Shell Program

By Jokers

TABLE OF CONTENT

BACK

NEXT

Libraries & Built-in Functions

FOLDER 01

Libraries:

- **<stdio.h>**

  Provides standard input/output functions like printf, fprintf, fgets, etc.

Examples:

```
printf("Hello, World!\n"); // Print a string to the console.
```

- **<stdlib.h>**

Contains general-purpose utility functions for memory allocation, process control, conversions, and others.

Example:

```
exit(0); // Exit a program successfully.
```

BACK

NEXT

Libraries & Built-in Functions

FOLDER 01

Libraries:

03

- **<unistd.h>**

Provides access to the POSIX operating system API, including functions for process control (fork, execvp), file handling, and more.

Example:

```
pid_t pid = fork(); // Create a new process.
```

04

- **<string.h>**

Contains string manipulation functions like strtok, strcmp, etc.

Example:

```
char str[] = "Hello,World";
char *token = strtok(str, ",");
```

BACK

NEXT

Libraries & Built-in Functions

FOLDER 01

Libraries:

05 **<sys/types.h>**

Defines data types used in system calls like pid_t for process IDs and off_t for file sizes.

06 **<sys/wait.h>**

Provides macros and functions to wait for process state changes, such as wait.

07 **<fcntl.h>**

Provides an interface for file control operations, such as open and fcntl.

BACK

NEXT

Libraries & Built-in Functions

FOLDER 01

Functions:

- **getlogin_r**
  Gets the username of the user logged in on the controlling terminal.
  Returns: 0 on success, or an error code.

- **gethostname**
  Gets the hostname of the current machine.
  Returns: 0 on success, or -1 on failure.

- **strcspn**
  Finds the length of the initial segment of a string that does not contain specified characters.
  Returns: The number of characters before the first match.

- **printf**
  Prints formatted output to standard output.
  Returns: The number of characters printed.

- **fflush**
  Forces a write of buffered data to the output stream.

- **fgets**
  Reads a line of text from a file or standard input.
  Returns: Pointer to the string, or NULL on failure.

- **getcwd**
  Gets the current working directory.
  Returns: Pointer to a string containing the path, or NULL on failure.

BACK

NEXT

Libraries & Built-in Functions

FOLDER 01

Functions:

- **strtok**
  Tokenizes a string into substrings based on delimiters.
  Returns: Pointer to the token, or NULL if no tokens are found.
- **strcmp**
  Compares two strings lexicographically.
  Returns: 0 if equal, <0 if s1 < s2, >0 if s1 > s2.
- **fprintf**
  Prints formatted output to a file stream.
  Returns: The number of characters printed.
- **chdir**
  Changes the current working directory.
  Returns: 0 on success, -1 on failure.
- **perror**
  Prints a description of the last error.
- **fork**
  Creates a new process.
  Returns: 0 to the child, PID of child to parent, -1 on failure.
- **execvp**
  Replaces the current process with a new one.
  Returns: Does not return on success, -1 on failure.
- **exit**
  Terminates the current process.

BACK

NEXT

Libraries & Built-in Functions

FOLDER 01

Functions:

- **wait**
  Waits for a child process to terminate.
  Returns: PID of the child that terminated, or -1 on error.

- **pipe**
  Creates a unidirectional data channel for inter-process communication.
  Returns: 0 on success, -1 on error.

- **dup2**
  Duplicates a file descriptor.
  Returns: New file descriptor, or -1 on error.

- **close**
  Closes a file descriptor.
  Returns: 0 on success, -1 on failure.

- **strstr**
  Finds the first occurrence of a substring.
  Returns: Pointer to the substring, or NULL if not found.

- **open**
  Opens a file.
  Returns: File descriptor, or -1 on error.

BACK

NEXT

MAIN FUNCTION

FOLDER 02

Key Components:

1. Variable Declarations:

- username[32]: Stores the username of the current user.
- hostname[32]: Stores the hostname of the machine.
- cwd[256]: Stores the current working directory.
- command[1024]: Stores the command entered by the user.

BACK

NEXT

MAIN FUNCTION

FOLDER 02

Key Components:

2. Fetching System Information:

- getlogin_r(username, sizeof(username)): Retrieves the username of the currently logged-in user.
- gethostname(hostname, sizeof(hostname)): Retrieves the system's hostname.

3. Shell Loop:

- The program runs an infinite while loop, continually prompting the user for input and executing commands.

BACK

NEXT

MAIN FUNCTION

FOLDER 02

Key Components:

4. Prompt Display:

- getcwd(cwd, sizeof(cwd)): Retrieves the current working directory.
- printf("[%s@%s:%s]$ ", username, hostname, cwd): Displays the shell prompt in the format [username@hostname:cwd]$.

