Operating Systems – spring 2024
Tutorial-Assignment 10

Instructor: Hans P. Reiser

Submission Deadline: Monday, March 25, 2024 – 23:59

A new assignment will be published every week. It must be completed before its submission deadline (late policy for programming assignments: up to two days, 10% penalty/day)

Lab Exercisess are theory and programming exercises discussed in the lab class. They are not graded, but should help you solve the graded questions and prepare for the final exam. Make sure to read and think about possible solutions before the lab class.

T-Questions are theory homework assignments and need to be answered directly on Canvas (quiz).

P-Questions are programming assignments. Download the provided template from Canvas. Do not fiddle with the compiler flags. Upload your solution as a single zip file on canvas.

The name of the zip file has to be "assignment<XX>-<LOGIN1>-<LOGIN2>.zip", where <XX> is the assignment number and <LOGIN1>, <LOGIN2> are your (no)skel login names. In each source code file, put your group number (if applicable) and names of all group members.

The topic of this assignment is the POSIX API for file and directory access.

Lab 10.1: File System Implementation

a. What are hard links?

b. What are symbolic links?

c. Suppose you have created a file f, a hard link h to the same file, and a symbolic link s to f. What happens if you rename f to g? Is the file still accessible via the hard link h? How about the symbolic link s?

d. Would the same be true if you had copied f to g first and then removed f?

e. What happens if you now create a new file f?

f. How can directories be implemented (in the OS)? What information is stored in them?

g. Which of the following data are typically stored in an inode: (a) filename, (b) name of containing directory, (c) file size, (d) file type, (e) number of symbolic links to the file, (f) name/location of symbolic links to the file, (g) number of hard links to the file, (h) name/location of hard links to the file, (i) access rights, (j) timestamps (last access, last modification), (k) file contents, (l) list of blocks occupied by the file?

   For each item state whether it is required or optional. For items not stored in inodes, state where the information is stored (if at all).

Lab 10.2: Accessing Files and Directories

a. What is the difference between an absolute and a relative path name?

b. What are the basic methods for accessing a file?

c. In Linux and Windows, random access on files is implemented via a special system call that moves the "current position pointer" associated with a file to a given position in the file. What are the names of these system calls in Windows and Linux?

d. You can use the `seek` system call to create sparse files: If you seek to a position beyond the end of a file (and if the file system implementation supports sparse files), then the skipped part may remain as a 'hole' without allocated disk storage. Verify this with a short sample program that creates a new file, seeks the position of 1 GiB after the start, writes a single byte, closes the file, and then checks the file size (total size, and block size on disk) using the stat system call.

e. Discuss alternative random access implementations without such a system call.

f. What system calls do you need to list the files in a directory in Linux?

Lab 10.3: Open Files

a. Discuss the in-kernel data structures that are required to allow for a Unix-like handling of open files.

T-Question 10.1: Files

a. *Sparse* files have unallocated holes, which correspond to zeros (i.e., the holes do not occupy space on disk; when read, they read as bytes with value zero). When a write is performed to an offset in a hole, a new block is allocated that can hold the written data.

Find out how can you determine if a file is a sparse file (i.e., it has “holes”), using the `stat` command line tool (which uses the `stat` system call). Hint: You are expected to research the answer to this question on your own. Take a look at the man page of `stat(1)`/`stat(2)`/`stat(3)` and the fields in `struct stat`.

Sample files are on noskel in `/home/sty24/A10/files`

Indicate if each of the files is a sparse file (yes) or not (no):

yes	no	
☐	☐	`out1.txt`
☐	☐	`out2.txt`
☐	☐	`out3.txt`
☐	☐	`out4.txt`

2 T-pt

b. Which of the following statements about ACLs in file systems are true? **2 T-pt**

true	false	
☐	☐	ACLs are file access permissions that are stored as a linked list in a configuration file.
☐	☐	An ACL can provide better fine granular control over file permissions than the standard Unix file permission model.
☐	☐	A typical ACL entry consists of a user or group name, followed by a set of permissions.
☐	☐	With ACLs, the security policy that defines which user can access which file is centrally controlled by a security policy administrator.

c. Assume you have to following permissions on a set of files: **3 T-pt**

```
$ ls -lad folder folder/*
drwxrwx---. 2 hansr users 43 Mar 12 17:26 folder
-rw-r--r--. 1 hansr users  6 Mar 12 17:24 folder/example.txt
-rw--w--w-. 1 hansr users  6 Mar 12 17:26 folder/example2.txt
```

