Question 1
Observation of program execution behaviors shows that many system calls are invoked as part of starting up a program. To examine this start-up behavior, construct a simple program that makes no system calls and analyze it using traceanal. Do all programs exhibit a similar start-up behavior in terms of which system calls are used and their relative sequence?

Empty Program	Ls
INPUT: ./traceanal seq < empty.slog	**INPUT:** ./traceanal seq < ls.slog

OUTPUT:
AAA: 31 invoked system call instances from 13 unique system calls
access 1
  access:openat 1
arch_prctl 2
  arch_prctl:access 1
  arch_prctl:mprotect 1
brk 1
  brk:arch_prctl 1
close 2
  close:arch_prctl 1
  close:openat 1
execve 1
  execve:brk 1
exit_group 1
fstat 2
  fstat:mmap 2
mmap 7
  mmap:close 2
  mmap:mmap 3
  mmap:mprotect 1
  mmap:pread64 1
mprotect 4
  mprotect:mmap 1
  mprotect:mprotect 2
  mprotect:munmap 1
munmap 1
  munmap:exit_group 1
openat 2
  openat:fstat 1
  openat:read 1
pread64 6
  pread64:fstat 1
  pread64:mmap 1
  pread64:pread64 4
read 1
  read:pread64 1
	OUTPUT:
	AAA: 104 invoked system call instances from 22 unique system calls
	access 2
	  access:openat 2
	arch_prctl 2
	  arch_prctl:access 1
	  arch_prctl:mprotect 1
	brk 3
	  brk:arch_prctl 1
	  brk:brk 1
	  brk:openat 1
	close 11
	  close:access 1
	  close:close 1
	  close:exit_group 1
	  close:fstat 1
	  close:ioctl 1
	  close:mmap 1
	  close:openat 5
	execve 1
	  execve:brk 1
	exit_group 1
	fstat 10
	  fstat:getdents64 1
	  fstat:mmap 5
	  fstat:pread64 2
	  fstat:read 1
	  fstat:write 1
	getdents64 2
	  getdents64:close 1
	  getdents64:getdents64 1
	ioctl 2
	  ioctl:ioctl 1
	  ioctl:openat 1
	mmap 27
	  mmap:arch_prctl 1
	  mmap:close 7
	  mmap:mmap 17

	mmap:mprotect 2
	mprotect 9
	  mprotect:mmap 2
	  mprotect:mprotect 6
	  mprotect:munmap 1
	munmap 1
	  munmap:set_tid_address 1
	openat 9
	  openat:fstat 4
	  openat:read 5
	pread64 8
	  pread64:fstat 2
	  pread64:mmap 2
	  pread64:pread64 4
	prlimit64 1
	  prlimit64:statfs 1
	read 7
	  read:close 1
	  read:fstat 3
	  read:pread64 2
	  read:read 1
	rt_sigaction 2
	  rt_sigaction:rt_sigaction 1
	  rt_sigaction:rt_sigprocmask 1
	rt_sigprocmask 1
	  rt_sigprocmask:prlimit64 1
	set_robust_list 1
	  set_robust_list:rt_sigaction 1
	set_tid_address 1
	  set_tid_address:set_robust_list 1
	statfs 2
	  statfs:brk 1
	  statfs:statfs 1
	write 1
	  write:close 1

Answer:
From the table above, we can gather that the startup sequence does run a very similar set of system calls. We can see that the ls function runs greater than or equal to the number of each system call produced from the empty program. However, the sequences detected in the empty program are slightly different. For example, after a quick scan, it can be found that close was followed by arch_prctl in the empty program, but not in ls.

Question 2
Researchers have proposed using the system call sequence of a program as a “signature”for that program as a means to detect if a copy of a program is substituted by an intruder.Investigate the validity of this idea by checking if the signature of different executions of the same program are the same. The particular counts of system calls may vary, but are the sequences similar? What if different command line arguments are used for a command? Is There variation in the sequence? Does the sequence change if the amount of data or duration of execution varies for a program?

Ls 1	Ls 2	Ls 3
INPUT: ./traceanal seq < ls.slog	**INPUT:** ./traceanal seq < ls.slog	**INPUT:** ./traceanal anotherarg seq < ls.slog
OUTPUT:	**OUTPUT:**	**OUTPUT:**
AAA: 104 invoked system call instances from 22 unique system calls	AAA: 104 invoked system call instances from 22 unique system calls	AAA: 104 invoked system call instances from 22 unique system calls
access 2	access 2	access 2
access:openat 2	access:openat 2	access:openat 2
arch_prctl 2	arch_prctl 2	arch_prctl 2
arch_prctl:access 1	arch_prctl:access 1	arch_prctl:access 1
arch_prctl:mprotect 1	arch_prctl:mprotect 1	arch_prctl:mprotect 1
brk 3	brk 3	brk 3
brk:arch_prctl 1	brk:arch_prctl 1	brk:arch_prctl 1
brk:brk 1	brk:brk 1	brk:brk 1
brk:openat 1	brk:openat 1	brk:openat 1
close 11	close 11	close 11
close:access 1	close:access 1	close:access 1
close:close 1	close:close 1	close:close 1
close:exit_group 1	close:exit_group 1	close:exit_group 1
close:fstat 1	close:fstat 1	close:fstat 1
close:ioctl 1	close:ioctl 1	close:ioctl 1
close:mmap 1	close:mmap 1	close:mmap 1
close:openat 5	close:openat 5	close:openat 5
execve 1	execve 1	execve 1
execve:brk 1	execve:brk 1	execve:brk 1
exit_group 1	exit_group 1	exit_group 1
fstat 10	fstat 10	fstat 10
fstat:getdents64 1	fstat:getdents64 1	fstat:getdents64 1
fstat:mmap 5	fstat:mmap 5	fstat:mmap 5
fstat:pread64 2	fstat:pread64 2	fstat:pread64 2
fstat:read 1	fstat:read 1	fstat:read 1
fstat:write 1	fstat:write 1	fstat:write 1
getdents64 2	getdents64 2	getdents64 2
getdents64:close 1	getdents64:close 1	getdents64:close 1
getdents64:getdents64 1	getdents64:getdents64 1	getdents64:getdents64 1
ioctl 2	ioctl 2	ioctl 2

