DSA Mini Project
Assignment Shell

Group : 37

Members:
shantanu.chauhan@research.iiit.ac.in
sneha.raghavaraju@students.iiit.ac.in
seetha.raghavendra@students.iiit.ac.in
vadi.neha@research.iiit.ac.in
thakkallapally.rao@students.iiit.ac.in

The project is an interactive shell that supports commands to create, update, compare Assignment submissions, etc. (All functions are described below).

Description of the functionality of each command and an example on how to use them.

- switch <subject>: Changes the name of the subject in the prompt and changes folders appropriately. Eg: switch DSA, would change the prompt from xyz/LA> to xyz>DSA and navigate to the DSA directory.

- create <assignment>: Creates a new assignment folder. Puts the contents of the dist folder and the problem statement into the current directory. Eg: create Assignment

- update <assignment>: Updates with the new assignment files. Deletes old files and replaces them with the new ones. Eg: update Assignment

- test <assignment>: Runs the submitter file in the dist folder. Store the errors in a file. Eg: test Assignment

- submit <assignment>: Makes a zip file of the assignment folder.

  The function takes the Assignment folder name as input. It zips the folder and stores it with the name 'AssignmentZip.zip', in the local directory.

  Eg: submit Assignment

- compare <assignment> <zipfile>: Compares assignment folder with the submitted zip.

  Compare function takes the Assignment folder name and the zip file name as inputs. Does md5 hash on the files in both directories and compares them to check if any of the files in the zip are different from any of the files in the assignment folder and prints a list of those files.

  Eg: compare Assignment AssignmentZip.zip

- use <assignment>: Changes the prompt to xyz/DSA/<assignment>. And for all subsequent commands, if the <assignment> argument is not passed, it will automatically use the current directory, i.e <assignment>.

  The use function takes the Assignment folder name as input. It gets the complete path of the Assignment folder and stores it in a log file so that subsequent function calls can use it.

  Eg: use Assignment

Division of Work

Sneha:
'Use' function and getPath function for subsequence calls.
'Compare' function using Md5 hash. (Handles all this inside an auxiliary directory which is deleted after use).
'Submit' function to create the final zip folder.

Neha:
'test' function.
Displaying files in a tree structure.

Raghavendra:
Handling input.

Rohan:
createDir function. (create directory function).

Shantanu:
'Switch' function.

‘Create’ function.
‘Update’ function.
‘Setup” function
Shell.c
Installer.c
Input and output handling.
Logging part of of test function
doesFileExist function
createDir, copyDir and all directoryFunc functions
makefile

Data structures Used

In the printdir function, used to display the directory structure, we make use of linked lists and stacks.

When printing the structure we must show the folders at the same levels and the files they contain. Starting from the topmost level and moving down the hierarchy, till we have traversed the entire directory in a way that is similar to a Breadth First Search.

This is implemented in the printdir function using recursion, and is similar to the functionality of a stack. In which we first push a folder, then all its subfolders, and so on.

System Calls:

This project makes use of c system calls as well as some basic shell commands like cd and pwd (change directory and print working directory).

Commands that are not supported by the shell are directed to the system using a system call. So commands that work in the terminal, work in this shell.

opendir() - This function is used to open a directory and return a pointer to the directory. It takes the pointer to the null-terminated path name of the directory to be opened as input.It returns a pointer to DIR, which is an open directoy stream(ordered sequence of all directories in the current directory).

readdir(DIR **dirp*) – The function is used to read into a directory.The function returns a pointer to a *dirent* structure.

closedir() - This function closes the directory stream indicates in the parentheses of type DIR* .It returns value 0 if it is successful else it returns -1.

system() - This function is used to pass commands that can be executed in the terminal of the OS, and returns the command after it is completed.

chdir() - The function changes the working directory of the process but doesn't change the working directory of the shell.- It takes the path of the required directory as input .It returns the value 0 iff successful else -1.

getcwd(char *buf, unsigned long size) - The function copies the pathname of the current working directory to the array pointed to by buf, which is of length size.If the length of the current path name is bigger than the buf size then it returns -1.The function returns a pointer to the string containing the current working directory if successful else returns NULL.

access(char* pathname, int mode) - This function is used to check if the calling program has access to a specified file. The path of the required file and the flag to perform the required action(F_OK) is given as input. The function returns 0 if it is successful and -1 otherwise.

mkdir(char * file) - The function is used to create a directory with a name file. The function returns 0 if it is successful and -1 otherwise.

lstat() - This function is used to determine information about a file based on its filename .On success, zero is returned. On error, -1 is returned.

sleep(unsigned int *seconds***) –** The function causes the calling thread to sleep until the number of real-time seconds have passed or until a signal arrives.It returns 0 if the requested time has passed, else the number of seconds remaining if it has been interrupted by a signal.

getenv() - The function searches the environment list to find the environment variable *name*, and returns a pointer to the corresponding *value* string. It returns a pointer to the value in the environment, or NULL if there is no match.

exit() - The function terminates the calling process normally. It performs the following operations

- Flushes unwritten buffered data.
- Closes all open files.
- Removes temporary files.
- Returns an integer exit status to the operating system.

It doesn't return anything.