Simple Shell in C

Group ID: 36
Group members:
Aditi Aryan (2023037)
Jaitrika Reddy (2023333)

Explanation of the code:

1. Command History Management:
   - Variables:
     1. History: stores the command history
     2. Pid_history: stores process IDs
     3. Time_history: stores execution times
     4. c_hist: tracks the number of stored commands.
   - Functions:
     1. add_to_history: Adds a command to the history, storing the command string, its process ID, start time, and end time.
     2. print_history: Prints all stored commands, their process IDs, and their execution durations. It also gets triggered when user presses ctrl+c via a signal handler.

2. Signal Handling:
   - Functions
     1. my_handler: Captures the SIGINT signal(ctrl+c), prints the command history, and terminates the shell.
     2. sig_handler: Sets up signal handling for ctrl+c using the sigaction function.

3. Command Parsing:
   - Functions
     1. break_delim: Takes a command string and breaks it into an array of arguments using a delimiter (e.g., spaces ‘ ’ or pipes ‘|’).

4. Command Execution:
   - Functions:
     1. launch: Forks a child process to execute commands. If the command contains &, it runs in the background. If not, the parent process waits for the child to finish. If the command is history, it prints the stored history.

5. Pipe Handling:
   - Functions:
     1. pipe_manager: Breaks a command into segments separated by the pipe (‘|’).
     2. pipe_execute: Manages the creation and execution of multiple commands connected by pipes. It uses dup2 to redirect input/output between processes.

6. Shell Main Loop:
   - The shell runs in an infinite loop, prompting the user for commands.
   - It checks for ‘&’ (background processes) and pipes, and either executes the command directly or handles pipes accordingly.
   - Commands without ‘&’ are added to the history, including their execution times and process IDs.

7. Time Management:
   - Functions:
     1. current_time: Returns the current time in microseconds, used to calculate the start and end time of each command execution.

Limitations:

- **No built-in command support:**
  The shell doesn't handle internal shell commands like cd or exit.
  Reason: These commands require interacting with the shell process itself, which execvp cannot do.

- **No I/O redirection:**
  The shell doesn't support redirecting input or output using > or <, which are common in Unix shells.
  Reason: I/O redirection in a shell would require the ability to open files and redirect the standard input/output streams (stdin, stdout) to these files.

- **No advanced signal handling:**
  The shell only handles SIGINT (ctrl+c).
  Reason: Handling signals like SIGTSTP (ctrl+z) would require more advanced process control mechanisms, like suspending processes, resuming them in the background, and bringing them back to the foreground which exceeds current scope of our knowledge.

- **Limited background process handling:**
  The current implementation does not support background execution for piped commands
  Reason: Supporting background execution of piped commands would require managing multiple background processes simultaneously, tracking of multiple child processes, and ensuring that pipes between them remain open while they execute in the background. This level of complexity has not been implemented in the current shell.

- **Fixed command history size (100 entries):**
  The shell limits command history storage to 100 entries.
  Reason: The command history is stored in a fixed-size array. To allow unlimited, the shell would need to use dynamic memory allocation (e.g., linked lists or dynamic arrays)

to expand as needed. With improper logic this could cause memory leaks or performance degradation.