Code Review

Author Name: Maddy Boyd

Reviewer(s) Name: Preston Duffield

Date: 5/5/2023

Project Name: Microshell

Reviewed File(s): builtin.c

## Code Review

The reviewer shall use the below table to document code defects and suggestions for improvement. Severity records how serious the infraction is: high (H), medium (M) or low (L). The first row (after the header) is an example. Remove it before submission. Check the appendix for a suggested C code review checklist.

|  |  |  |  |
| --- | --- | --- | --- |
| Function name | Line number | Severity | Brief Description |
| exitProgram pwd cd cp touch | ~ | L | Missing above function comments |
| ls | 103 | M | There are doubled up parenthesis, I believe the line should be `while ((d = readdir(direct)) != NULL)` |
| pwd ls | 78  79  110  114 | L | The “magic” number 100 is used multiple times, pull out to a const, or use the c language buffer size const. |
| ls cp touch | ~ | H | These functions are not part of the built in commands and can be forked to another process. In other words, these do not need to be here. |
| exitProgram | 71 | M | exit(args[1]) is incorrect because args[1] is a char\* and not an integer. Convert the string to an integer before passing it to exit `exit(atoi(args[1]));` |

The reviewer can add any other notes they have in the below space.

There is a typo in the cd function on line 90. Replace chdir{argcp[1]); with chdir(args[1]);.  
Also you should inititalize pointers like f1 and f2 to NULL:

FILE \*f1 = NULL, \*f2 = NULL;

And check for NULL pointers after fopen calls in cp function

f1 = fopen(args[1], "r");

if (f1 == NULL) {

perror("fopen");

return;

}

f2 = fopen(args[2], "w");

if (f2 == NULL) {

perror("fopen");

fclose(f1);

return;

}

Great work!

## Plan for improvement (to be written by the code author)

## Appendix – C Code Review Checklist

1. Commenting:  top of file, start of function, code that needs an explanation
2. Style is consistent and follows style guidelines
3. No redundant, dead, commented out, unused code & variables
4. Conditional expressions evaluate to a Boolean value; no assignments
5. Parentheses used to avoid operator precedence confusion
6. All switch statements have a default clause; preferably an error trap
7. Single point of exit from each function
8. Loop entry and exit conditions correct; minimum continue/break complexity
9. Conditionals should be minimally nested (generally only one or two deep)
10. Are "magic numbers" avoided? (use named constants rather than numbers)
11. Variables have well-chosen names and are initialized at definition
12. Input parameter checking is done
13. Error handling for function returns is appropriate
14. Null pointers, division by zero, null strings, boundary conditions handled
15. Buffer overflow safety (bound checking, avoid unsafe string operations)
16. Large arrays are dynamically allocated on the heap.
17. Pointer variables are named in a consistent fashion.
18. Pointers are initialized to NULL.
19. Pointers are tested for NULL before being referenced.
20. Dynamically allocated memory is deallocated when no longer needed.
21. Does the code match the detailed design (correct functionality)?
22. Is the code as simple, obvious, and easy to review as possible?