In our practice today, we will implement SystemTap scripts. This exercise assumes you are using Fedora (as suggested by the instructor) or another RedHat based distribution. Commands for installation might differ for different distros, but the scripts are “portable”, i.e., if you use another flavor of Linux (e.g., Ubuntu) and/or you already got SystemTap installed, you just need to skip the first step of this exercise.

1. The first and most basic step is the actual installation of SystemTap. To install it in your VM using Fedora, use the following commands:

```
sudo yum install systemtap systemtap-runtime
```

The command above will install the actual SystemTap programs, e.g., stap utility.

```
sudo yum install kernel-devel-`uname -r`
sudo debuginfo-install kernel-`uname -r`
```

These two commands install tools that SystemTap uses for monitoring. SystemTap essentially generates a module, so we need the kernel development tools. (You probably already have it installed). We also need debug information in the kernel so we can get function names and variables.

2. Now, that we installed SystemTap, let us write our first script. Just for tradition, we will write an improved “hello world” script. In a file `hello.stp`, type the following code:

```
probe begin
{
    print ("Hello, world\n")
    print ("I am going to rest and than leave\n")
}

probe timer.ms(4000) # waits for 4 seconds
{
    print("Bye\n")
    exit()
}
```

To execute this script, we will use the following command. (-v is for verbose).

```
stap -v hello.stp
```

Observe the steps for executing the script. It is parsed, translated into C, compiled as a module and loaded.
3. Now, we will write a script that counts how many forks happened in the system. Write the script below in a file `fork.stp`.

```stp
global proc_counter
probe begin {
    print ("Starting monitoring..."
    print ("Press ^C to terminate\n"
    printf ("%-25s %­10s %­s", "Process Name", "PID", "Flags")
}
probe kernel.function("do_fork") {
    proc_counter++
    printf("%-25s %­10d 0x%­x", execname(), pid(), $clone_flags)
}
probe end {
    printf ("\n%d processes forked", proc_counter)
}
```

Some interesting details to observe in this script are the use of a global variable (`proc_counter`) and the probe for a specific kernel function. It is also interesting to notice that is prints `$clone_flags` that we do no refer anywhere else in our code. `code_flags` is actually a variable that exists in the function `do_fork`.

Now, execute the script to see its output. You can open another terminal window and call a few commands (e.g., ls, cat). Each program that you execute should cause our script to print a line for `bash`.

4. Another interesting feature is the ability of choosing which program we want to monitor. The following script (called `targeted.stp`) will print the 5 more executed system calls for a given target.

```stp
global syscalls
probe syscall.* {
    if ( pid() == target() )
        syscalls[name] += 1
}
probe end {
    foreach(n in syscalls­ limit 5)
        printf("%s = %d
", n, syscalls[n])
}
```

To determine a target, we have two options. First, you can give an PID to stap, using the option `-x` as the command line below. Feel free to test it with the PID of a process in execution.

```bash
sudo stap targeted.stp -x 1234
```
Another way is giving the command or program (option `-c`). Notice that if the corresponding process is not in execution, it will wait until it finds a matching process. When that process is terminated, the script also ends. The following command will monitor the command `ls`.

```
sudo stap targeted.stp -c ls
```

In another terminal windows, try to call `ls` and see what happens to our script.

5. Can you modify the script above to print the 5 less executed system calls for a targeted process?

6. And what about all the system calls?

7. Now, write a script that, every ten seconds, displays the top five most frequent users of the `open` system call during that interval.

   Hint: You can use `kernel.function`, but try to use `syscall` probe for this. Use the function `uid()` to identify users.

   Remember, the following commands can give you a more extensive list of probes and functions you can use in your scripts.

   ```
   man stapprobes
   man stapfuncs
   ```

   For your reference, a solution for this question shown in the next page. But you should try that by yourself.

8. Challenge Question: Write a script that traces kernel control flow starting at each system call made by a process.
Solution for step 7 (*open_frequusers.stp*)

```plaintext
global users

probe begin
{
    print ("Counting opens...\nPress ^C to terminate this script\n")
}

probe syscall.open {
    users[uid()] += 1
}

probe timer.s(10)
{
    foreach (u in users - limit 5)
        printf("User %d called open %d times\n", u, users[u])
    delete users
}
```

A few notes:
- As only user of your system, your list might have just 2 users: Root (uid=0) and you (on Fedora, first user create receives uid=500)
- The timer probe has options for measuring time using different units. So, `timer.ms(10000)` is equivalent to `timer.s(10)`. 