System-Level I/O

Annalise Tarhan

October 6, 2020

10 Homework Problems

10.6 What is the output of the following program?

fd2 = 4 0-2 are for stdin, stdout, and stderr. fd1 is assigned 3 for foo.txt and fd2 is initially assigned 4 for bar.txt. When fd2 is closed, 4 is released and reassigned to fd2 for baz.txt.

10.7 Modify the cpfile program so that it uses the RIO functions to copy standard input to standard output, MAXBUF bytes at a time.

No need to use a buffer here, since copying to the buffer and then to output all at once is just an extra step.

```
int main(int argc, char **argv) {
        int n;
        char buf[MAXBUF];

        while(
        (n = rio_readn(STDIN_FILENO, buf, MAXBUF)) != 0)
        {
                rio_writen(STDOUT_FILENO, buf, n);
        }
}
```

10.8 Write a version of the statcheck program called fstatcheck that takes a descriptor number on the command line rather than a filename.

```
int main (int argc, char **argv) {
        struct stat stat;
        char *type, *readok;

        fstat(atoi(argv[1]), &stat);
        if (S_ISREG(stat.st_mode))
        type = "regular";
        else if (S_ISDIR(stat.st_mode))
        type = "directory";
        else
        type = "other";
        if ((stat.st_mode & S_IRUSR))
        readok = "yes";
        else
        readok = "no";

        printf("type: %s, read: %s\n", type, readok);
        exit(0);
}
```

10.9 Consider the invocation of fstatcheck: fstatcheck 3 < foo.txt, which fails with a "bad file descriptor." Given this behavior, fill in the pseudocode that the shell must be executing between the fork and execve calls.

The shell seems to be closing foo.txt before calling execve.

```
if (Fork() == 0) {
        int fd = open("foo.txt", O_RDONLY, 0);
        dup2(fd, STDIN_FILENO);
        close(fd);
        execve("fstatcheck", argv, envp);
}
```

10.10 Modify the cpfile so that it takes an optional command-line argument infile. If infile is given, then copy infile to standard output; otherwise, copy standard input to standard output as before. In both cases, use the original copy loop.

```
int main(int argc, char **argv) {
        int n;
        rio_t rio;
        char buf[MAXLINE];

        if (argc == 2) {
                int fd = open(argv[1], O_RDONLY, 0);
                dup2(fd, STDIN_FILENO);
                close(fd);
        }

        rio_readinitb(&rio, STDIN_FILENO);
        while((
        n = rio_readlineb(&rio, buf, MAXLINE)) != 0)
        {
                rio_writen(STDOUT_FILENO, buf, n);
        }
}
```