5. Reading and Processing Commands:

- fgets(command, sizeof(command), stdin): Reads a command from the user.
- command[strcspn(command, "\n")] = '\0';: Removes the newline character from the command string.

MAIN FUNCTION

FOLDER 02

Key Components:

6. Command Handling:

- if (strcmp(command, "exit") == 0): Exits the loop (and the program) if the user types exit.
- Piping (|):
  - If the command contains |, the program calls handle_piping(command) to process it.
- Redirection (>):
  - If the command contains >, the program calls handle_redirection(command) to handle output redirection.
- Compound Commands (;):
  - If the command contains ;, it calls handle_compound_commands(command) to execute multiple commands sequentially.
- Normal Commands:
  - For other cases, it calls execute_command(command) to execute a single command.

BACK

NEXT

MAIN FUNCTION

FOLDER 02

Key Components:

6. Command Handling (cont.):

- Logical AND (&&):
  - If any command fails, subsequent commands are skipped.
- Logic:
  + It checks if the user's input contains && using strstr(command, "&&").
  + If found, it calls handle_and_commands(command) to process the commands.
- Functionality:
  + handle_and_commands splits the input at &&, executes each command in order, and stops if a command fails (returns a non-zero exit code).

BACK NEXT

EXECUTE COMMAND FUNCTION

FOLDER 03

1. Parse Command:

Splits the input into arguments (args[]) using strtok based on spaces.

2. Handle Built-in Commands:

- If the command is cd:

Change the directory using chdir.

Handle errors like missing arguments or invalid directories.

3. Special Handling for grep:

Removes starting and ending double quotes (") from arguments for better compatibility.

4. Execute Other Commands:

A child process is created using fork.

In the child process, execvp runs the command.

The parent process waits for the child to finish.

5. Error Handling:

If fork or execvp fails, it prints an error message using perror.

BACK

NEXT

EXECUTE COMMAND FUNCTION

FOLDER 03

Example:

- Input: cd /home
  - Changes directory to /home.
- Input: grep "hello world" file.txt
  - Handles the quotes and runs the grep command.
- Input: ls -l
  - Runs ls -l in a child process.

BACK

NEXT

HANDLE PIPING Function

FOLDER 04

A

Split Commands:

The input command is split at the | character into an array commands[].

E.G: ls | grep "^T"

ls

grep "^T"

BACK

NEXT

HANDLE PIPING FUNCTION

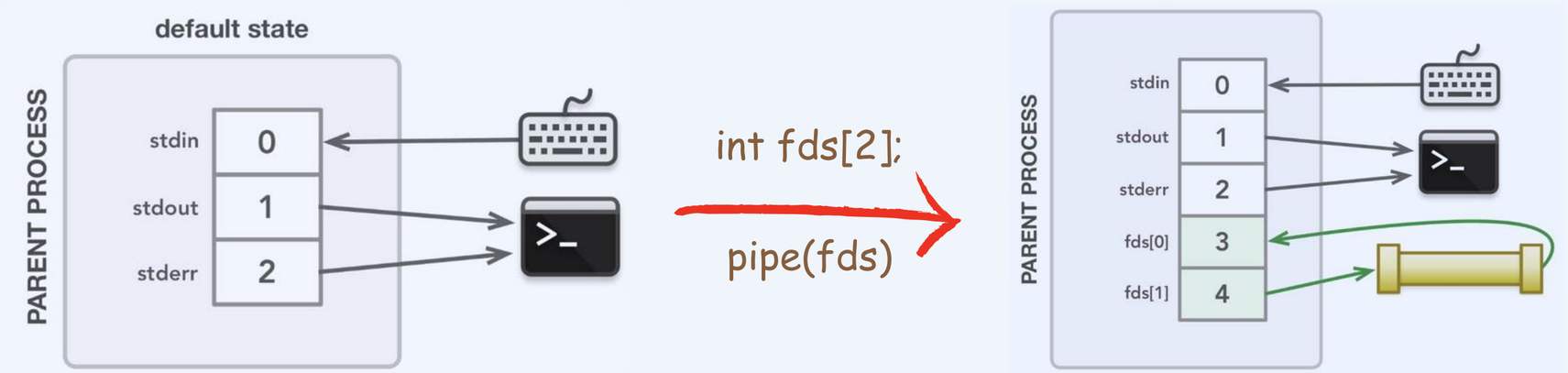
FOLDER 04

Set Up Piping:

B

A pipe is created for inter-process communication between commands.

in_fd keeps track of the input file descriptor for the next command.



BACK

NEXT

HANDLE PIPING FUNCTION

FOLDER 04

Fork for Each Command & Parent Process:

C

Fork for Each Command:

- For each command in commands[]:

A child process is created using fork.

Input is redirected to in_fd using dup2.

