In Computer System, system call is a way programmer interacts with Operating System(OS). The program requests for services from the kernel of the Operating System it is executed on. During the process of tracing HelloWorld program I have found lots of system calls which is described below:

execve("./a.out….: This system call executes the program named hello.out in our case.

brk(NULL): parameter given in brk will be the end of the data segment which means after the number of a given location of any uninitialized data segment.

arch\_prctl(0x3001 : this system call sets architecture’s specific thread state.

access("/etc/ : Checks user’s permission for files.

openat(AT\_FDCWD, "/mnt/local/lib/tls/libstdc++.so.6", O\_RDONLY|O\_CLOEXEC)….. : Open a files specified at given path. Returns -1 if not found.

fstat: This system call returns the file status pointed by a buffer.

mmap(NULL, 85007, PROT\_READ, MAP\_PRIVATE, 3, 0): Creates a new mapping in the virtual address space of the calling process. Here in our case is null so the kernel will choose a nearby page boundary. It is the most beneficial mapping when Null as the first parameter.

read(3, "\177ELF\2\1\1\3\0\0\0\0\0\0\0\0\3\0>\0\1\0\0\0`\373\10\0\0\0\0\0"..., 832) : Attempts to read 832 bytes from the file descriptor which is 3 into the buffer starting at 177ELF\2\1\1\3\0\0\0\0\0\0\0\0\3\0>\0\1\0\0\0`\373\10\0\0\0\0\0.

Close(3): When the file descriptor holds some file and do not let other processes to use it to prevent the race condition. The close system call is the one that closes the file descriptor and releases the file so another process can reuse it.

lseek: Allows file offset to be set beyond the end of the file.

mprotect: This system call changes the access protection of the calling process’s memory pages.

write: Writes allocated bytes in the buffer which goes to scree. In our case puts the text hello world on screen.

exit\_group: Terminates the calling process and threads.

In java, We found a lot of system calls for accessing libraries and files from different locations in comparison to what we found in C++. There were some new system calls we saw in Java.

readlink("/proc/self/exe", "/usr/lib/jvm/java-1.8.0-openjdk-"..., 4096): Reads the path specified and puts in the buffer specified in parameter which is of size 4096 in our case.

stat("/usr/lib/jvm/java-1.8.0-openjdk-1.8.0.232…: Gets the file status

getgid(): Returns the real group Id of calling process.

getuid(): Returns the real user Id of calling process.

futex(0x7f761c1750e8, FUTEX\_WAKE\_PRIVATE, 2147483647) : It helps in waiting until a certain condition becomes true.

getpid(): Returns the process Id of the calling process.

In Python there were even more system calls then what we found in Java and C++. The total number of system calls were 600 in python which is more than double of the number in Java. We found more new system calls in python then we did in Java.

munmap(0x7f0e23af1000, 85007) = 0 : Maps and unmap the files or devices into memory. Works in a same way as mmap.

set\_tid\_address(0x7f0e23aee910) = 10930 : Sets pointer to thread id.

set\_robust\_list(0x7f0e23aee920, 24) = 0 : Get/Set list of robust futex.

rt\_sigaction(SIGRTMIN, {sa\_handler=0x7f0e2316b9a0, : This system call is used to change the action taken by a process on receipt of a specific signal.

prlimit64( : Puts new limit by replacing old resource limit.

getrandom("\xd7\xc8 : Obtains a series of random numbers.

fcntl(3, F\_GETFD) : Performs the operation described in the parameter in the open file descriptor. In our case it is getting file.

getdents64(3, /\* 0 entries \*/, 32768) : Reads directory referred by open file descriptor into the pointed buffer.

dup(0) : Duplicates the file descriptor.

ioctl(2, TCGETS, 0x7fffdbcdc620) : This system call controls the device.

After analyzing the code, we have found C++ relatively calls fewer files than Java and Python. It has a smaller number of openat, stat, mprotect system calls in its execution. When we go towards Java we can see a lot more of openat command. The number can be approximately 5 times as compared to java. It is reading a lot of files from Java library. And we go towards python we can see lots of openat, stat, readlink, rt\_sigaction, lstat and many more. The system calls are all over the place and they are used a lot in comparison to C++. I think the number of system call has to do with the simplicity of programming language. Java and Python have garbage collection, they have very efficient built-in library that can map an object of different types. In python, object mapping gets even flexible while adding the simplicity in code. We could clearly see in our tracing that python was pulling lots of files from its engine. That must be because of the number of built-in libraries in python that in Java and C++. When we don’t specify what we want and simply write a line to print a string, an interpreter has to pull everything needed that makes a normal program run.

After doing all the tracing I have developed huge respect towards system programmer and electrical engineer. For a simple program, numbers of system calls have to happen. Each call will cycle CPU 100-1000 times to complete its task. Different system call running in different processes and threads. I feel like these things are magic, who thought 500 years ago that we could put a life in non-living things. But today computers are doing the magic. It is a dead body where software as a soul makes it alive.

Importing more libraries than needs definitely slows down the process. Today I was able to see how there is a tradeoff between the simplicity and usability of the computer program. Our main goal should be to decrease the number of lengthy system calls while making the program simple to code so we can enjoy both simplicity and faster processing. There is always a bad side to the good things and vice versa. Python is a lot of people's favorite but it is relatively slow in real-world if the code we wrote is executing only a few times. While the same task can be done relatively fast with C. But we don’t want to write a code in C which will run very few times. There is always a trade-off and it was a good assignment to analyze those aspects.