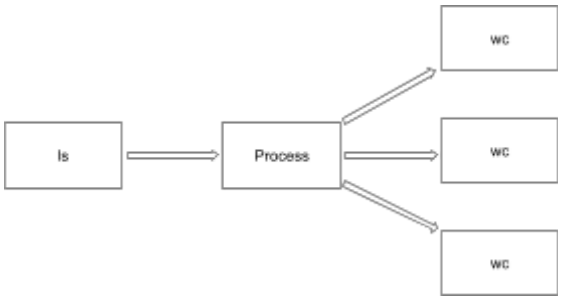
The shell program starts with first creating a new process group with itself as the group leader which is immediately made to be the foreground group of the controlling terminal. Then it blocks all signals which can lead to termination of the shell - SIGINT, SIGQUIT, SIGKILL, SIGSTOP. This prevents the user from accidentally terminating the shell program. After the following initialization, the shell enters an infinite loop where it executes one command from the user in every iteration.

In every iteration, the shell reads the file containing the shortcut commands. Then it unblocks the SIGINT command to allow the user to enter the shortcut command mode by issuing the SIGINT signal. Then the shell waits for input from the user. The user can either enter a simple command or a shortcut command. Shortcut commands are handled using a signal handler which takes the index of the command from the user and converts it into the actual stored command. The signal handler uses a jump statement to return back to the main executing function.After getting the command for the current iteration, the shell forks a new process. The child process handles the execution of the command whereas the parent either waits for the child if it's a foreground command or continues to the next iteration if it is a background command. The parent also makes the child the foreground process on the controlling terminal if it is a foreground command.

In the child process, all signals are unblocked in case of a foreground process. The command is split into tokens which are iterated over sequentially. Actions are specified when special tokens are encountered. The special tokens are sc, <, >, |, || and |||. The actions are as follows:-

a) sc - When the ‘sc’ token is encountered, the command is assumed to be intended to insert or delete a shortcut command. This is handled by maintaining an in-memory data structure containing all the containing shortcut commands. This data structure is appropriately updated when an ‘sc’ command is issued.
b) <, > - When either of ‘<’ or ‘>’ token is encountered, the next token is assumed to contain the name of the file where redirection takes place. The file is opened in read mode for the ‘<’ and write mode for ‘>’. The file descriptor of the file is then duplicated to the appropriate file descriptor (0 or 1) for redirection.
c) |, ||, ||| - When any of the three tokens are encountered, it indicates some form of pipelining will take place. All the tokens encountered till now will form the left side of the pipeline. So a new process is forked from the current one. This new process then uses the execv function to execute the left side of the pipeline. A pipe is used to transfer data across the pipeline. If the pipeline token is ‘||’ or ‘|||’, another process is forked which takes data from the left side of the pipeline and replicates it to multiple pipelines each of which is used by one of the processes on the right side of the pipeline. For example, if the command is ls ||| wc, wc, wc,

the data flow would look like the diagram below. Each arrow represents a pipe and each square represents a process.



After this every time ‘,’ is encountered, the next pipe is duplicated to file descriptor 0 and all the tokens are executed as one command. To handle nested pipelines, a FIFO list is maintained and the next pipe to be used is chosen from the most recent node.

At the end of iterating through all the tokens, a final exec call is made to execute the final step of the command.