Output is redirected to the pipe if it's not the last command.

execute_command() is called to run the command in the child.

Parent Process:

- Waits for the child process to finish.
- Closes the write end of the pipe and updates in_fd to the read end for the next command.

BACK

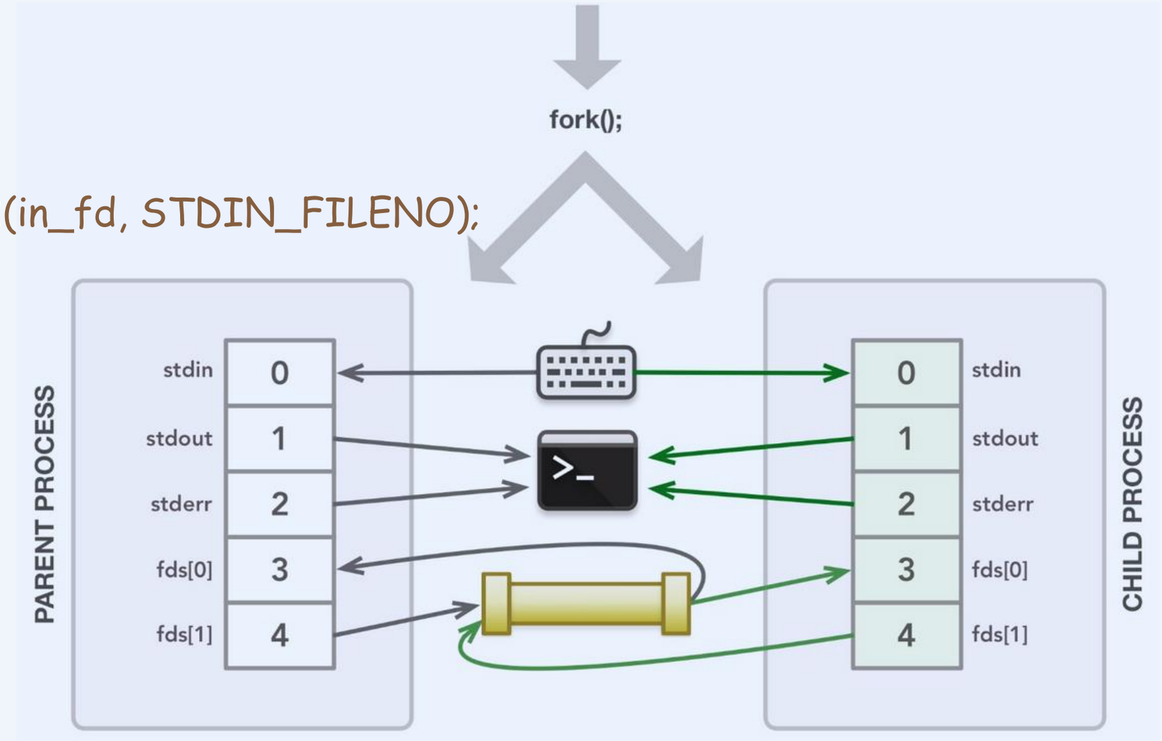
NEXT

HANDLE PIPING FUNCTION

FOLDER 04

Fork for Each Command:

1. Fork
2. dup2(in_fd, STDIN_FILENO);



BACK

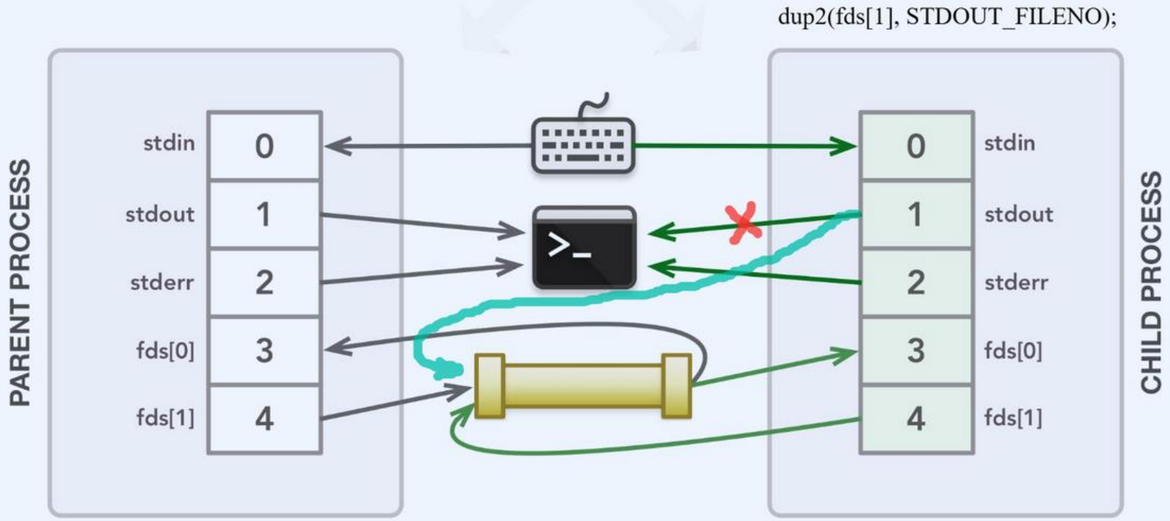
NEXT

HANDLE PIPING FUNCTION

FOLDER 04

Fork for Each Command:

