bc_shell
Design

Benny Chao
January 19th, 2023

Structures

State

Field	Purpose
in_redirect_regex	Find input redirection: < *path*
out_redirect_regex	Find output redirection: *[1]>[>] path*
err_redirect_regex	Find output redirection: 2*>[>] path*
path	An array of directories to search for external commands
prompt	Prompt to display to the user, defaults to $
max_line_length	The longest possible command line
current_line	The current command line
current_line_length	The length of the current command line
command	The command to execute
fatal_error	True if an error happened that should exit the shell

Command

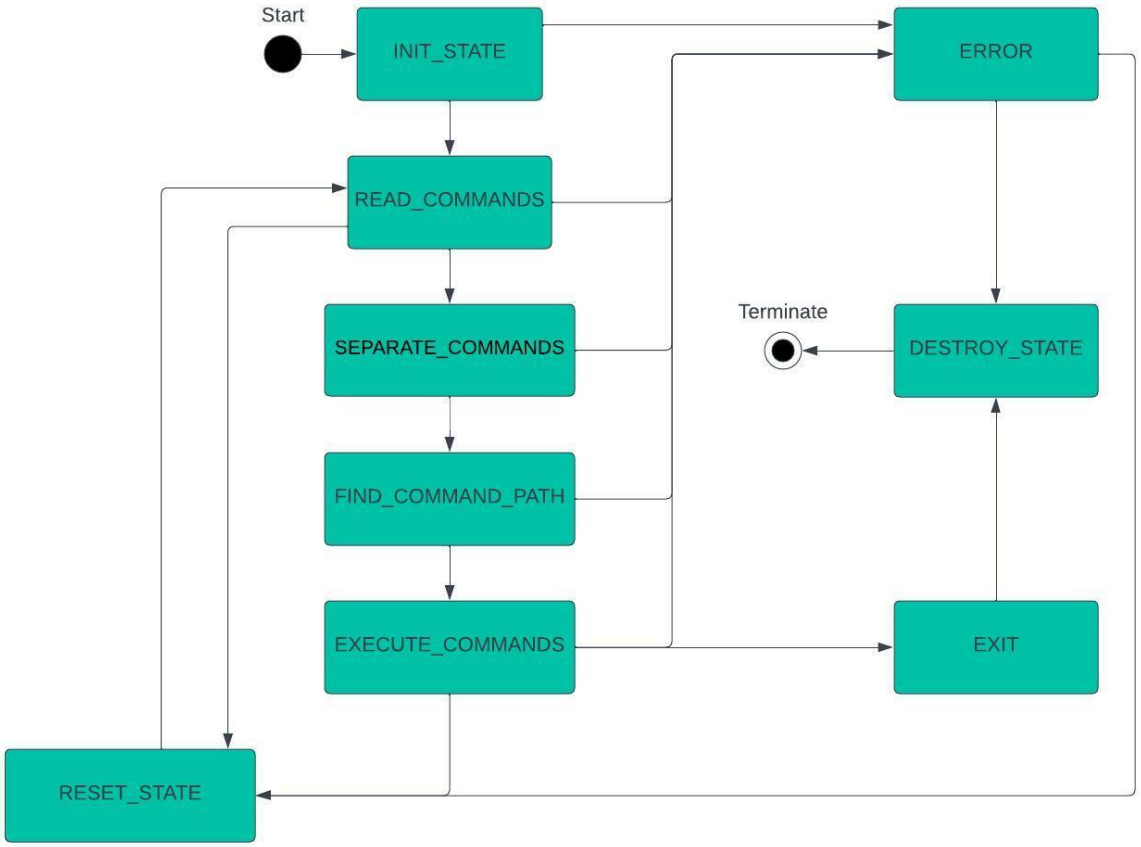
line	The command line for this command
command	The command (e.g. ls, exit, cd, cat)
argc	The number of arguments passed to the command
argv	The arguments passed to the command
stdin_file	The file to redirect stdin from, or NULL
stdout_file	The file to redirect stdout to, or NULL
stdout_overwrite	Append to or truncate the stdout file
stderr_file	The file to redirect stderr to, or NULL
stderr_overwrite	Append to or truncate the stderr file
exit_code	The status returned from the command

Finite State Machine

State Table

From State	To State	Action
BC_FSM_INIT	INIT_STATE	init_state
INIT_STATE	READ_COMMANDS	read_commands
INIT_STATE	ERROR	error
READ_COMMANDS	RESET_STATE	reset_state
READ_COMMANDS	SEPARATE_COMMANDS	separate_commands
READ_COMMANDS	ERROR	error
SEPARATE_COMMANDS	FIND_COMMAND_PATH	find_command_path
SEPARATE_COMMANDS	ERROR	error
FIND_COMMAND_PATH	EXECUTE_COMMANDS	execute_commands
FIND_COMMAND_PATH	ERROR	error
EXECUTE_COMMANDS	RESET_STATE	reset_state
EXECUTE_COMMANDS	EXIT	do_exit
EXECUTE_COMMANDS	ERROR	error
RESET_STATE	READ_COMMANDS	read_commands
EXIT	DESTROY_STATE	destroy_state
ERROR	RESET_STATE	reset_state
ERROR	DESTROY_STATE	destroy_state
DESTROY_STATE	BC_FSM_EXIT	—

State Transition Diagram



Functions

init_state

Purpose

Initialise the state object.

Parameters

The state object to initialise.

Return

Type	Next State
Success	`READ_COMMANDS`
Failure	`ERROR`

Pseudocode

```
If any errors occur
     set state.fatal_error to true and return ERROR

set input_max_length
set status // to determine type of command
set output_file to NULL
set error_file to NULL
set output_append to 0
set error_append to 0

return READ_COMMANDS
```

read_commands

Purpose

Read a command line from stdin.

