Operating Systems
Constructor University
Dr. Jürgen Schönwälder

Module: CO-562
Date: 2025-09-05
Due: 2025-09-12

Problem Sheet #1

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

System call errors are usually indicated by returning a special value (usually -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 that 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 1.2: *repeat a string many times* (3+3+1+1 = 8 points)

The C library provides buffered I/O primitives on file streams. There are three buffering modes:

- In unbuffered mode (`_IONBF`), no buffering is used.
- In line buffered mode (`_IOLBF`) buffers are flushed whenever a newline character appears or the buffer is full or a flush is forced (via fflush()) or the file stream is closed.
- In block buffered mode (`_IOFBF`) buffers are flushed if they are full or a flush is forced (via fflush()) or the file stream is closed.

The function `setvbuf()` can be used to control buffering of a file stream.

a) Write a C program `repeat` writing a message in a loop to standard output. The message is printed by calling putchar() for each character in a loop. The command syntax is:

```
repeat [-r count] [-n] [msg] ...
```

The `-r` option can be used to define how many times the message is written to the standard output. The default of `n` is 1. A newline is printed after each iteration writing the message. This can be turned off using the `-n` option.

Use `getopt()` to parse the command line arguments.

Some example invocations:

```
$ ./repeat -r 2 "hello world"
hello world
hello world
$ ./repeat -r 0 "nothing"
$ ./repeat -r 3 a
a
a
a
$ ./repeat -r 3 -n a
aaa$
```

b) Add two additional command line options to your program controling the buffering of the standard output file stream.

```
repeat [-b size] [-l size] [-u] [-r count] [-n] [msg] ...
```

The option `-u` makes the standard output unbuffered. The option `-l n` makes the standard output line buffered with a buffer of size `n` bytes and the option `-b n` makes the standard output fully buffered with a buffer of size `n` bytes. If multiple buffering options appear on the command line, the last option wins. For example, the invocation

```
$ ./repeat -l 512 -b 1024 -u -r 1 hello
```

will print `hello` once using no buffering.

c) Verify that your program works as expected using `ltrace` and `strace`.

d) Perform a series of measurements to understand the effect of the buffering settings on the performance of your program. Vary the buffer sizes in meaningful ways.

Hand in the source code of your `repeat` program and the results of your analysis. Make sure that your program handles *all* error situations appropriately.