Minishell Bug Fixes - Complete Conversation Summary

Overview

This document summarizes the work done to fix critical bugs in the minishell project, preparing it for evaluation according to the 6/11/2021 minishell evaluation sheet.

Initial State

- Repository needed to be cleaned to "project-only" state
- Multiple critical bugs identified:
  1. Double free with semicolon-separated commands (`ls; echo $?` crashes)
  2. Double free in `export` command (`export TEST=42` crashes)
  3. Heredoc produces no output (`cat << EOF` produces nothing)
  4. Quote expansion order incorrect (surprise tests fail)
  5. Exit code after semicolons incorrect (`/bin/ls /doesnotexist; echo $?` prints 0 instead of 1)
  6. PATH unset behavior incorrect

Fixes Implemented

1. Fixed Double Free with Semicolon-Separated Commands

Problem: `ls; echo $?` crashes with double free error.

Root Cause: The parser was freeing `args` and `ori_args` arrays multiple times across different execution paths.

Solution: - Modified `process_quotes_and_exec` in `src/parser/parser_helpers.c` to split command line into segments based on semicolons - Each segment gets its own duplicated `args` and `ori_args` arrays - Ensures independent memory management for each command segment

Files Modified: - `src/parser/parser_helpers.c`: Refactored `process_quotes_and_exec` to extract `process_last_segment` and `process_semicolon_segment` helper functions

Test Cases: - `ls; pwd` - `ls; echo hi; /bin/ls` - `false; echo $?`

2. Fixed Double Free in Export Command

Problem: `export TEST=42` crashes with double free error.

Root Cause: `ft_setenv_envp` and `strarradd_take` were freeing the old `envp` array, but the calling code was also attempting to free it.

Solution: - Modified `handle_env_update` to check if `envp` array was actually replaced before attempting to free the old one - Updated `one_commnad` to only free `old_envp` if it's different from `new_envp` and the command is not a builtin

Files Modified: - `src/cmd/cmd_export_update.c` - `src/pipe/pipe_helpers.c`

Test Cases: - `export TEST=42` - `export TEST=43` - `env | grep TEST` - `unset TEST`

3. Fixed Heredoc No Output

Problem: `cat << EOF` produces nothing.

Root Cause: 1. When stdin was piped, the main input reader (`read_chunks` via `read_line_fd`) would consume all available input in large chunks, leaving nothing for the heredoc loop 2. Heredoc content was not being attached to the command's stdin in the child process

Solution: 1. Implemented a static buffer (`g_remainder`) in `src/core/line_reader.c` to store unconsumed data from `read` calls, allowing `read_line_fd` to correctly read line by line 2. Modified `get_heredoc_line` in `src/redirection/heredoc_utils.c` to use the buffered `read_line_fd` 3. Moved heredoc processing (`process_heredoc_during_parsing`) to occur synchronously during the parsing phase (in `src/parser/parser.c`), before the main command execution loop 4. Added heredoc attachment logic in child execution paths: - `exec_child` (for piped commands) - `exec_proc` (for non-piped external commands) - `one_commnad` (for non-piped builtins)

Files Modified: - `src/core/line_reader.c`: Added static buffer `g_remainder` for line-by-line reading - `src/redirection/heredoc_utils.c`: Modified `get_heredoc_line` to use buffered reader - `src/parser/parser.c`: Added `process_heredoc_during_parsing` to run heredoc synchronously during parsing - `src/pipe/pipe_core.c`: Added heredoc attachment in `exec_child` - `src/cmd/cmd_utils.c`: Added heredoc attachment in `exec_proc` - `src/pipe/pipe_helpers.c`: Added heredoc attachment in `one_commnad` - `src/redirection/exec_redir.c`: Removed `cleanup_heredoc_file` call and `unlink(".temp")` from parent

Test Cases: - `cat << EOF\nhello\nEOF` (should print `hello`) - `cat << EOF | cat\nhello\nEOF` (should print `hello`)

4. Fixed Quote Expansion Order

Problem: - `echo "'$USER'"` was printing `foo` instead of `'foo'` - `echo '"$USER"'` was printing `$USER` instead of `"$USER"`

Root Cause: The quote removal logic in `process_char_with_lookahead` was removing all quotes, including literal quotes inside other quotes.