3 dup2(fds[1], STDOUT_FILENO);



BACK

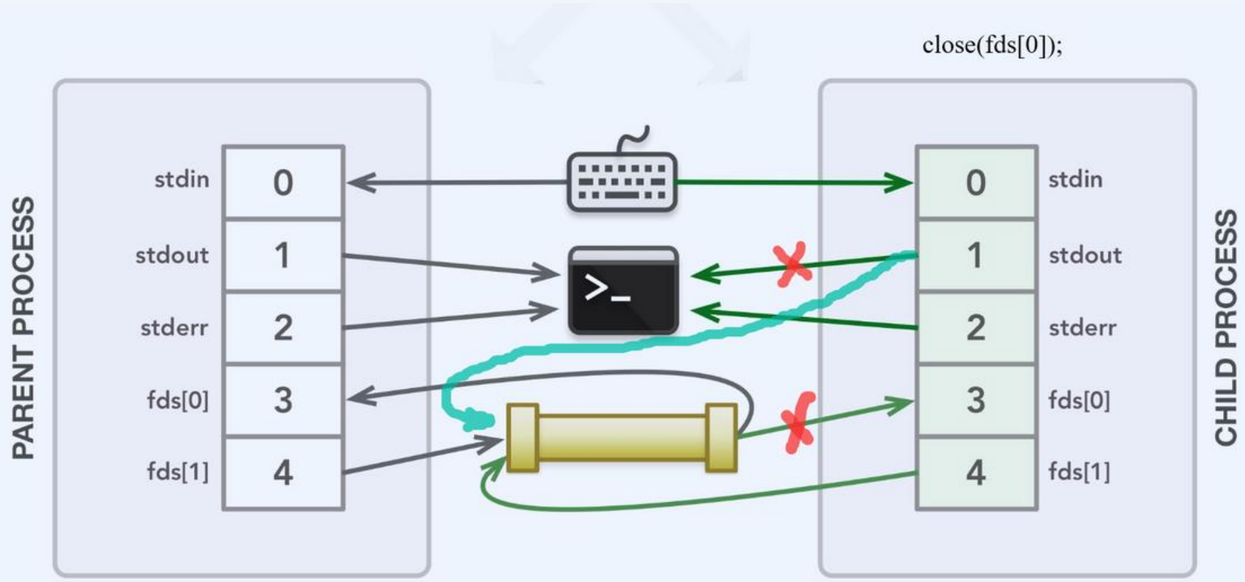
NEXT

HANDLE PIPING FUNCTION

FOLDER 04

Fork for Each Command:

4 close(fds[0]);



BACK

NEXT

HANDLE PIPING FUNCTION

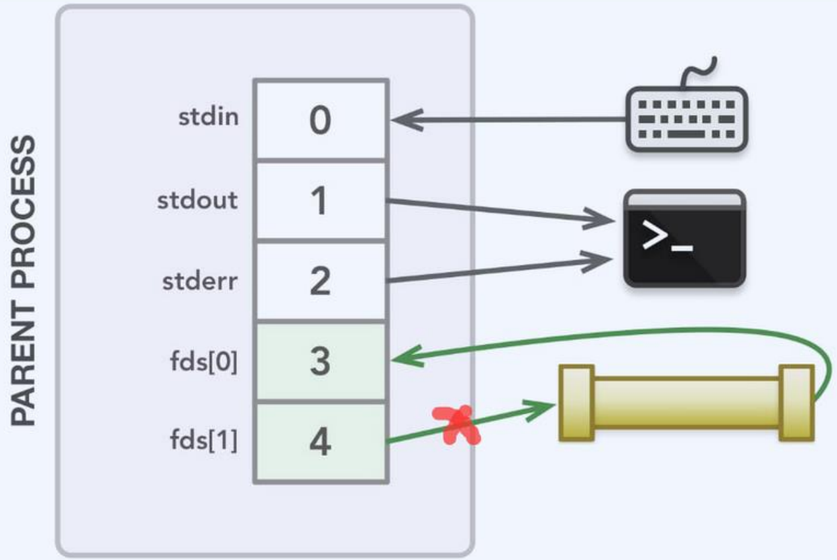
FOLDER 04

Parent Process:

