1. Run "/usr/share/bcc/tools/tplist". What does the output represent?
   1. This tool displays the tracepoints and probes within the kernel which could be used for tracing the block layer.
2. Inspect the implementation of syscount.bt and killsnoop.bt using your favourite text editor.
   1. Inspected with sudo nano /usr/local/share/bpftrace/tools/syscount.bt
   2. Inspected with sudo nano /usr/local/share/bpftrace/tools/killsnoop.bt
3. Are any of the lines outputted by tplist used in the implementation of syscount.bt? If so, which line(s)?
   1. tracepoint:raw\_syscalls:sys\_enter isn’t in the output of tplist
4. tplist uses Python's built-in file IO functions to open a file. Which file does tplist open? Why?
   1. Opens /sys/kernel/debug/tracing/events/anycategory/anyevent/format file
      1. Format file once navigated
5. A common characteristic of bpftrace tools is its use of of the args struct. For example, in killsnoop.bt, args->pid is used to retrieve the pid of the process that initiated the kill system call. Where can we find the fields of the args struct? **Hint**: Explore the directory structure you found in the previous question.
   1. The format file contains the fields of the args struct where it says field:blah blah blah
      1. Example would be field:int common\_pid; for args->pid
6. The Linux kernel code that partially implements the tracepoint for task creation can be found [here](https://elixir.bootlin.com/linux/latest/source/include/trace/events/task.h):
   1. What do you think each line does?
      1. TP\_PROTO is the prototype of the function called by this tracepoint
      2. TP\_ARGS is a list of arguments passed to terminal\_panel if this is a “grapcon\_generator” object
      3. TP\_STRUCT\_\_entry field definitions, these correspond to the fields which are assigned when the trace point is triggered
      4. TP\_fast\_assign statemnents which take the raw argument to the tracepoint (the skb) and set the associated field values (skb len, skb pointer, etc)
      5. TP\_printk which is responsible for using those field values to display a relevant tracing message
   2. How do you think this kernel space code affects the args struct that is available from userspace?
      1. They might not have the same fields for args struct between the available userspace and kernel space
7. Using the trace\_write\_syscall() function as a reference, finish the trace\_open\_syscall() function to make it trace the sys\_open system call. Make use of every field of the args struct in your implementation.
8. Run opensnoop.bt:
   1. What files are being opened when you run ls? Are these what you expect?
      1. It should open the files in their respected directories as expected when ls is ran
         1. The actual files opened isn’t what I expected it doesn’t have the names of the actual files or directories listed in ls for the folder
   2. What files are being opened when you run top? What directory are most of them in? Why?
      1. The files are stat and statm being opened which are all in the proc directory
         1. Stat is a command which gives information about the file and filesystem
         2. Proc directory / file system contains a hierarchy of special files which represent the current state of the kernel
      2. Top is a tool used for displaying system-performance information
   3. Compare the files being opened by bash and static-sh. Is there a significant difference? Why?
   4. What files does 3000shell open? Can you change what files it opens by changing how you create the 3000shell binary?
9. Run execsnoop.bt:
   1. Does every typed command result in one execve system call?
   2. If you type in a non-existent command, does it produce any execve's in bash? What about 3000shell?
   3. When you ssh into the VM you should see many execve calls. Where are they coming from?
10. Run killsnoop.bt:
    1. Are kill system calls happening when you do nothing? Who is sending them?
    2. When you interrupt a command using Ctrl-C or Ctrl-S, is a kill system call generated?
    3. If you hit Ctrl-C at a bash prompt, does it generate a kill system call?
    4. Are all signals being sent via kill system calls? How do you know?
    5. What is a 0 signal used for? Do you see any process sending these signals when you log in or out?
11. Run syscount.bt:
    1. What programs are generating lots of system calls?
    2. What calls are the most common? Use /usr/include/x86\_64-linux-gnu/asm/unistd\_64.h for the names associated with system call numbers on the class VM.
12. Run bashreadline.bt:
    1. What does this program do?
    2. Look at the code for this script. How do you think it works?
    3. If bash is already running, this script won't report the first command that you type. Why?