Operating System
Blen Daniel Assefa

OS 2021 Problem Sheet #11

Problem 11.1: open files and file updates and metadata changes

a) The program is being opened by catloop and is infinitely executing the following three things:

- It forks itself, and waits for its child to die. After it dies, the program sleeps for 1 second. If the parent leaves the while loop, using fd from open() it will close the program and exit.

- The forked child executes the code in the conditional. It declares a char c to be a single byte-sized buffer. It sets the fille offset to 0(beginning of the file). Then it reads characters one by one until EOF or some kind of other error occurs. For each read(), a write() call is made to write to STDOUT_FILENO. After leaving the while loop the child closes and exits.

- Catloop gets executed with foo(empty file created by touch). It will attempt to read from foo, but wont be able to because it will run into EOF, so it will wait until the file has content inside of it. Using echo it will write “hello ” in foo. Then the program will start reading “hello “ from foo and it will output it to the screen every second.

b) - After we open the file in the next while loop the program will read “hello world\n” from foo. After truncating the file to size 0, catloop will create a child process and it will read from the empty file. The program will run into EOF and wont output anything. The program will continue to create and wait for its child processes to die every second.

- Advantages of this behavior are:

The contents of files can be updated while there are still open file descriptors by running programs. This allows the possibility to add or remove content from the file and let the programs continue execution. Also when two processes need a large file, both of them can continue execution, given that they don’t effect the other functions.

- Disadvantages of this behavior are:

There is a possibility when performing write() system calls for the system call to be finished before read() system call is being performed. One particular case would be when the content being appended to the file exceeds system limits or the write() tries to write a lot of content. Therefore while accessing the file contents, the information won’t reflect the correct state. This would lead to loss of data or incorrect future read() calls on the file due to the contents being NULL terminated.

c) By changing the permissions of foo while catloop is running, the program will still continue to read the contents of foo. Because of permissions being checked when a file is opened. This means that if a file is opened and then its permissions are

modified after its opening, in this case using chmod, there is still a possibility of reading with that open file descriptor.

- By removing the file foo while catloop is running, the program will still continue to read the contents of foo successfully.

- One possible implication would be:
If we chose to do such changes(removing a file mid execution) we need to double-check if all processes that are using that specific file, and then remove their instruction or the files. This can be dangerous and cause multiple errors.

Problem 11.2: file system permissions

a) `$ ls -l foo`
- User alice can read, can modify, and can execute the file.
- Group co-526 users can read and can modify, but cannot execute the file.
- Other users can only read the file.
`$ ls -ld bar`
- User alice can list the contents, can create and delete files and subdirectories, can rename, can modify the directory attributes and can enter the directory.
- Group co-526 users can create and delete files and subdirectories, can rename, can modify the directory attributes and can enter the directory.
- Other users do not have any of the previously mentioned permissions, therefore they cant do anything.

b) Bob, who is a member of the group co-526 cannot read the contents of the directory bar, since members of the group do not have permission to read. On the other hand, bob can create a file inside the directory.

c) A regular user with umask of 0022 means that files created from that user will have permissions set to 644 (-rw-r--r--). This means the files will be readable and modifiable, but not executable by default by the user. The files will be readable, but not modifiable and not executable by the group the user belongs to or other users in the system. The owner of the file will be the user and the file will be in group ownership of the user's group.

d) Normally, the bit permissions are set to -rwxr-xr-x which means the owner of the file can read, modify and execute the file, while the users of the group belong to (root) and other users can only run and execute the file. Because this file has also special file permissions, in this case the setuid bit is set, the executable bit is replaced by a s. In this case, the executable bit is set, because the setuid bit is s. This means that when the executable is run, it does not run with the privileges of the user who executed it, but rather with the privileges of the executable's owner.

Problem 11.3: index node file system

a) There are 2^{32} blocks and each block has a size of 128 bytes.

Therefore the largest possible file system size will be
$2^{32} * 128 = 239.239 = 549755813888\ bytes,\ equivalent\ to\ 512GB\ or\ 2^{9}GB.$

b) The file system uses 64 bytes out of the 128 bytes to store file attributes. Therefore there are still 64 bytes left for direct data blocks and indirect indexes to data blocks. These 64 bytes are divided into 13 direct data blocks (52 bytes) and 3 indirect indexes (12 bytes). The 13 direct pointers will point to $13 * 2^{7}$ bytes of data. The single indirect index will point to 2^{12} bytes of data. The double indirect index will point to 2^{17} bytes of data. The triple indirect index will point to 2^{22} bytes of data. The maximum file size will therefore be:
$13 * 2^{7} + 2^{12} + 2^{17} + 2^{22} = 4331136\ bytes,\ or\ approximately\ 4.13\ megabytes.$

c) In total, 3 block reads would be needed. The inode, the block pointed to by the double indirect index, the block pointed to by the single indirect index of the block pointed to by the double indirect index, and the direct block which contains the initial position of the offset.