5 execute_command(commands[i]);

6 exit(EXIT_FAILURE);

7 close(fds[1]);



BACK

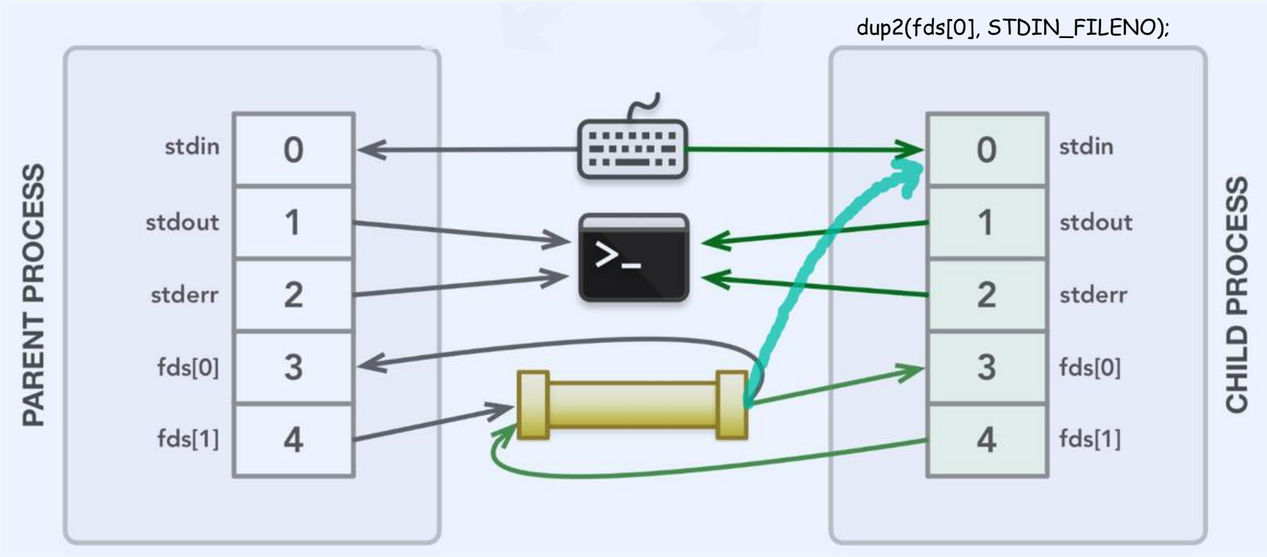
NEXT

HANDLE PIPING FUNCTION

FOLDER 04

Fork for Each Command:

8 dup2(fds[0], STDIN_FILENO);



BACK

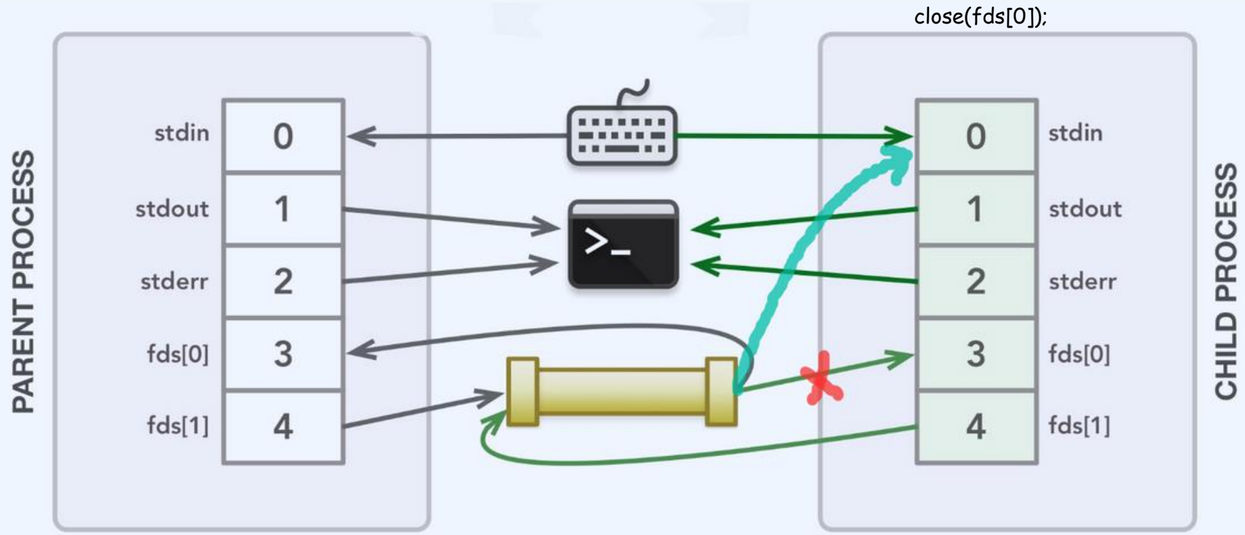
NEXT

HANDLE PIPING FUNCTION

FOLDER 04

Fork for Each Command:

9 close(fds[0]);



BACK

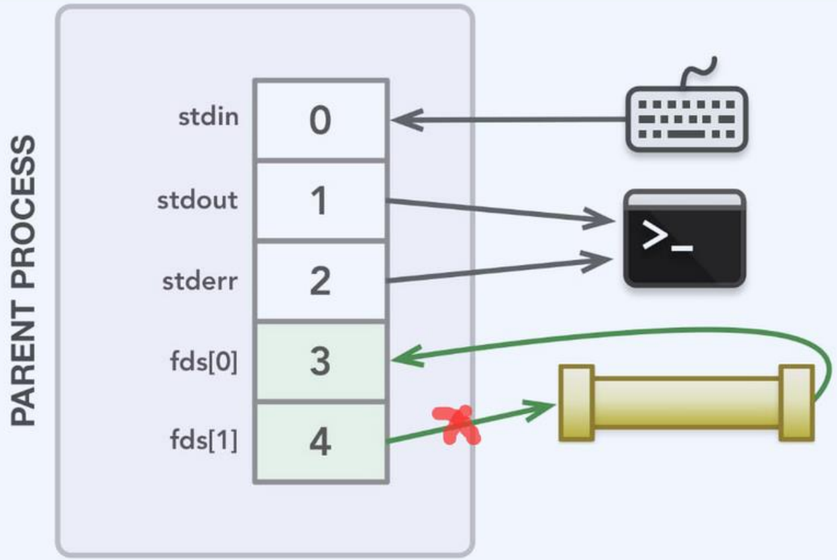
NEXT

HANDLE PIPING FUNCTION

FOLDER 04

Parent Process:

```
10  execute_command(commands[i]);
11   exit(EXIT_FAILURE);
12  close(fds[1]);
```



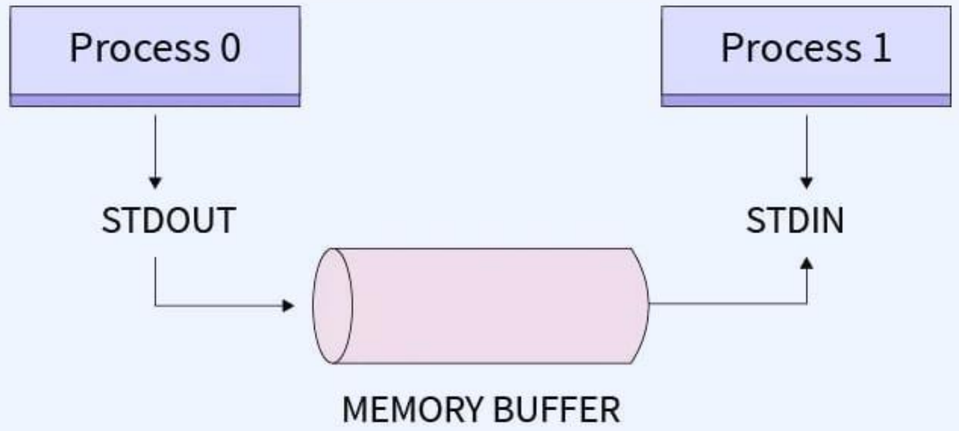
BACK

NEXT

HANDLE PIPING FUNCTION

FOLDER 04

Illustration



BACK

NEXT

HANDLE PIPING FUNCTION

FOLDER 04

Example:

Input: ls | grep file | wc -l

ls writes output to a pipe.

grep file reads from the pipe and writes its output to another pipe.

wc -l reads from the second pipe.

BACK

NEXT

HANDLE REDIRECTION Function

FOLDER 05

1. Detect Redirection:
   - Checks if the command contains:
     >> for appending to a file.
     > for overwriting a file.
   - Splits the command at the redirection operator to separate the command and the output file name.

2. Open File:
   - Opens the specified file with appropriate flags:
     O_APPEND for >> (append mode).
     O_TRUNC for > (overwrite mode).
   - Creates the file if it doesn't exist and sets permissions to 0644.

BACK

NEXT

HANDLE REDIRECTION Function

FOLDER 05

3. Fork Process:
   - In the child process:
     - Redirects STDOUT_FILENO to the file descriptor using dup2, so the command's output goes to the file.
     - Executes the command using execute_command.
   - The parent process waits for the child to finish and closes the file.