Solution: - Modified `process_char_with_lookahead` in `src/parser/parser_quotes_helpers.c` to only remove quotes that act as delimiters at the outermost level, preserving literal quotes - Added `is_inside_double_quotes` helper in `src/parser/parser_expand_scan.c` to correctly handle variable expansion inside double quotes

Files Modified: - `src/parser/parser_quotes_helpers.c`: Modified `process_char_with_lookahead` to preserve nested quotes - `src/parser/parser_expand_scan.c`: Added `is_inside_double_quotes` helper and updated `process_envvar` to handle quote context correctly

Test Cases: - `set USER=foo` (or `export USER=foo`) - `echo "'$USER'"` (should print `'foo'`) - `echo '"$USER"'` (should print `"$USER"`) - `echo '$USER'` (should print `$USER`) - `echo "$USER"` (should print `foo`)

5. Fixed Exit Code After Semicolons (IN PROGRESS)

Problem: `/bin/ls /doesnotexist; echo $?` prints `0` instead of `1` (or `2`).

Root Cause: 1. `$?` was being expanded during the initial parsing phase (`process_parser_input`), before any commands in a semicolon-separated sequence had actually executed 2. This meant `$?` always reflected the shell's initial exit status (usually 0) or the status of the very last command, not the immediately preceding one

Solution Attempted: - Modified `process_envvar` in `src/parser/parser_expand_scan.c` to skip expansion of `$?` during the initial global expansion if it's part of a semicolon sequence, instead copying it literally - Introduced `expand_dollar_question_in_segment` and called it within `process_last_segment` and `process_semicolon_segment` in `src/parser/parser_helpers.c` - This ensures that `$?` is expanded *after* each segment executes, reflecting the correct exit status of the immediately preceding command

Current Status: - The exit status works correctly when commands are on separate lines (`false` followed by `echo $?` prints `1`) - However, with semicolons, the exit status from the first segment is not being preserved for the second segment - When `process_semicolon_segment` calls `execute` for `false`, the exit status should be set to 1, but when `process_last_segment` calls `expand_dollar_question_in_segment`, it reads 0 instead of 1

Files Modified: - `src/parser/parser_expand_scan.c`: Modified `process_envvar` to skip `$?` expansion during initial parsing for semicolon sequences - `src/parser/parser_helpers.c`: Added `expand_dollar_question_in_segment` function and integrated it into segment processing

Test Cases (Currently Failing): - `/bin/ls /doesnotexist; echo $?` (should print `1` or `2`, currently prints `0`) - `true; echo $?` (should print `0`, currently prints `0` - this one works) - `false; echo $?` (should print `1`, currently prints `0`) - `/bin/ls; /bin/ls /doesnotexist; echo $?` (should print `0` then `1`, currently prints `0`)

Next Steps Needed: - Verify that `execute` correctly sets the exit status via `post_wait_set_status` - Ensure the exit status is preserved in `node->rt->exit_status` between segment executions - Debug why `expand_dollar_question_in_segment` reads 0 instead of the previous segment's exit status

6. PATH Unset Behavior (PENDING)

Problem: After `unset PATH`, a bare `ls` should fail with "command not found" and exit code 127.

Status: Not yet tested/fixed.

Test Cases Needed: - `unset PATH` - `ls` (should print "command not found" and exit with 127) - `echo $?` (should print `127`) - `export PATH=/bin:/usr/bin` - `ls` (should work) - `echo $?` (should print `0`)

Memory Management Fixes

Pipe Double Free Fix

Problem: `echo hi | cat` crashes with "pointer being freed was not allocated".

