OS 2022 Problem Sheet #12

Problem 12.1: *open files and file updates and meta data changes* (1+1+1 = 3 points)

```
/*
 * catloop.c --
 */

#define _POSIX_C_SOURCE 201112L

#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/types.h>
#include <sys/wait.h>
#include <fcntl.h>

int main(int argc, char *argv[])
{
    int fd;
    pid_t pid;

    if (argc != 2) {
        fprintf(stderr, "catloop: missing 'file' argument\n");
        return EXIT_FAILURE;
    }

    fd = open(argv[1], O_RDONLY);
    if (fd == -1) {
        perror("catloop: open");
        return EXIT_FAILURE;
    }

    while (1) {
        pid = fork();
        if (pid == -1) {
            perror("catloop: fork");
            (void) close(fd);
            return EXIT_FAILURE;
        }
        if (pid == 0) {
            char c;
            (void) lseek(fd, 0, SEEK_SET);
            while (read(fd, &c, 1) == 1) {
                (void) write(STDOUT_FILENO, &c, 1);
            }
            (void) close(fd);
            return EXIT_SUCCESS;
        }
        (void) waitpid(pid, NULL, 0);
        sleep(1);
    }

    (void) close(fd);
    return EXIT_SUCCESS;
}
```

Save the source code shown above into the file `catloop.c`. Compile the C code to produce the executable `catloop` and afterwards execute the following shell commands on a Linux system (the behavior may be different on a Windows system):

```
$ rm -f foo
$ touch foo
$ ./catloop foo &
$ echo -n "hello " > foo
```

Answer the following questions, always with the same initial setup.

a) Describe what the program doing.

b) What happens if you append content to the file `foo` while `catloop` is running?

```
$ echo "world" >> foo
```

What happens if you truncate the file `foo` while `catloop` is running?

```
$ truncate -s 0 foo
```

Discuss the advantages and disadvantages of the behavior you have observed in the previous step. Could there be other file system update semantics?

c) What happens if you change the permissions of the file `foo` while `catloop` is running?

```
$ chmod 0 foo
$ ls -l foo
```

What happens if you remove the file `foo` while `catloop` is running?

```
$ rm -f foo
```

What are possible implications of this behavior?

Problem 12.2: *file system permissions* (1+1+1+1 = 4 points)

Unix file system objects have basic permissions associated with (i) the file owner, (ii) the file's group members, and (iii) everybody else with access to the file system. Answer the following questions:

a) Who has which access permissions for the file `foo`?

```
$ ls -l foo
-rwxrw-r-- 1 alice co-562 0 Nov 30 14:53 foo
```

Who has which access permissions for the directory `bar`?

```
$ ls -ld bar
drwx-wx--- 2 alice co-562 4096 Nov 30 14:56 bar
```

b) Can `bob`, a member of the group `co-562`, read the content of the directory `bar`? Can `bob` create a file in the directory `bar`? Explain.

c) A regular user (with a umask of 0022) executes the following shell command. What are the file permissions of the new file and who is the owner of the file? Explain.

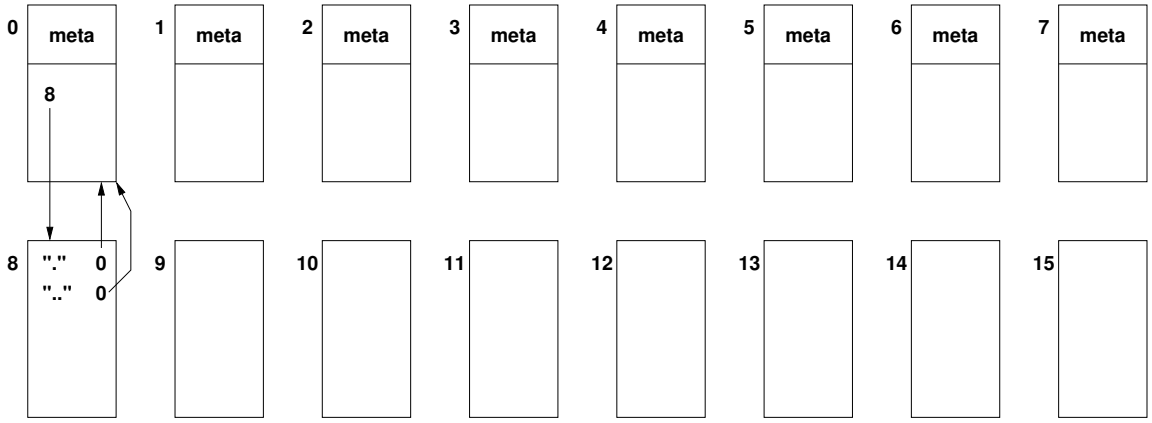
```
$ rm -f world
$ sudo echo hello > world
```

d) What is the meaning of the following access permissions?

```
$ ls -l /usr/bin/sudo
-rwsr-xr-x 1 root root 182600 Feb 27  2021 /usr/bin/sudo
$ ls -ld /tmp
drwxrwxrwt 8 root root 700 Nov 17 13:02 /tmp
$ ls -ld /usr/local/bin
drwxrwsr-x 2 root staff 4096 Jul 30  2018 /usr/local/bin/
```

Problem 12.3: *file system updates* (3 points)

Consider a small file system residing on a block device with 16 blocks (numbered 0..15). All blocks have the same size. The first 8 blocks (0...7) are reserved for inodes. An inode stores file attributes (metadata) of a file system object as well as references to data blocks. The remaining blocks (8...15) are reserved for storing data or directory information, they are called data blocks, or short dnodes. The root directory is always found in inode 0. Below is a figure visualizing the situation right after initializing the file system.



In the following, we focus on the block references that the file system maintains. The initial situation can be described as follows:

```
inode 0: 8                        // inode 0 refers to dnode(s) 8
inode i: undef                    // inode i has undefined content (i in {1..7})
dnode 8: {(".", 0), ("..", 0)}    // dnode 8 has 2 directory entries (to inode 0)
dnode i: undef                    // dnode i has undefined content (i in {9..15})
```

We are now making a number of changes to the file system. In the following steps, write down (using the notation shown above) which inodes and/or dnodes are updated. In each step, write down only the blocks that actually change. If you need to allocate so far unused inodes or dnodes, choose lower numbered unused inodes or dnodes first.

a) `mkdir /a`

b) `mkdir /a/b`

c) `echo "hello" > /a/b/c`

d) `mv /a/b/c /a`

e) `ln /a/c /a/b/d`

f) `ln -s /a/b/d /e`