Operating Systems
Constructor University
Dr. Jürgen Schönwälder

Module: CO-562
Date: 2026-09-04
Due: 2026-09-11

Problem Sheet #1

Problem 1.1: *system call errors* (1+1+2 = 4 points)

System call errors are often indicated by returning a special value (usually -1 for system calls returning an int). The reason of the system call failure is stored in the (thread-local) variable `int errno`, declared in `errno.h`.

a) For each of the following system calls, describe a condition that causes it to fail (i.e., a condition that causes -1 to be returned and that sets `errno` to a distinct value).

- `int open(const char *path, int oflag, ...)`
- `int close(int fildes)`
- `ssize_t write(int fd, void* buf, size_t count)`
- `int kill(pid_t pid, int sig)`

b) What is the value of `errno` after a system call completed without an error?

c) Implement a C program that causes these system calls to fail. Implement a generic error printing helper function. Your program should produce the following output on the standard error.

```
open: File exists (errno 17)
close: Bad file descriptor (errno 9)
write: Bad address (errno 14)
kill: Operation not permitted (errno 1)
```

(The error messages and error numbers you get on your computer may differ as they can be system specific.)

Problem 1.2: *simple cat (scat) using library and system calls* (6 points)

Write a C program `scat` (slow cat) copying data from the standard input to the standard output.

a) Implement the data copying loop using the C library functions `getc()`/`putc()` and using the system calls `read()`/`write()`, copying on a single byte in each iteration. Your `scat` program should accept the command line options `-l` and `-s`: The option `-l` selects the C library copy loop while the option `-s` selects the system call copy loop. In case there are multiple options on the command line, the last option wins. If there is neither a `-l` nor a `-s` option, the program uses the C library copy loop.

b) Use your `scat` program to copy a large file to `/dev/null` (a device file that discards all data) and measure the execution times:

```
time ./scat -l < some-large-file > /dev/null
time ./scat -s < some-large-file > /dev/null
```

Repeat the measurements a few times to get stable results. What do you observe? Explain. Use `strace` to investigate the read/write sizes that are used by the two variants of your program. How many read/write calls in total are executed while copying your large file?

c) Implement another copy loop that uses the Linux specific `sendfile()` system call. The `-p` option selects this copy loop. Set the amount of data that is copied in each call of `sendfile()` such that it matches the amount of bytes read and written by the C library copy loop. Measure the execution time:

```
time ./scat -p < some-large-file > /dev/null
```

What do you observe? Explain.

Hand in the source code of your `scat` program and the results of your analysis. Make sure that your program handles *all* error situations appropriately. Use the `getopt()` function of the C library for parsing command line options.