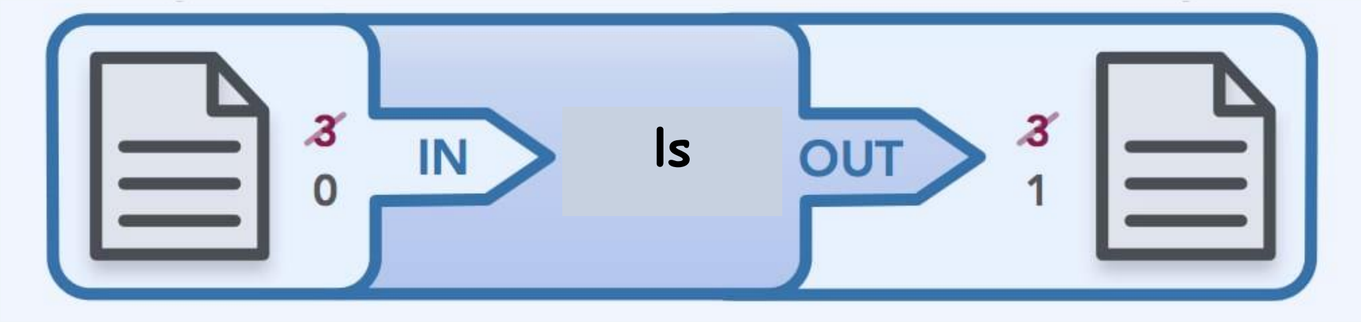
4. Fallback:
   - If no output file is specified, the command is executed as normal.

