OS 2020 Problem Sheet #1

Problem 1.1: *freshie crash*   (2 points)

A freshmen is learning to program in C. He wrote the following program but it keeps crashing or producing unexpected output. Explain why the program crashes or produces unexpected output.

```c
1   #include <string.h>
2   #include <stdio.h>
3
4   char* strdup(const char *s)
5   {
6       size_t len = strlen(s);
7       char d[len+1];
8       return strncpy(d, s, len+1);
9   }
10
11  int main(int argc, char *argv[])
12  {
13      int i;
14
15      for (i = 1; i < argc; i++) {
16          puts(strdup(argv[i]));
17      }
18
19      return 0;
20  }
```

Problem 1.2: *system call errors*   (2 points)

System call errors are usually indicated by returning a special value (often -1 for system calls that return an int) and by indicating the details in the global 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 sets `errno` to a distinct value).

   - `int open(const char *path, int oflag, ...)`
   - `int close(int fildes)`

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

Problem 1.3: *simple cat (scat) using library and system calls*   (6 points)

Write a program `scat` (simple cat or slow cat) that copies 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.