Parameters

The state to store the command line into.

Return

Read	Next State
<whitespace>\n	`RESET_STATE`
.*\n	`SEPARATE_COMMANDS`
Failure	`ERROR`

Pseudocode

```
if an error getting the current working directory
     return ERROR
write the user input from stdin into input
If error reading
     return ERROR
If empty string
     return RESET_STATE
return SEPARATE_COMMANDS
```

separate_commands

Purpose

Separate the commands and store them into the separate variables. Currently, only simple commands are supported, so there will only be one command.

Parameters

The state object to store the commands in.

Return

Type	Next State
Success	`PARSE_COMMANDS`
Failure	`ERROR`

Pseudocode

```
If any errors occur
     return ERROR

Retrieve command by using strtok(input, " ");

return PARSE_COMMANDS
```

find_command_path

Purpose

Find the path of args[0] (the command).

Parameters

The state object to store the command data into.

Return

Type	Next State
Success	`EXECUTE_COMMANDS`
Failure	`ERROR`

Pseudocode

```
If there is an error
     return ERROR
getenv(“PATH”)
Create copy of PATH variable
Tokenize the PATH variable by ‘:’
builds the full path of the command by concatenating the
current path and the command

check if the full path exists
If path exists
     Set command_path
     Break;
Else
     return ERROR

free(path)

return EXEUTE_COMMANDS
```

execute_commands

Purpose

Parse the commands to separate the command name, arguments, and I/O redirection.

Parameters

The state object to store the command data in.

Return

Type	Next State
“exit” command	`EXIT`
Success	`RESET_STATE`
Failure	`ERROR`

Pseudocode

```
call execute()
     If execute has an error
          return ERROR

print the command to stdout

return RESET_STATE
```

builtin_cd

Purpose

Change the current working directory.

Parameters

The command object with the parameters.

Return

None

Pseudocode

```
Create var to store current working directory
Set var previous_directory to current working directory

if(path == NULL || strcmp(path, "~")  == 0) {
     chdir(getenv("HOME")); // go to the home directory
}
else if (strcmp(path, "..") == 0) {
     chdir(".."); // go to the parent directory
}
else if (strcmp(path, "/") == 0) {
     chdir("/"); // go to the root directory
}
else if(strcmp(path, "-") == 0) {
     chdir(prev_dir); // go to the previous directory
}

else if fail to find path
     return ERROR
```

execute

Purpose

Execute an external command.

Parameters

The command object with the parameters.
The path found from find_command_path

Return

None

Pseudocode

```
if path == NULL
     return ERROR
pid_t pid = fork()
if (pid == 0)
     Call execv from child process passing params
     if command not found
          return ERROR
else if (pid > 0) // parent process
     var status
     waitpid
     if (WIFEXITED(status))
          int exit_status = WEXITSTATUS(status);
          return exit_status;
     else
          return ERROR
else // fork failed
     return ERROR
```

redirect

Purpose

Setup any I/O redirections for the process.

Parameters

The command to perform the I/O redirection on.

Return

None

Pseudocode

```
If any errors, close any open files and return

If the command.stdin_file is not NULL
     Open the file
     Call dup2 for the file and stdin

If the command.stdout_file is not NULL
     Open the file for truncation or appending depending on
          command.stdout_overwrite
     Call dup2 for the file and stdout

If the command.stderr_file is not NULL
     Open the file for truncation or appending depending on
          command.stderr_overwrite
     Call dup2 for the file and stderr
```

run

Purpose

Run a process.

Parameters

The command object to run.
The array of PATH directories to search for the program.

Return

Only returns if all of the calls to execv fail.

Pseudocode

```
If the command.command has a /
     Set command.argv[0] to command.command
     Call exeve for the command
Otherwise
     If path is empty
          Set err to ENOENT
     otherwise
          Loop over the path array
               Set cmd to path[i]/command.command
               Set command.argv[0] to cmd
               Call execv for the cmd

               If the error from execv is not ENOENT
                    Exit the loop
```

error

Purpose

Display the correct error message when a process fails

Parameters

The command that executed

Return

Error code

Pseudocode

```
Print the message and return the appropriate value for the given
error code
```

do_exit

Purpose

Exit the shell. This will lead to the termination of the shell.

Parameters

The current state object.

Return

`DESTROY_STATE`

Pseudocode

```
call do_reset_state()
return DESTROY_STATE
```

reset_state

Purpose

Reset the state object for reading a new line.

Parameters

The state to reset.

Return

`READ_COMMANDS`

Pseudocode

```
call do_reset_state()
return READ_COMMANDS
```

do_reset_state

Purpose

Reset the state object for reading a new line.

Parameters

The state to reset.
The error object to reset.

Return

`void`

Pseudocode

```
Free any dynamically allocated memory
Set current_line to NULL
return READ_COMMANDS
```

error_handler

Purpose

Handle an error.

Parameters

The current state object.

Return

state.fatal_error	Next State
true	`DESTROY_STATE`
false	`RESET_STATE`

Pseudocode

```
If the current_line is NULL
     Print error code

If fatal_error
     Return DESTROY_STATE

return RESET_STATE
```

destroy_state

Purpose

Reclaim memory from the state object and zero it out (NULL, 0, false). This will terminate the shell.

Parameters

The state to destroy.

Return

`DC_FSM_EXIT`

Pseudocode

```
Free up any dynamic memory in the state object
Return DC_FSM_EXIT
```