**Phase1.c**

**Keywords and Constructs**

1. **#include**: This preprocessor directive is used to include the standard input-output library (**<stdio.h>**), standard library (**<stdlib.h>**), Unix standard functions (**<unistd.h>**), types used in system calls (**<sys/types.h>**), wait functions (**<sys/wait.h>**), and string manipulation functions (**<string.h>**).
2. **#define**: This preprocessor directive defines constants. In this case, **MAX\_INPUT\_SIZE** is set to 1024 (the maximum size for user input), and **MAX\_ARG\_SIZE** is set to 100 (the maximum number of arguments).
3. **void**: This keyword specifies that a function does not return a value.
4. **int**: This keyword specifies that a function returns an integer value.
5. **char**: This keyword is used to declare character variables or arrays, which in this code are used to handle strings.
6. **pid\_t**: This is a data type used for process IDs. It is defined in **<sys/types.h>**.
7. **while (1)**: This creates an infinite loop, which will continue until a **break** statement is encountered.
8. **if, else**: These are conditional statements that execute code blocks based on whether the condition evaluates to true or false.
9. **return**: This keyword exits a function and optionally returns a value to the calling function.

**Functions**

1. **print\_prompt()**: This function prints the shell prompt **phase1-shell>** to the console.
2. **print\_help()**: This function displays a list of available commands in the shell.
3. **main()**: This is the entry point of the program. It initializes variables, runs the main loop, and handles user input and command execution.
4. **fgets()**: This function reads a line from the specified input stream (in this case, **stdin**) and stores it in the **input** array. It stops reading after the newline character or when the maximum size is reached.
5. **perror()**: This function prints a description for the last error that occurred. It is used here to print error messages if **fgets** or **fork** fails.
6. **strcspn()**: This function computes the length of the initial segment of **input** that consists entirely of characters not in the second string (in this case, the newline character **"\n"**). It is used to remove the newline character from the input string.
7. **strtok()**: This function tokenizes the input string into individual arguments based on the specified delimiter (in this case, a space). It modifies the original string by replacing delimiters with null characters.
8. **fork()**: This function creates a new process by duplicating the calling process. It returns the child's process ID (PID) to the parent process and 0 to the child process. If it fails, it returns a negative value.
9. **execvp()**: This function replaces the current process image with a new process image specified by the command and its arguments. If it fails, it returns -1.
10. **wait()**: This function makes the parent process wait for the child process to finish execution. It can also retrieve the exit status of the child process.

**Overall Structure**

* The program runs an interactive shell that prompts the user for commands.
* It supports a set of predefined commands (**help**, **exit**, and system commands like **ls**, **ps**, etc.).
* User input is read, tokenized into arguments, and then executed in a child process using **fork** and **execvp**.
* The parent process waits for the child to complete before continuing to prompt for the next command.

**Example Flow**

1. The shell prompts the user with **phase1-shell>**.
2. The user enters a command.
3. The command is parsed into arguments.
4. If the command is **exit**, the loop breaks, and the program terminates.
5. If the command is **help**, the available commands are printed.
6. For other commands, a child process is created, and the command is executed.
7. The parent process waits for the child to finish before displaying the prompt again.

**Phase2.c**

**Keywords and Constructs**

1. **#include**: This preprocessor directive is used to include standard libraries. The included libraries are:
   * **<stdio.h>**: Standard input-output functions (e.g., **printf**, **fgets**).
   * **<stdlib.h>**: General utilities (e.g., **exit**, **malloc**).
   * **<unistd.h>**: Unix standard functions (e.g., **fork**, **exec**).
   * **<sys/types.h>**: Data types used in system calls (e.g., **pid\_t**).
   * **<sys/wait.h>**: Macros related to process termination (e.g., **wait**).
   * **<string.h>**: String manipulation functions (e.g., **strcpy**, **strtok**).
2. **#define**: This preprocessor directive defines constants:
   * **MAX\_INPUT\_SIZE**: Maximum size for user input (1024 characters).
   * **MAX\_ARG\_SIZE**: Maximum number of arguments a command can have (100).
   * **MAX\_HISTORY\_SIZE**: Maximum number of commands to store in history (100).
3. **void**: This keyword specifies that a function does not return a value.
4. **int**: This keyword specifies that a function returns an integer value.
5. **char**: This keyword is used to declare character variables or arrays, which are used to handle strings.
6. **pid\_t**: A data type used for process IDs, defined in **<sys/types.h>**.
7. **while (1)**: Creates an infinite loop that continues until a **break** statement is encountered.
8. **if, else**: Conditional statements that execute code blocks based on whether the condition evaluates to true or false.
9. **return**: Exits a function and optionally returns a value to the calling function.

**Functions**

1. **print\_prompt()**: This function prints the shell prompt **phase2-shell>** to the console.
2. **print\_help()**: This function displays a list of available commands in the shell.
3. **print\_history(char history[][MAX\_INPUT\_SIZE], int history\_count)**:
   * This function prints the command history. It takes a 2D array of strings (**history**) and the count of commands stored (**history\_count**) as parameters.
   * It loops through the history and prints each command with its corresponding index.
4. **clear\_history(char history[][MAX\_INPUT\_SIZE], int \*history\_count)**:
   * This function clears the command history by resetting the **history\_count** to zero.
   * It also prints a confirmation message that the history has been cleared.
5. **main()**: The entry point of the program, where the main logic is implemented.
   * It initializes variables, runs the main loop, and handles user input and command execution.
6. **fgets()**: Reads a line from the specified input stream (**stdin**) and stores it in the **input** array. Stops reading after a newline character or when the maximum size is reached.
7. **perror()**: Prints a description for the last error that occurred. Used here to print error messages if **fgets** or **fork** fails.
8. **strcspn()**: Computes the length of the initial segment of **input** that consists entirely of characters not in the second string (in this case, the newline character **"\n"**). Used to remove the newline character from the input string.
9. **strcpy()**: Copies a string from the source (in this case, **input**) to the destination (an entry in the **history** array).
10. **strtok()**: Tokenizes the input string into individual arguments based on the specified delimiter (space). It modifies the original string by replacing delimiters with null characters.
11. **fork()**: Creates a new process by duplicating the calling process. It returns the child's process ID (PID) to the parent process and 0 to the child process. If it fails, it returns a negative value.
12. **execvp()**: Replaces the current process image with a new process image specified by the command and its arguments. If it fails, it returns -1.
13. **wait()**: Makes the parent process wait for the child process to finish execution. It can also retrieve the exit status of the child process.

**Overall Structure**

* The program runs an interactive shell that prompts the user for commands and maintains a history of executed commands.
* It supports several commands:
  + **help**: Displays the list of available commands.
  + **exit**: Exits the shell.
  + **history**: Displays the command history.
  + **clearhistory**: Clears the command history.