Files report

copy

I wrote with_retry utils for both read and write, so they automatically retry when EINTR happens. I used this for the rest of the programs too. I could have created a shared file, and linked and reused it, but I'm lazy. Also the with_retry call signature gives a warning because the write requires the buffer to be const, but I still decided to keep this.

I later realized that I should have done the retrying for close() too, but it's too late.

This program just reads the given file into a buffer, and writes however much it read to the output file. That's it.

truncate

Pretty straightforward. At the end I just take a buffer of size 20 because I know that the file shouldn't be longer than that (actually should be 10). I read the file into the buffer, at the end of the buffer I set '\0' to print it as a string.

reverse

I seek to the end of the destination, see how many bytes it has. Then I read one byte, print it, lseek -2, because the read incremented the pointer. I could have read a batch, printed it in the reverse order, then repeated, that would be more efficient.

log

I checked the system, the max pid is 4194304, which is 7 digits long. I assume the input size of the user is max 100 long. I create a buffer of size 113. Because the string "PID=1234567:" has length 12, and at the end snprintf will add '\0', so I need 13. I read from file descriptor 0, by size 100, and save it after my string in the buffer, making sure '\0' is rewritten.

Then because I know how many bytes I've read, I add 12 to it, and write it at the end of my file.

sparse

Nothing special here. I left the required comment.

overwrite

I lseek all around the place here.

First I create a buffer of size 5, because I know that the string "10\n" has 3 chars, and '\0' needs to be appended by snprintf (yes I should have allocated buffer of size 4). I write the buffer to the file, for all 10 numbers. I could have done all this in one write. But it seemed the task wanted me not to.

I close the file, then I open it in readonly mode. I read each byte one by one, when I find '4', I call the rewrite function.

Rewrite first does lseek with SEEK_CUR, to find how at which position '4' + 1 is, then does lseek with SEEK_END to see the whole file size. Subtracts to see how many bytes there are after '4'. Since at this point the pointer is at the end of the file, lseeks again with SEEK_CUR to -remaining position to go back to '4' + 1. Reads the whole remaining bytes into a buffer into a malloced memory (because I wanted this program to be somewhat dynamic). Then goes back to '4' by doing SEEK_CUR with -remaining - 1. Writes "100" at that place, and writes all the contents of the buffer after it. The '\n' comes with it conveniently.

compare

Pretty simple. Reads both files by some chunk size. If both of the reads return 0, it means both files have reached the end simultaneously, so they're identical. Otherwise loops to the maximum one (if we read equal amount from both, maximum will be equal to both). If any of the files returned less bytes than the other one, return current index. Or if we find a mismatch of the characters, return the index.

Github repo: https://github.com/VaheHayrapetyan24/CS331_operating_systems