
ARR[4]: 1
Before Sorting:
ARR[0]: 4   ARR[1]: 22   ARR[2]: 56   ARR[3]: 2   ARR[4]: 1

Order: (1 for increasing/ 0 for decreasing): 0
After Sorting:
ARR[0]: 56   ARR[1]: 22   ARR[2]: 4   ARR[3]: 2   ARR[4]: 1
miniOS (0)>
```

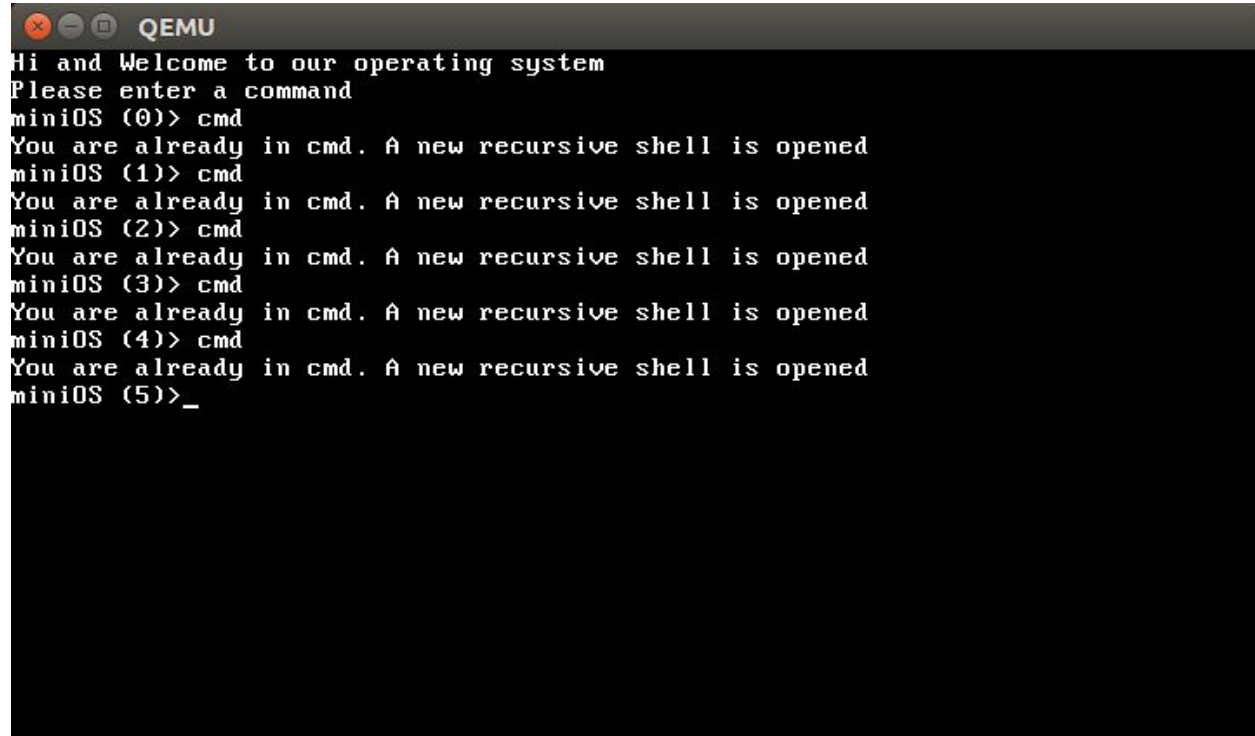
The screenshot shows a QEMU terminal window with a black background and white text. The window title is 'QEMU'. The user enters the command 'sort' in the 'miniOS (0)>' prompt. The program outputs the array size (5) and the initial values of the array (ARR[0] to ARR[4]). It then displays the state 'Before Sorting' with the same values. The user is prompted for the sort order (1 for increasing, 0 for decreasing) and enters '0'. The program then displays the state 'After Sorting' with the array values rearranged in descending order (56, 22, 4, 2, 1). The prompt 'miniOS (0)>' is shown again at the bottom.

Calc Function :

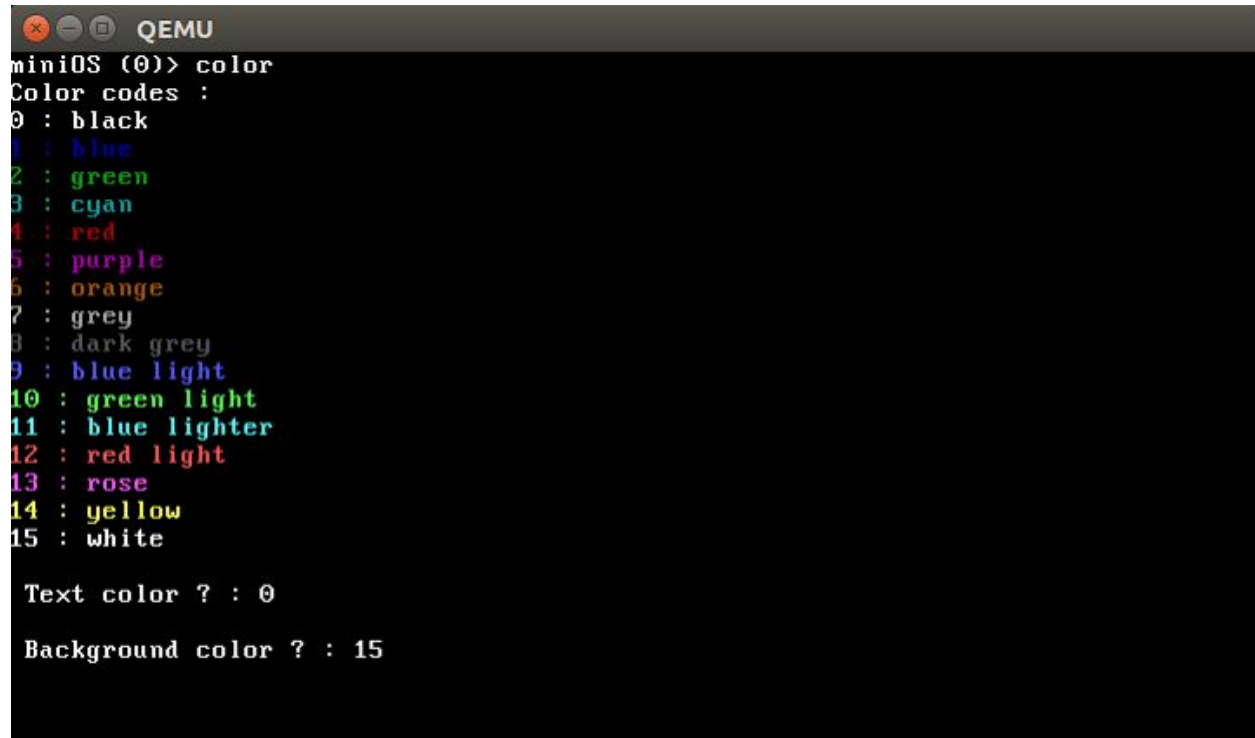
```
QEMU
miniOS (0)> calc
press 1 for addition
press 2 for subtraction
press 3 for multiplication
press 4 for division
press 5 for percentage
press 0 for exit
3
Enter 1st number : 39
Enter 2nd number : 41
ans : 1599
press 1 for addition
press 2 for subtraction
press 3 for multiplication
press 4 for division
press 5 for percentage
press 0 for exit
 