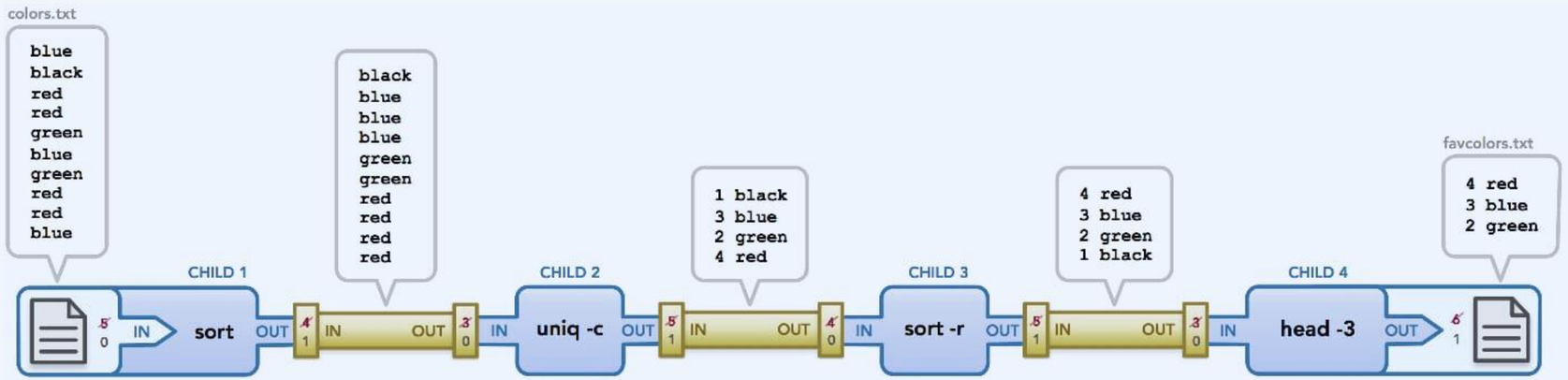
BACK

NEXT

HANDLE REDIRECTION Function

FOLDER 05



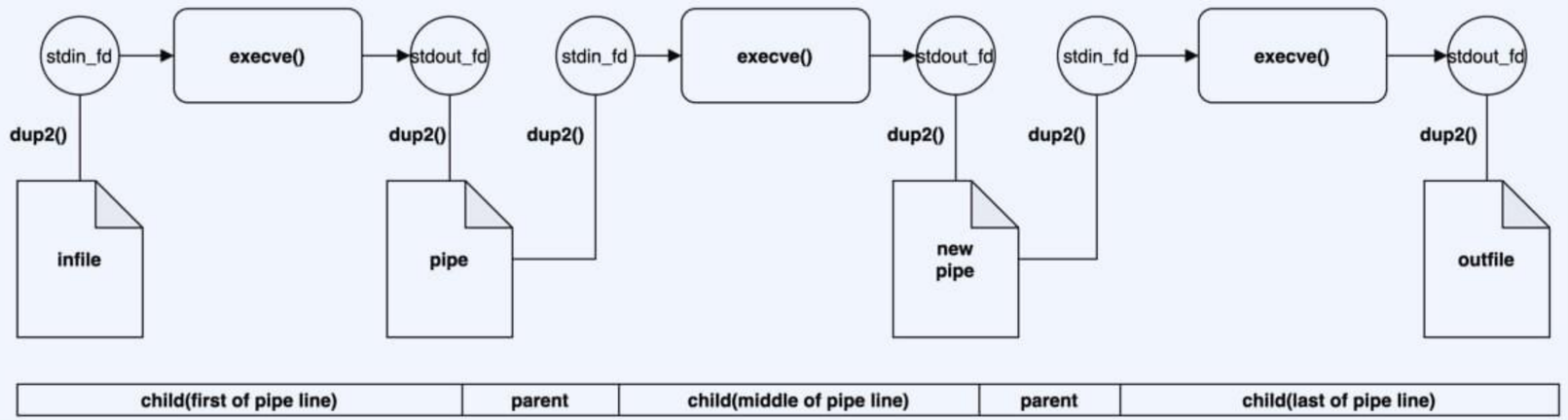


BACK

NEXT

HANDLE REDIRECTION Function

FOLDER 05



BACK

NEXT

HANDLE REDIRECTION Function

FOLDER 05

Example:

Input: ls > file.txt

Redirects the output of ls to overwrite file.txt.

Input: echo "Hello" >> file.txt

Appends "Hello" to file.txt.

BACK

NEXT

HANDLE COMPOUND COMMANDS FUNCTION

FOLDER 06

1. Split Commands:
   - Splits the input string into individual subcommands at each ; using strtok.
   - Stores the subcommands in the subcommands[] array.
2. Execute Commands:
   - Iterates through each subcommand and executes it using the execute_command function.

Purpose:

The function ensures that all subcommands run sequentially, making it ideal for processing multiple independent commands in one line.

BACK

NEXT

HANDLE COMPOUND COMMANDS FUNCTION

FOLDER 06

Example:

Input: ls; echo Hello; pwd

Splits into:

ls

echo Hello

pwd

- Executes each in order, regardless of success or failure.

BACK

NEXT

HANDLE AND COMMANDS FUNCTION

FOLDER 07

1. Split Commands:
   - Splits the input string into subcommands at each && using strtok.
   - Stores the subcommands in the subcommands[] array.
2. Execute Commands:
   - Iterates through each subcommand and executes it using execute_command.
3. Check Status:
   - After each command, checks its exit status using wait(&status).
   - Stops execution if a command fails (i.e., status != 0).

Purpose:

Ensures conditional execution of commands, where the next command only runs if the previous one succeeds.

BACK

NEXT

HANDLE AND COMMANDS Function

FOLDER 07

Example:

- Input: mkdir test && cd test && touch file.txt
  - Executes:
    i. mkdir test (if successful, proceeds to next).
    ii. cd test (if successful, proceeds to next).
    iii. touch file.txt.
- If any command fails, subsequent commands are skipped.

BACK

NEXT

RESULT

This code implements a basic custom shell in C, capable of handling:

1. Simple Command Execution: Runs single commands using execvp.
2. Built-in Commands: Supports cd for directory changes.
3. Piping (|): Executes commands connected via pipes (e.g., ls | grep file).
4. Redirection (>, >>): Redirects output to files, with overwrite (>) or append (>>) modes.
5. Compound Commands (;): Executes multiple commands sequentially (e.g., mkdir test; cd test).
6. Conditional Execution (&&): Executes commands sequentially, stopping if a command fails (e.g., mkdir test && cd test).
7. Handling Quoted Arguments: Processes commands like grep "pattern".

RESULT

Example Outputs:

- **Simple Commands:**

  Input: ls

  Output: Lists files in the current directory.

- **Built-in Commands:**

  Input: cd folder

  Action: Changes the working directory.

- **Piping:**

  Input: ls | grep file

  Output: Filters ls output for entries containing "file".

- Redirection:

  Input: ls > output.txt

  Action: Writes the directory listing to output.txt.

  Input: echo "Hello" >> output.txt

  Action: Appends "Hello" to output.txt.

- Compound Commands:

  Input: mkdir test; cd test; touch file.txt

  Action: Creates a directory, enters it, and creates a file.

- Conditional Commands:

  Input: mkdir test && cd test && touch file.txt

  Action: Executes commands sequentially, stopping if any fail.

Exit Behavior:

Input: exit

Action: Exits the shell loop.

BACK

NEXT

RESULT

FOLDER 05

Overall Result:

This shell allows execution of common commands with advanced features like piping, redirection, and conditional execution. However, it lacks features like error handling for invalid commands or advanced parsing capabilities (e.g., nested commands).

BACK

NEXT

Our Team

Shell Program

أحمد السيد أحمد البهجي

محمد أحمد محمد أحمد عطية

محمد أحمد محمد حسن الجزار

محمد خالد فتحي محمود بدر

محمد خالد محمد محمد إبراهيم العبادي

مهند محمد السيد عبد الكريم

مؤمن أحمد طه الشعراوي

THANK YOU

By Jokers