Can the user `anna`, in group `users`, perform the following actions (assuming that `anna` can access everything between the root of the file system and `folder`, and that no additional ACLs are used)?

true	false	
☐	☐	List the contents of `folder`
☐	☐	Append data at the end of the file `example2.txt`
☐	☐	Delete the file `example.txt`
☐	☐	Execute `folder` as program in a child process
☐	☐	Read the content of the file `example.txt`
☐	☐	Read the meta data (file size, access time stamps, etc.) of `example.txt`

d. Assume you execute the sequence of commands below on a shell. What will be the link count of `test.txt`, `b.txt`, `c.txt`, and `link.txt` (answer with -1 if not applicable/file does not exist)? **2 T-pt**

```
echo "Hello" > test.txt
ln -s test.txt b.txt
ln test.txt c.txt
ln c.txt link.txt
rm c.txt
```

e. After execution of the commands of the previous question, you execute the two commands below. What will be the content of `link.txt` and `b.txt`? **1 T-pt**

```
rm test.txt
echo "Bye" > test.txt
```

P-Question 10.1: Directory Listing

Download the template **p1** for this assignment from Canvas. You may only modify and upload the file `ls.c`.

We expect your solution to compile without errors and run on a Linux system (Intel x86-64), using the compiler flags in the template.

In this question you will write a program that prints a directory listing just like the 'ls' and 'dir' commands in Linux and Windows respectively.

a. Write a function that prints a listing of all files in the directory specified by `path` (ignore the `recursive` parameter for now). Your function should perform the following tasks: **4 P-pt**

- Iterate over all files in the given directory using the `opendir()`, `readdir()`, and `closedir()` system calls.[1]
- Skip all hidden entries (i.e., file names that start with '.')
- For each file, retrieve the size in bytes and the type of the file using `lstat()` or `fstatat()`.
- Print the gathered information via the template's `_printLine()` function. You shall print the complete file name with the path (if path is "p1" and readdir returns a name "ls.c", you shall print "p1/ls.c")
- The `typestr` parameter to `_printLine` shall be "/" for directories, "|" for pipes, "*" for executable programs. For symbolic links, it should be the string " -> " and the target file name of the symbolic link.
- Return 0 on success, -1 otherwise (in case of any error).

```
int list(const char* path, int recursive);
```

b. Extend your function from part (a) such that it recursively descends into subdirectories if the parameter `recursive` is not 0. **2 P-pt**

- Expected output when run on noskel, with directory "/home/sty24/A10/files":

```
         size  name and type
1099511628800  /home/sty24/A10/files/out3.txt
          512  /home/sty24/A10/files/out1.txt
         4096  /home/sty24/A10/files/subdirectory/
          512  /home/sty24/A10/files/subdirectory/hardlink
           11  /home/sty24/A10/files/subdirectory/symlink2 -> ../out1.txt
         4096  /home/sty24/A10/files/subdirectory/deep/
            1  /home/sty24/A10/files/subdirectory/deep/loop -> .
            0  /home/sty24/A10/files/testpipe|
        45056  /home/sty24/A10/files/out2.txt*
            0  /home/sty24/A10/files/out4.txt
            8  /home/sty24/A10/files/symlink -> out1.txt
```

[1] On Linux, in fact you use the POSIX library function `readdir()`, which internally uses the `getdents()` system call.

P-Question 10.2: File Copy

Download the template **p2** for this assignment from Canvas. You may only modify and upload the file `copy.c`. In this question you will write a program that copies a file.

The template contains a parser for command line arguments that accepts command lines in the form '-b <n> <source> <destination>'. The parser puts all arguments into a `CopyArgs` structure.

The `-b` (blocksize) parameter is optional (default value 4096) and specifies the number of bytes that you should read/write with a single system call.

Example: `./copy -b 1024 hello.txt out.txt` with a file `hello.txt` containing the string '`Hello world!`' should result in a file `out.txt` containing '`Hello world!`'.

a. Write a function that performs the actual file copy. Your function should fulfill the following requirements: **2 P-pt**

- Performs the copy using the `open()`, `read()`, `write()`, and `close()` system calls, using the specified block size per system call.
- Fails if the destination file already exists.
- Returns 0 on success, -1 otherwise (on all errors).

```
int doCopy(CopyArgs *args);
```

b. Extend the function such that it produces sparse files: If all bytes of a block are 0, your function shall not write the block to the target file, but instead skip it with a `lseek()` system call.
In addition, make sure that after copying the file, the access permissions of the file are the same as those of the source file (if possible). **2 P-pt**

Total:
10 T-pt
10 P-pt