Solution: - Modified `exec_child_envp_handling` to explicitly free `child_args` and `node->path`, and to avoid freeing the original `envp` passed to the child - Modified `run_parent_segment` to close the write end of the pipe (`node->fds[1]`) in the parent immediately after forking - Removed `strarrfree(node->ori_args)` call from `run_parent_segment` to prevent double freeing, as `node->ori_args` is a pointer to the original parsed line, which should be freed once globally - Added `NULL`ing after `free(...)` in child cleanup

Files Modified: - `src/pipe/pipe_core.c`: Refactored `exec_child_envp_handling` to properly manage child resources - `src/pipe/pipe_helpers.c`: Modified `run_parent_segment` to close pipe FDs and prevent double free - `src/cmd/cmd_exit_helpers.c`: Modified `cleanup_child_resources` to only free child-specific resources

Test Cases: - `echo hi | cat` (should work without crash) - `echo hi | cat | cat` (should work without crash) - `ls | cat` (should work without crash) - `/bin/ls /nope | cat` (should work without crash)

Code Structure Improvements

Norminette Compliance

Changes Made: - Refactored `handle_exit_with_args` in `src/cmd/cmd_exit_parse.c` to extract `handle_too_many_args` helper - Refactored `one_commnad` in `src/pipe/pipe_helpers.c` to extract `handle_empty_command` helper - Refactored `process_quotes_and_exec` in `src/parser/parser_helpers.c` to extract `process_last_segment` and `process_semicolon_segment` helpers

Status: Some violations may remain. Need to run norminette again after all fixes.

Key Code Changes

src/parser/parser_helpers.c

```c
static char **expand_dollar_question_in_segment(char **segment, t_node *node)
{
	int i;
	char *expanded;

	i = 0;
	while (segment && segment[i])
	{
		if (ft_strchr(segment[i], '$') && ft_strnstr(segment[i], "$?", ft_strlen(segment[i])
		{
			expanded = expand_envvar(ft_strdup(segment[i]), NULL, node);
			if (expanded)
			{
				free(segment[i]);
				segment[i] = expanded;
			}
		}
		i++;
	}
	return (segment);
}

static char **process_semicolon_segment(char **args, char **envp, t_node *node,
			int start, int semi_idx, char **old_ori_args)
{
	char **segment;
	char **ori_segment;

	segment = extract_segment(args, start, semi_idx);
	if (!segment)
	{
		strarrfree(args);
		if (old_ori_args)
			strarrfree(old_ori_args);
		return (envp);
	}
	ori_segment = extract_segment(old_ori_args, start, semi_idx);
	node->ori_args = ori_segment;
	node->semicolon_sequence = true;
```

```c
    segment = expand_dollar_question_in_segment(segment, node);
    envp = execute(segment, envp, node);
    strarrfree(segment);
    if (ori_segment)
        strarrfree(ori_segment);
    return (envp);
}
```

src/core/line__reader.c

```c
static char g_remainder[4096];
static size_t g_remainder_len = 0;

static char *read_chunks(int fd)
{
    // Implementation with static buffer to handle line-by-line reading
    // even when read() returns multiple lines at once
}
```

src/parser/parser.c

```c
static int process_heredoc_during_parsing(char **args, char **envp, t_node *node)
{
    int i;
    int ret;

    if (!args || !node || !node->ori_args)
        return (0);
    i = 0;
    ret = 0;
    while (node->ori_args[i] && !ret)
    {
        if (isdlr(node->ori_args[i]) || istlr(node->ori_args[i]))
        {
            if (!node->ori_args[i + 1] || !node->ori_args[i + 1][0])
                return (1);
            if (node->redir_fd < 0 && setup_heredoc_file(node))
                return (1);
            ret = left_double_redir(args, envp, &i, node);
        }
        i++;
    }
    return (ret);
}

char **parser(char *str, char **envp, t_node *node)
```

```c
{
    char **args;

    // ... existing code ...
    args = split_joined_quote_after_cmd(args);
    if (process_heredoc_during_parsing(args, envp, node))
    {
        strarrfree(args);
        return (envp);
    }
    return (process_quotes_and_exec(args, envp, node));
}
```

Current Status

Completed

1. [DONE] Fixed double free with semicolon-separated commands
2. [DONE] Fixed double free in export command
3. [DONE] Fixed heredoc no output
4. [DONE] Fixed quote expansion order for nested quotes
5. [DONE] Fixed pipe double free

In Progress

1. IN PROGRESS Fix exit code after semicolons - Exit status not being preserved between segments

Pending

1. PENDING Re-check PATH unset behavior
2. PENDING Run Valgrind tests (Linux) - Need to verify:
   - Main process: 3 open FDs, 0 bytes in 0 blocks
   - Every child process: no extra FDs, no "still reachable" from parser
3. PENDING Run norminette and fix remaining issues