ioctl:ioctl 1	ioctl:ioctl 1	ioctl:ioctl 1
ioctl:openat 1	ioctl:openat 1	ioctl:openat 1
mmap 27	mmap 27	mmap 27
mmap:arch_prctl 1	mmap:arch_prctl 1	mmap:arch_prctl 1
mmap:close 7	mmap:close 7	mmap:close 7
mmap:mmap 17	mmap:mmap 17	mmap:mmap 17
mmap:mprotect 2	mmap:mprotect 2	mmap:mprotect 2
mprotect 9	mprotect 9	mprotect 9
mprotect:mmap 2	mprotect:mmap 2	mprotect:mmap 2
mprotect:mprotect 6	mprotect:mprotect 6	mprotect:mprotect 6
mprotect:munmap 1	mprotect:munmap 1	mprotect:munmap 1
munmap 1	munmap 1	munmap 1
munmap:set_tid_address 1	munmap:set_tid_address 1	munmap:set_tid_address 1
openat 9	openat 9	openat 9
openat:fstat 4	openat:fstat 4	openat:fstat 4
openat:read 5	openat:read 5	openat:read 5
pread64 8	pread64 8	pread64 8
pread64:fstat 2	pread64:fstat 2	pread64:fstat 2
pread64:mmap 2	pread64:mmap 2	pread64:mmap 2
pread64:pread64 4	pread64:pread64 4	pread64:pread64 4
prlimit64 1	prlimit64 1	prlimit64 1
prlimit64:statfs 1	prlimit64:statfs 1	prlimit64:statfs 1
read 7	read 7	read 7
read:close 1	read:close 1	read:close 1
read:fstat 3	read:fstat 3	read:fstat 3
read:pread64 2	read:pread64 2	read:pread64 2
read:read 1	read:read 1	read:read 1
rt_sigaction 2	rt_sigaction 2	rt_sigaction 2
rt_sigaction:rt_sigaction 1	rt_sigaction:rt_sigaction 1	rt_sigaction:rt_sigaction 1
rt_sigaction:rt_sigprocmask 1	rt_sigaction:rt_sigprocmask 1	rt_sigaction:rt_sigprocmask 1
rt_sigprocmask 1	rt_sigprocmask 1	rt_sigprocmask 1
rt_sigprocmask:prlimit64 1	rt_sigprocmask:prlimit64 1	rt_sigprocmask:prlimit64 1
set_robust_list 1	set_robust_list 1	set_robust_list 1
set_robust_list:rt_sigaction 1	set_robust_list:rt_sigaction 1	set_robust_list:rt_sigaction 1
set_tid_address 1	set_tid_address 1	set_tid_address 1
set_tid_address:set_robust_list 1	set_tid_address:set_robust_list 1	set_tid_address:set_robust_list 1
statfs 2	statfs 2	statfs 2
statfs:brk 1	statfs:brk 1	statfs:brk 1
statfs:statfs 1	statfs:statfs 1	statfs:statfs 1
write 1	write 1	write 1
write:close 1	write:close 1	write:close 1

Answer:
From testing we can see that the exact same program will make the exact same system calls every time it is run. It will also make the same system calls with extra input arguments. In all cases we also see order of system calls to be similar. Regardless of these factors, the program runs the same consistently. However, if the data input is changed between execution, this can

affect the output slightly. In this case, the system ran one more system call (105 total) when I analysed a log file produced from “ls ~/” instead of “ls ./”. Also, it was found that programs running the same input data still produced similar system calls even if their runtimes were lightly different.

Question 3
How much variation and commonality do you observe from invocations of different commands? You should try to separate out the start-up behavior common to all commands and the command-specific portion.

Answer:
Apart from the very similar start-up sequences. Different commands such as ls, cd, ifconfig, and others tend to produce additional sequences and varying system call totals. These totals and sequences are identifiable enough to the point that a theoretical program signature could possibly be implemented. This signature would have to be identified as the expected output of an already predefined input. It is important for this input to be predefined because different inputs can cause minor variation in output. If the system call sequences or totals are incorrect. You could be sure the program has been altered.