Implement a Custom System Call using Docker

Step -1:

Install docker on the host machine/vm

sudo apt update

sudo apt install -y docker.io qemu-system-x86

verify whether docker has been installed in the terminal

docker –version

Step -2:

Create docker image for kernel build

i)create a project directory

(e.g) kernel-docker.

```
student@student-virtual-machine:~$ mkdir kernel-docker
student@student-virtual-machine:~$ cd kernel-docker
```

Step -3:

Creating a Dockerfile and building it.

```
student@student-virtual-machine:~/kernel-docker$ vi Dockerfile
student@student-virtual-machine:~/kernel-docker$ docker build -t kernel-builder
```

```
FROM ubuntu:22.04
ENV DEBIAN_FRONTEND=noninteractive
RUN apt update && apt install -y \
        build-essential \
        bc \
        bison \
        flex \
        libssl-dev \
        libelf-dev \
        libncurses-dev \
        git \
        vim \
        wget \
        cpio \
        && rm -rf /var/lib/apt/lists/*
WORKDIR /kernel
```

Step -4:

To run the kernel builder container

```
student@student-virtual-machine:~/kernel-docker$ docker run -it --name kernel-dev \
-v $PWD:/kernel \
kernel-builder
```

Step -5:

Download linux kernel source code in the docker and extract it using tar cmd

```
root@92f206db8b9a:/kernel# wget https://cdn.kernel.org/pub/linux/kernel/v6.x/linux-6.6.tar.xz
--2026-02-03 16:05:01--  https://cdn.kernel.org/pub/linux/kernel/v6.x/linux-6.6.tar.xz
```

After installing navigate to linux-6.6 directory

```
root@92f206db8b9a:/kernel# cd linux-6.6
```

Step -6:

Adding custom system call

i)create a folder in kernel directory of linux-6.6

```
root@92f206db8b9a:/kernel/linux-6.6# mkdir kernel/custom_syscalls
root@92f206db8b9a:/kernel/linux-6.6# vim kernel/custom_syscalls/stringrev_syscall.c
```

ii)create a program file for implementing custom system call for string reverse

```c
# include <linux/kernel.h>
# include <linux/syscals.h>
# include <linux/uacesss.h>

SYSCALL_DEFINE3(reverse_string,
    char __user *, input,
    char __user *, output,
    int, len)
  {
    char kbuf[256];
    int i;
    if (len <= 0 || len > 255)
      return -EINVAL;
    if (copy_from_user(kbuf,input,len))
      return -EFAULT;
    kbuf[len] = '\0';
    for (i = 0; i < len / 2; i++) {
      char temp = kbuf[i];
      kbuf[i] = kbuflen - i - 1];
      kbuf[len - i - 1] = temp;
    }
    if (copy_to_user(output, kbuf, len +1))
      return -EFAULT;
    }
    printk(KERN_INFO "reverse_string syscall executed\n");
    return 0;
}
```

Step -7:

create a makefile for custom system call program

```
root@92f206db8b9a:/kernel/linux-6.6# vim kernel/custom_syscalls/Makefile
```

In makefile add this

obj-y := stringrev_syscall.o

Then Add a entry in makefile for the custom system call in kernel makefile

```
root@92f206db8b9a:/kernel/linux-6.6# vim kernel/Makefile
```

```
obj-y += custom_syscalls/
```

Step -8:

Register the syscall number in (x86_64) tbl file

```
root@92f206db8b9a:/kernel/linux-6.6# vim arch/x86/entry/syscalls/syscall_64.tbl
```

Add this line

```
454     common          reverse_string          sys_reverse_string
```

Step -9:

Declare syscall prototype in syscalls.h header file.

```
root@92f206db8b9a:/kernel/linux-6.6# vim include/linux/syscalls.h
```

```
asmlinkage   long  sys_reverse_string(const char __user *input, char __user *output, int len);
```

Step -10:

Configure the kernel now

```
root@92f206db8b9a:/kernel/linux-6.6# make defconfig
```

Step -11:

Build the kernel inside the docker

```
root@92f206db8b9a:/kernel/linux-6.6# make -j$(nproc)
```

```
Kernel: arch/x86/boot/bzImage is ready  (#1)
```

Step -12:

Now create a minimal root file system

i)create rootfs folder in kernel-docker directory and navigate to it and create necessary directories inside it

```
mkdir rootfs
cd rootfs
mkdir -p bin sbin etc proc sys usr/bin usr/sbin
```

ii)create init process

1)create a init file

```
vim init
```

2)add this line inside

```
#!/bin/sh
mount -t proc none /proc
mount -t sysfs none /sys
exec /bin/sh
```

3) make the file executable

```
chmod +x init
```

Step -13:

Install busybox in host or docker

```
sudo apt install -y busybox-static
```

Copy busybox to rootfs directory

```
cp /bin/busybox rootfs/bin/
```

Now create applet links for required one with busybox in rootfs

```
ln -s busybox rootfs/bin/sh
ln -s busybox rootfs/bin/cd
ln -s busybox rootfs/bin/dmesg
ln -s busybox rootfs/bin/head
ln -s busybox rootfs/bin/tail
ln -s busybox rootfs/bin/mkdir
```

now build initramfs from rootfs directory

```
cd rootfs
find . | cpio -o --format=newc > ../initramfs.cpio
```

Step -14:

Test the custom system call using a program

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

#define __NR_reverse_string 454

int main()
{
    char input[256];
    char output[256];
    int len;

    scanf("%s", input);
    len = strlen(input);

    long ret = syscall(__NR_reverse_string, input, output, len);

    if(ret < 0){
        perror("syscall failed");
        return 1;
    }

    if(ret >= 0)
        write(1, "Reversed: ", 10);
        write(1, output, strlen(output));
        write(1, "\n", 1);

    return 0;
}
```

Compile the program in static mode

```
gcc test.c –static –o test
```

move the obj file to kernel-docker/rootfs and create subdirectory there and store it.

```
mv ~/kernel-docker/test ~/kernel-docker/rootfs/test
```

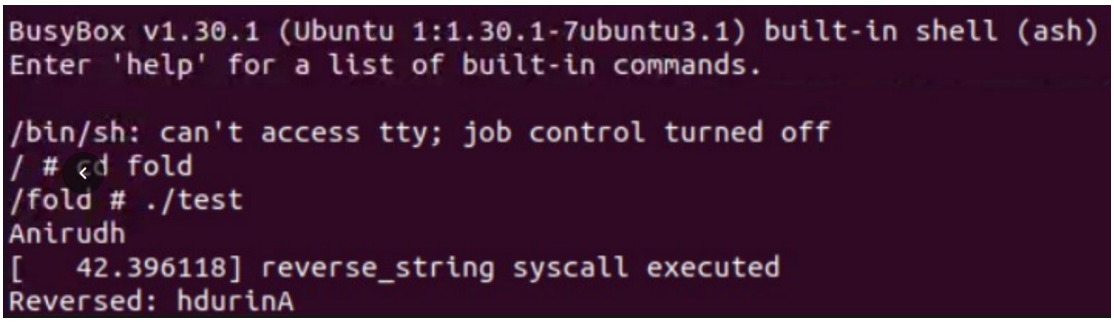
now once again build initramfs from rootfs directory

```
cd rootfs
find . | cpio -o --format=newc > ../initramfs.cpio
```

Step -15:

Boot kernel using QEMU (Host)

Navigate to your folder you created in rootfs for obj file and run it to test the custom system call.



Then type dmesg cmd to check the kernel buffer output during custom system call