miniOS (0)>_
```

Echo Function :

```
QEMU
miniOS (0)> echo
this is our os project
this is our os project
miniOS (0)>_
```

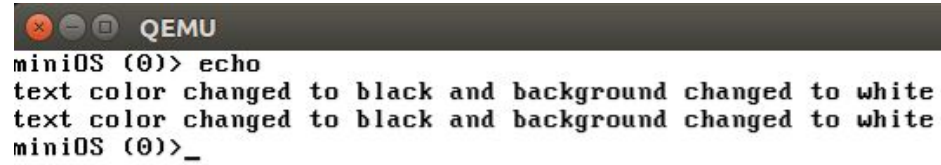
Recursive Shell using cmd Function :

```
QEMU
Hi and Welcome to our operating system
Please enter a command
miniOS (0)> cmd
You are already in cmd. A new recursive shell is opened
miniOS (1)> cmd
You are already in cmd. A new recursive shell is opened
miniOS (2)> cmd
You are already in cmd. A new recursive shell is opened
miniOS (3)> cmd
You are already in cmd. A new recursive shell is opened
miniOS (4)> cmd
You are already in cmd. A new recursive shell is opened
miniOS (5)>_
```

Changing colour using Color Function :A screenshot of a QEMU terminal window. The window title is 'QEMU'. The prompt is 'miniOS (0)>'. The user has entered the command 'color'. The output shows a list of color codes from 0 to 15, each with a corresponding color name. The text is displayed in the specified colors: 0 is black, 1 is blue, 2 is green, 3 is cyan, 4 is red, 5 is purple, 6 is orange, 7 is grey, 8 is dark grey, 9 is blue light, 10 is green light, 11 is blue lighter, 12 is red light, 13 is rose, 14 is yellow, and 15 is white. Below the list, the prompt asks 'Text color ? : 0' and 'Background color ? : 15'.

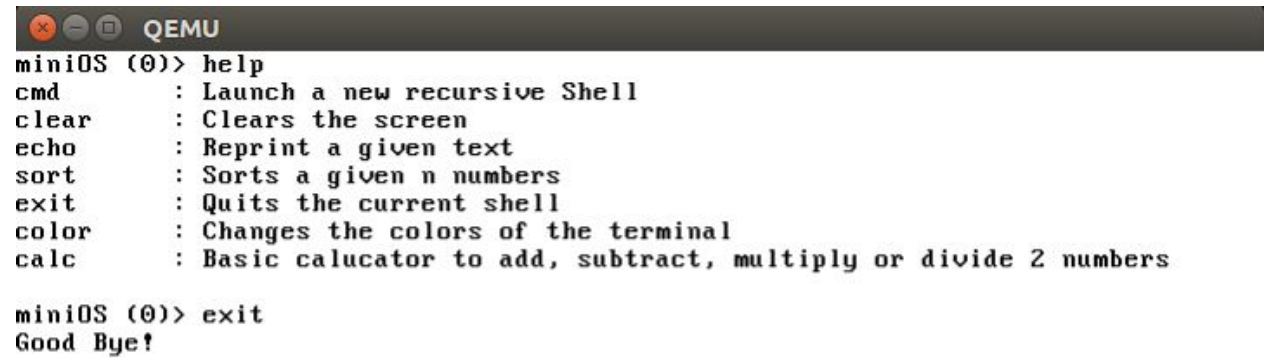
```
QEMU
miniOS (0)> color
Color codes :
0 : black
1 : blue
2 : green
3 : cyan
4 : red
5 : purple
6 : orange
7 : grey
8 : dark grey
9 : blue light
10 : green light
11 : blue lighter
12 : red light
13 : rose
14 : yellow
15 : white

Text color ? : 0
Background color ? : 15
```

Color Changed :A screenshot of a QEMU terminal window. The window has a dark gray title bar with the text "QEMU" and three window control buttons (close, maximize, and a third button). The terminal content shows a command prompt "miniOS (0)>" followed by the command "echo". The output of the command is displayed on two lines: "text color changed to black and background changed to white". The prompt "miniOS (0)>" is followed by an underscore character "_".

```
miniOS (0)> echo
text color changed to black and background changed to white
text color changed to black and background changed to white
miniOS (0)>_
```

Exit function to exit a shell :



```
QEMU
miniOS (0)> help
cmd      : Launch a new recursive Shell
clear    : Clears the screen
echo     : Reprint a given text
sort     : Sorts a given n numbers
exit     : Quits the current shell
color    : Changes the colors of the terminal
calc     : Basic calucator to add, subtract, multiply or divide 2 numbers

miniOS (0)> exit
Good Bye!
```

References :

[1] <http://wiki.osdev.org/>