Tutorial 0 : Introduction

August 11, 2020

Objective : To introduce the concept of virtualization performed by OS

Operating System is the software which facilitates interaction of applications and hardware devices. In addition it allows multiple programs to share memory and thus enables the CPU to run these programs in a time sharing fashion.

OS provides all the above mentioned facilities through a technique known as virtualization - it takes aphysical resources such as the processor, or memory, or a disk and transforms them into into a more general and easy-to-use virtual form.

This tutorial is intended to provide the students an introduction to the concept of virtualization.

The first program *cpu.c* shows how the os virtualizes the cpu – makes each application (program) believe that it has it own cpu on which it is running. However, in reality the applications (programs) share the single cpu.

The next program *mem.c* shows how OS virtualizes memory. Each process accesses (running program) has its own private virtual address space, which the OS maps on to the physical memory of the machine. A memory reference within one process (running program) does not affect the address space of other processes (or the OS itself)

Recommended Systems/Software Requirements: Any avour of Linux

References:

Operating Systems: Three Easy Pieces
Remzi H. Arpaci-Dusseau and Andrea C. Arpaci-Dusseau
Arpaci-Dusseau Books
August, 2018 (Version 1.00)

Web : http://pages.cs.wisc.edu/~remzi/OSTEP/

- **Virtualizing CPU :**

Instructions : Compile and run the program *cpu.c*
Make sure to include the *common.h* file in the same folder as the *cpu.c*

Here is what we will see:

```
prompt> gcc –o cpu cpu.c –Wall

prompt> ./cpu "IIITA"


IIITA
IIITA
IIITA
....
```

Press ^*c* to stop the execution (halt the program)

The system begins running the program, which repeatedly checks the time until a second has elapsed. Once a sec-ond has passed, the code prints the input string passed in by the user (in this example, the letter "*IIITA*"), and continue until *Control-C* is pressed

ii. Run different instances of the same program

```
prompt> gcc –o cpu cpu.c –Wall

prompt> ./cpu "IIITA"  ./cpu "IIITL" ./cpu "IIITD"
[27989]
[27990]
[27991]
IIITA
IIITL
IIITD
IIITA
IIITL
IIITD
IIITA
IIITL
IIITD
....
```

Even though the system utilizes one processor, all three instances of the program (three processes) seem to be running at the same time. This creates an illusion that the system has a large number of vitual processors each catering to one instane of the program.

- **Virtualizing memory**

Physical memory model presents Memory as an array of bytes. The data stored at an address of memory can be read by providing the addressof the location wehere the data resides. Similarly, while writing data to any address of a memory, one needs to specify the address for writing as well as the data to be written to the given address. Memory is accessed all the time when a program is running. A program keeps all of its datastructures in memory, and accesses them through various

instruction. Instructions themeselves reside in the memory which have to be fetched.

Instructions:

i. Execute the code *mem.c*

Make sure to include the common.h file in the same folder as the mem.c.

To execute use the following commands :

```
prompt> gcc mem.c
prompt> ./a.out 0

(28332) addr pointed to by p: 0x9054008

(28332) value of p: 1
(28332) value of p: 2
(28332) value of p: 3
......
```

The program does a couple of things.

First, it allocates some memory(line 1). Then, it prints out the address of the memory (line 2), and the puts the number zerointo the first slot of the newly allocated memory(line 3). Finally,it loops, delaying for a second and incrementing the value stored at the slot.

ii. Now, run multiple instances of this same program

```
prompt> gcc mem.c
prompt> ./a.out 0& ./a.out 0

[1] 28335
[2] 28336
(28335) address pointed to by p: 0x200000
(28336) address pointed to by p: 0x200000
(28335) p: 1
(28336) p: 1
(28335) p: 2
(28336) p: 2
.......
```

We see that each running program has allocated memory at the same address (0x200000), and yet each seems to be updating the value at 0x200000 independently. It is as if each running program has its own private memory, instead of sharing the same physical memory with other running programs. This is how OS virtualizes memory. Each process accesses its own private virtual address space