Problem 1.1 :

a-)

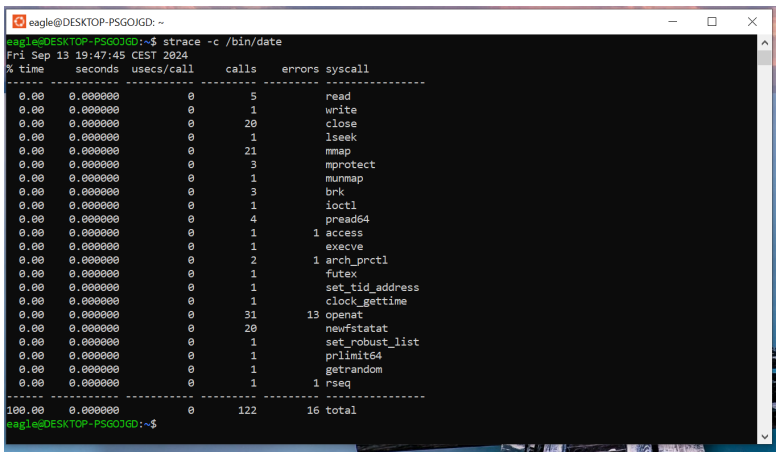

Figure 1: system calls using strace

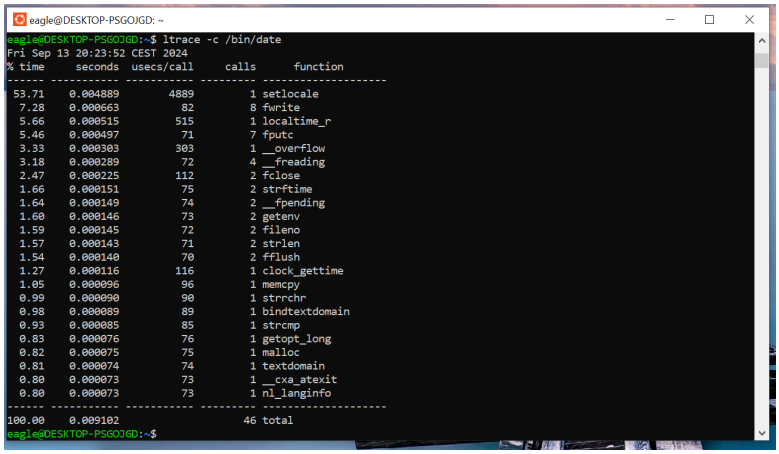

Figure 2: library calls using ltrace

– Strace allows tracing system calls made by a process. -c provide a summary of system calls as seen above. It will output the total number of system calls along with the count for each specific type of call. The total number of system calls after executing /bin/date on my Unix machine is 122 (see Figure 1).
– On the other hand, ltrace allows tracing library function calls made by a process. Similarly to strace, the -c argument provides a summary of library calls made by the process. The total number of library calls after executing /bin/date on my Unix machine is 46 (see Figure 2).

b-)

- **For *strace*** (Figure 1)***:***

  "openat" : 31 calls. Used to open a file or directory relative to a directory file descriptor. similar to the "open" system call but provides more flexibility in handling relative paths.
  "mmap" : 21 calls. maps files or devices into memory. It allows processes to access files as if they are part of the process's memory.
  "close" : 20 calls. Used to close a file descriptor. Upon closing a file descriptor, all associated resources

are released with it, including file locks, and flushes any buffered data to disk (if applicable).

- ***For ltrace*** (Figure 2) ***:***

  "fwrite" : 8 calls. Is a function that belongs to C std::io library. Used to write blocks of data from memory to a file, specifically it's used to write binary data to a file.
  "fputc" : 7 calls. Is a function that belongs to C std::io library. Used to write a single character to a file.
  "_freading" : 4 calls. Is an internal function or can be macro in some cases. It is used internally to determine if the stream is in reading mode.

Problem 1.2 :

a-) The open() system call is used to open a file specified by path and returns a file descriptor if successful. If the file specified by path to open does not exist, the call will fail (No such file or directory), causing -1 to be returned and setting errno to a distinct value.
The close () system call is used to close an open file descriptor. If fildes argument is not a valid file descriptor (e.g., it was never opened or has already been closed). The call will fail (No such file or directory), causing -1 to be returned and setting errno to a distinct value.

b-) The value of errno is the same as before the system call, if this later completes without an error. The value errno updates only when a system call fails, and remains unchanged from its previous state when it's successful.

Problem 1.3 :

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <errno.h>

extern char **environ;

                                        //prints current environment
void print_environment() {
    char **env = environ;
    while (*env) {
        printf("%s\n", *env++);
    }
}

                                //removes a var from environment
int remove_from_environment(const char *name) {
    if (unsetenv(name) != 0) {
        fprintf(stderr, "Error: Failed to remove %s from environment\n", name);
```

```
        return -1;
    }
    return 0;
}

int main(int argc, char *argv[]) {
    int verbose = 0;                    // -v option
    int opt;
    char *remove_var = NULL;

                                //Using getopt to parse options
    while ((opt = getopt(argc, argv, "vu:")) != -1) {
        switch (opt) {
        case 'v':
            verbose = 1;
            break;
        case 'u':
            remove_var = optarg;
            break;
        default:
            fprintf(stderr, "Usage: %s [-v] [-u name] [name=value]... [command [args]...]\n",
            exit(EXIT_FAILURE);
        }
    }

                        // If -u is specified, specified var is removed
    if (remove_var) {
        if (verbose) {
            fprintf(stderr, "Removing variable: %s\n", remove_var);
        }
        if (remove_from_environment(remove_var) != 0) {
            exit(EXIT_FAILURE);
        }
    }

                                // Adding pairs to env
    int i = optind;
    while (i < argc && strchr(argv[i], '=')) {
        if (verbose) {
            fprintf(stderr, "Adding to environment: %s\n", argv[i]);
        }
        if (putenv(argv[i]) != 0) {
            perror("putenv");
            exit(EXIT_FAILURE);
        }
```

```
        i++;
    }

    // If no provided cmd, environment is printed
    if (i == argc) {
        if (verbose) {
            fprintf(stderr, "Printing current environment:\n");
        }
        print_environment();
        return 0;
    }

                        // Execute cmd if provided
    if (verbose) {
        fprintf(stderr, "Executing command: %s\n", argv[i]);
    }
    if (execvp(argv[i], &argv[i]) == -1) {
        perror("execvp");
        exit(EXIT_FAILURE);
    }

    return 0;
}
```