Linux Shell

Name: Samiksha Modi
Roll No: 2019331

The system calls fork,execvp,wait and exit were used to handle external commands.

- fork

  Following are the header files required and the parameters of the system call
  #include <stdio.h>
  #include <sys/types.h>
  #include <unistd.h>
  fork();

  To create a child process we use fork(). It returns <0 if there was a failure in creating a new child process. 0 for the child process. >0 i.e. process id of the child process to the parent process. When >0 the parent process will execute.

  Child status information reported by wait is more than just exit status of the child. It also includes normal/abnormal termination, termination cause, exit status
  Following gives information about the status
  - WIFEXITED(status): child exited normally
  - WIFSIGNALED(status): child exited because a signal was not caught
  - WIFSTOPPED(status): child is stopped

- wait

  A call to wait() blocks the calling process until one of its child processes exits or a signal is received. After the child process terminates, the parent continues its execution after wait system call instruction.
  The child process may terminate due to the following reasons
  1. It calls exit()
  2. It returns (int) from main
  3. It receives a signal (from OS or another process) whose default action is to terminate.

  The following is the argument of the system call
  pid_t wait(int *stat_ptr);

  Parameters
  status_ptr: It points to the location where wait() can store the status value. This status value is zero if child process explicitly returns zero status. If it's not zero, then it can be analyzed with the status analysis macros. The status_ptr pointer can be NULL, in which case wait() ignores the child's return status.

- execvp

  Following are the header files required and the parameters of the system call
  #include <unistd.h>
  int execvp(const char * file, char * const argv[]);

  Parameters
  file: It is used to construct a pathname that identifies the new process image file. If the file argument contains a slash character, the file argument is used as the pathname for the file. Otherwise, the path

prefix for this file is obtained by a search of the directories passed as the environment variable PATH.
Argv: An array of character pointers to NULL-terminated strings. The application must ensure that the last member of this array is a NULL pointer. These strings constitute the argument list available to the new process image. The value in argv[0] must point to a filename that's associated with the process being started. Neither argv nor the value in argv[0] can be NULL.
When execvp() is successful, it doesn't return, otherwise, it returns -1 and sets errno.

- exit

  Following are the header files required and the parameters of the system call
  #include <stdlib.h>
  exit(int status)
  exit() terminates the process normally.

  Parameters
  status: The status value is returned to the parent process. The status value of EXIT_SUCCESS or 0 indicates success, and any other value or EXIT_FAILURE indicated failure.
  When exit() is called, any open file descriptors belonging to the process are closed and any children of the process are inherited by process 1, init, and the process parent is sent a SIGCHLD signal.

How My Shell Works

The shell works in an infinite loop (while(1)). It prints the shell prompt, takes the user input as string using fgets, then parses it using strtok and stores them in a 2d array called command.

Then it checks the command, if it is null then it justs print another prompt, else if it is a valid internal command then it executes that command internally, else if it is a valid external command then executes it using fork, execvp, and wait, else tells the user that it is an invalid command. It also saves the history of the current shell using a 2d char array.

Assumptions

- Only the history of the current shell session is saved.
- It does not support two options at once for any command. For eg ls -a is valid but ls -a -r is not.
- In ls only one directory can be read at a time. ls Desktop is valid but ls Desktop Pictures is not.

Options for Shell Commands

My shell handles following
Internal commands - cd, echo, history, pwd and exit
external commands - ls,cat,date,rm and mkdir

- cd
    - .. goes back one folder up
    - ~ goes to home
- echo
    - -n do not append a newline
    - -E explicitly suppress interpretation of backslash escapes
- history

    - -c clear the history list by deleting all of the entries
    - -w write the current history to the history file. If FILENAME is given, it is used as the history file
- pwd
- exit
- ls
    - -a do not ignore entries starting with .
    - -r reverse order while sorting
- date
    - -u print or set Coordinated Universal Time (UTC)
    - -I (capital i) output date in ISO 8601 format
- cat
    - -n number all output lines
    - -E display $ at end of each line
- mkdir
    - -p no error if existing, make parent directories as needed
    - -v print a message for each created directory
- rm
    - -i prompt before every removal
    - -d remove empty directories

Handling of Shell Commands

- cd
  It uses chdir to navigate to the given directory.
    - If more than one argument is given, it prints the error message “Too many arguments.
    - If an invalid option is given, it prints the error message “Invalid argument”.
    - If the given directory does not exist, it prints the error message “Directory not found”.

- echo
  It processes the input according to the flag and gives the required output.
- history
  History is kept track of in the shell using a 2d char array. To print the history it traverses the array and prints it.
    - If just -w option is used without any file name then it writes the current history to the historyfile.txt, else it creates the file specified and writes history to that file.
    - If the program is unable to create the file, it prints the error message “Unable to create file”.
    - If an invalid option is given, it prints the error message “Invalid argument”.
    - If arguments are given with history for eg history hello world, then it prints the message “Invalid argument”.
- pwd
  It prints the current working directory by using getcwd
    - If getcwd returns null, it prints the error message “Unable to get working directory”
    - If an invalid option is given then it print the error message “Invalid argument”
- exit
  It uses exit() to exit the current terminal session.
- ls
  I used readdir to read the files present in the given directory and store them in a 2d array a. Then depending on the flag given I sort them alphabetically. If the flag is -r then I sort them in reverse order. If the flag is -a I print the directories starting with ‘.’ also.
    - If an invalid option is given, it prints the error message “Invalid argument.”
    - If the directory given can’t be found then it prints the error message “Directory not found”