Testing Commands

Functional Tests

```bash
# Semicolon tests
/bin/ls /doesnotexist; echo $?
true; echo $?
false; echo $?
/bin/ls; /bin/ls /doesnotexist; echo $?

# Pipe tests
```

```
echo hi | cat
echo hi | cat | cat
ls | cat
/bin/ls /nope | cat

# Heredoc tests
cat << EOF
hello
EOF
cat << EOF | cat
hello
EOF

# Quote tests
export USER=foo
echo "'$USER'"
echo '"$USER"'
echo '$USER'
echo "$USER"

# Export tests
export TEST=42
export TEST=43
env | grep TEST
unset TEST

# PATH tests
unset PATH
ls
echo $?
export PATH=/bin:/usr/bin
ls
echo $?
```

Valgrind Tests (Linux)

```
# With suppressions (evaluator style)
valgrind --leak-check=full --leak-resolution=high -s \
  --track-origins=yes --num-callers=500 --show-mismatched-frees=yes \
  --show-leak-kinds=all --track-fds=yes --trace-children=yes \
  --suppressions="$PWD/bin.supp" --suppressions="$PWD/readline.supp" \
  ./minishell

# Without suppressions
valgrind --leak-check=full --leak-resolution=high -s \
  --track-origins=yes --num-callers=500 --show-mismatched-frees=yes \
  --show-leak-kinds=all --track-fds=yes --trace-children=yes \
  ./minishell
```

Commands to test under Valgrind: - `echo hi | cat` - `echo hi | cat | cat` - `cat << EOF\nhello\nEOF` - `/bin/ls /nonexistent` - `/bin/ls /noexist; echo $?`

Expected Results: - Main minishell: FILE DESCRIPTORS: 3 open (3 std), in use at exit: 0 bytes in 0 blocks - Every child process: no extra FDs (no 4,5 from backups), no "still reachable" from parser

Next Steps

1. **Fix exit code after semicolons**:
   - Debug why exit status from first segment is not preserved for second segment
   - Verify `execute()` correctly sets exit status via `post_wait_set_status`
   - Ensure `node->rt->exit_status` is preserved between segment executions
   - Check if `expand_dollar_question_in_segment` is reading the correct exit status
2. **Re-check PATH unset behavior**:
   - Test that `unset PATH` followed by `ls` fails with exit code 127
   - Verify error message matches bash
3. **Run Valgrind tests**:
   - Test on Linux with evaluator flags
   - Verify no memory leaks in main process
   - Verify no memory leaks in child processes
   - Verify correct FD counts (3 in main, no extra in children)
4. **Run norminette**:
   - Fix any remaining violations
   - Ensure all helper functions are in appropriate files
5. **Final evaluation sheet pass**:
   - Run the complete 6/11/2021 minishell evaluation sheet
   - Verify all tests pass
   - Create final report

Important Notes

- The exit status is stored in `node->rt->exit_status` and accessed via `get_exit_status_n(node)` and `set_exit_status_n(node, status)`
- The exit status works correctly when commands are on separate lines, but not with semicolons
- This suggests the issue is in how `process_semicolon_segment` and `process_last_segment` handle the exit status
- The order of operations is: `expand_dollar_question_in_segment` is called BEFORE `execute`, so it should use the previous segment's exit

status

- For the first segment, there's no previous segment, so it should use 0 (which is correct)
- For subsequent segments, it should use the exit status from the immediately preceding segment

File Structure

Key files involved: - `src/parser/parser_helpers.c` - Segment processing and `$?` expansion - `src/parser/parser_expand_scan.c` - Initial `$?` expansion handling - `src/parser/parser.c` - Main parser with heredoc processing - `src/core/line_reader.c` - Buffered line reading for heredoc - `src/pipe/pipe_core.c` - Pipe execution and heredoc attachment - `src/cmd/cmd_utils.c` - Command execution and heredoc attachment - `src/pipe/pipe_helpers.c` - Pipe helpers and heredoc attachment - `src/core/runtime.c` - Exit status getter/setter functions

---

Last Updated: 2025-11-10 **Status**: Exit code after semicolons still needs debugging