Lab 2 : Operating System Interfaces and Process Information

Javonda Walker

1.How many system calls?

When I ran option 4 and checked syscalls.txt, I saw about this many calls in total (both

methods combined): openat=4, read=6, write=29, close=4. The library version ends up

doing fewer read()calls because it reads bigger chunks at a time. The low-level version

reads in smaller fixed chunks, so it would usually need more read()s to get the same data.

The exact numbers can vary by machine, but the pattern is the same; fewer, larger reads

with the library and more, smaller reads with raw syscalls.

2.What’s the difference between open() and fopen()?

open() is the low-level system call that talks directly to the kernel and gives you an integer

file descriptor. fopen() is the C library’s higher-level function that uses open() under the

hood but returns a FILE\* and sets up buffering. With fopen() you get helpers like fgets() and

fprintf(), so the code is usually simpler to write.

3.Why does the library method make different calls?

Because of buffering. The stdio layer (fopen/fgets) keeps an in-memory buffer, so it can

read a bigger chunk from the OS and then serve the program multiple lines from that

buffer. That means fewer trips into the kernel (fewer read()syscalls). Also a couple of extra

setup calls like fstat are seen, but overall there are fewer reads and that usually helps

performance for straight sequential reading.

For this kind of sequential file read, the stdio approach tends to make fewer system calls

and can be faster, while the raw syscall approach gives you tighter control over buffer sizes

and exactly how reads/writes happen. In my trace, the stdio path reduced the number of

read() calls, which matches what I expect from its buffering