    - If multiple directories are given like ls Desktop Picture, then it shows the error message “Too many arguments”
- date
  I use the time function for this. Passing the argument local to asctime() gives me the IST time and passing the argument utc to asctime() gives me the utc time. I also use strftime() to print the date in iso 8601 format.
    - If the function time() is unable to execute properly, it prints the error message “Time failed” and returns.
    - If an invalid option is given, it prints the error message “Invalid argument”
- cat
  I use fopen to open the file given as argument then read it using fgets and then print it based on the options.
    - If the given file doesn’t exists, it prints the error message “File [name] doesn’t exist”
    - If an invalid option is given, it prints the error message “Invalid argument”
- mkdir
  I use stat to find out whether the directory already exists, if it doesn’t I use mkdir to make the directory if the command and arguments are valid. Depending if the -v flag is used, I print the message for each directory created.
    - If just mkdir is given without any arguments, it prints the error message “Insufficient arguments”.
    - If I don’t specify the name of the directory to be made after the option eg mkdir -p , it prints the error message “Insufficient arguments”
    - If the directory to be created already exists, it prints the error message “Directory [name] already exists”
    - If mkdir is unable to create the directory, it prints the error message “Unable to create directory”

    - If a nested directory is given eg abc/xyz and abc doesn't exist and the -p option is not used, it prints the error message "Cannot create directory [name]. No such file or directory exists"
    - If an invalid option is given, it prints the error message "Invalid argument"
- rm

  I use remove to remove the files and directory (if they're empty and -d option is used). If the -i option is used, it asks for a yes before deleting the file, and deletes only if the answer is yes.
    - I do not allow the user to delete the files that are used by my shell to run. If they're given as arguments, it prints the error message "You do not have permission to remove [name] file"
    - If you try to do rm on directories without -d option, it prints an error message "Can't do rm on directories". I use opendir() to determine if the given argument is a directory.
    - If you try to do rm on non empty directories even with the -d option, it prints an error message "Cannot remove file or directory"
    - If just rm is used without any argument, it prints the error message "Insufficient arguments"
    - If an invalid option is given, it prints the error message "Invalid argument"
- If any command other than the ones specified above are used, it prints the error message "Invalid command".

Test Cases

cd -> Goes to home
cd -t -> Invalid argument. Not supported.
cd Desktop -> Goes to Desktop
cd .. -> Goes back to home
cd joker -> Directory not found
cd ~ -> Goes to home
cd Desktop Pictures -> Too many arguments
cd Desktop/shell -> Goes to Desktop/shell

echo hello world -> Prints hello world
echo hello -> Prints hello
echo -n hello -> Prints hello without newline
echo -E hello\tworld -> Prints hellotworld
echo -E "hello\tworld" -> Prints hello\tworld
echo -> Prints newline.

history -> Prints history of current shell
history -c -> Clears history
history -> Prints history since last history clean
history -w -> Writes history to file historyfile.txt
history -w f1 -> Writes history to file f1
history -r -> Argument not supported. “Invalid argument”
history hello world -> “Invalid argument”

pwd -> Prints the current directory
pwd -r -> Invalid argument. Not supported

exit -> exit terminal

ls -> Prints contents of current directory excluding files starting with '.'

ls -a -> Prints contents of current directory including files starting with '.'
ls -r -> Prints contents of current directory in reverse order.
ls Desktop -> Prints contents of Desktop directory
ls -a Desktop -> Prints contents of Desktop directory including files starting with '.'
ls -r Desktop -> Prints contents of Desktop in reverse order.
ls haha -> Directory not found
ls -t -> Invalid argument. Not supported

date -> Print IST date
date -u -> Print UTC date
date -I -> (capital i) Print date in ISO 8601 format
date -t -> Invalid argument. Not supported

cat -> Opens cat in interactive mode
cat -n -> number all output lines in interactive mode.
cat -E -> display $ at end of each line
cat -t -> Invalid argument. Not supported
cat shell.c -> Prints contents of shell.c
cat shell.c ls.c -> Prints contents of shell.c and ls.c
cat -n shell.c -> numbers all output lines from shell.c
cat -E shell.c -> display $ at end of each line from shell.c
cat joker -> File does not exist

mkdir -> Insufficient arguments
mkdir foo -> Makes directory foo
mkdir abc/xyz -> Cannot create parent directories if they don't exist. Error.
mkdir -p abc/xyz -> No error if existing else makes parent directories as needed. Creates abc then inside it xyz.
mkdir -v hello world -> Print a message for each created directory
mkdir hello -> Directory hello exists already
mkdir -t -> Invalid argument

rm hello -> Can’t do rm on directories
rm -d hello -> Removes hello
rm -i f1 -> Asks before removing f1. If answer is y then removes.
rm -d abc -> Cannot remove non empty directory
rm -d abc/xyz abc -> Removes abc and the directory inside it which is xyz
rm -t -> Invalid argument. Not supported.