READ-ME for Question 1

FILES USED: main.c, SR.c, ST.c, makefile

We will make three files for this programme, they are as follows: main.c and ST.c and SR.c

In main.c we have a sigterm handler wherein we print whatever we get from SR.c. In our main function, we spawn two child processes SR and ST respectively. Then whenever we are executing the respective child process either SR or ST there we execute the different respective processes for SR and ST while the parents remain the same. So from this we create two more processes which are different in nature unlike the similar child processes. Here we also pass the parent ID in the form of a string

Now in SR.c we initiate a itimerval structure and initialises its parameter. We further created a sigaction structure as well and initialise its handler parameter and then call it.Then using sigaction we call the sigalarm which calls the random number generator mentioned in our inline assembly and then we first print it here in our sigalarmhandler function and then we further queue it in siguqueue. This then gets printed in our main.c file where we have passed the parent ID as well.

Lastly in ST.c we do the same with initialiser of itimerval and sigaction. But here we first call the SIGALARM and then we call the SIGTERM using sigaction. In sigalarm handler we first queue the union into SIGTERM and at SIGTERM we find the cpu cycles printed in the form of hrs and minutes and seconds. The getCPUCycles() is an inline assembly code which assigns a value and this is then assigned to the union which is then queued at the SIGTERM. We then print it in hrs and minutes. However it may or may not be accurate depending upon the frequency of our processor. However we see that it keeps incrementing second by second. So it acts like a sort of a clock.

To stop the code from running we press control + c and stop it.

In order to run the files and compile it we make use of the makefile command. In order to do that we just write

```
make
```

In order to remove all output files we write

```
make clean
```

READ-ME for Question 2

FILES USED: testCase, output.patch

For this question we need to do basically two things.

1) We need to add the syscall to the table of already existing sys calls
2) Then we need to edit the files where the implementation of all sys calls is present and there we need to put in the implementation of our kernel_2d_memcpy.

Adding syscall to the table of syscalls

We first go to build/linux-5.xx.xx/arch/x86/entry/syscalls/ syscall_64.tbl . Now depending upon our linux version we will have 440+ sys calls . We now add a new syscall as follows:

```
448 kernel_2d_memcpy sys_kernel_2d_memcpy
```

Editing the file sys.c using vim

Now we need to go to build/linux-5.xx.xx/kernel then we write ‘vim sys.c’ now we have opened up the sys.c file. In this first we go to the WRITE MODE, then we find some empty segment where we make our function as follows:

```
SYSCALL_DEFINE4(kernel_2d_memcpy, float *, mat1, float *, mat2, int, row_number, int, column_number)
```

Here we take in 4 parameters, first we have the name of our sys call then we input two matrices (we read from matrix 2 and writing it on matrix 1) and the other two parameters are the row and column.

We use two functions to achieve the same which are copy_from_user and copy_to_user

In copy_from_user (destination address in kernel space, source address in user space, no. of bytes to copy) we input a temporary matrix as our destination matrix and on our source we input the matrix 2. Now for bytes to copy we simply use sizeof(new Matrix) function. If this does not return 0 then we print an error.

In copy_to_user (destination address in user space, source address in kernel space, no. of bytes to copy). Here we input our matrix 1 where we need to provide the data and that is provided by making use of our temporary matrix. For bytes, again simply use sizeof() function. If this does not return 0 then we print an error.

After doing the following we just need to update our kernel and then we have achieved the making of kernel_2d_memcpy syscall.