P2

The aim of this problem is to develop a clustershell program.It runs in such a way that there is one main node and the other are the secondary nodes., that receives instructions from the main node.
The design is such that the program custershell_client.c is run on the main node and clustershell_server.c is run on the secondary node.

Client.c

This code is supposed to be run on the main node.
Important functions.
1)**parse_ip(FILE *fp)** parses the config file to record the ip addresses of the respective nodes and populates the relevant data structures.

2)**connect_nodes()** connects with the nodes using the ip addresses contained in the config file. It uses the data structure ***Node_address[]*** which is an array of **struct sockaddr_in** It stores the necessary information required for connecting with the respective nodes.

3)**is_local(char *input)** checks whether the entered command has to be executed in the main node or in the secondary nodes. Returns 1 if local and 0 otherwise.

4)**execute_local(char *input)** executes the entered command on the main node itself and displays the output.

5)**execute_node(char* input)** executes the entered command on multiple nodes. It starts by instructing the first node in entered command to execute its respective sub-command.
For example, if the entered command is "***n1.ls|n2.wc***" the main node sends the command "ls" to node 1. It also sends the remaining command to the node 1 so that it knows it has to send its data to node 2. Therefore, it sends the command "**ls|n2.wc**" to the node 1. The clustershell_server.c is designed in such a way that it connects with node 2 and sends its data to node 2, which executes its respective command.After instructing the first node in the entered command, the main node waits to receive a reply from the last node that is supposed to execute the command.In the above example it waits for data from node 2. The has_pipe(int len,char *input) is used to identify whether the commands needs to be executed on multiple nodes or on a single node.

All the data structures declared are global.The limit on the amount of data that can be sent is 1000 bytes.MAXLEN is the maximum number of connections ,i.e. maximum number of nodes.
Important Data Structures

1)**char *node_ip[MAXLEN]** stores the respective ip address of the nodes. node_ip[i]=ip address of node i.

2)**char read_buf[MAX_MSG_LEN]** buffer used to store the received data.

3)**int active_connections[MAXLEN]** stores a value 1 at the ith location if the node i is active else it stores 0.

MAIN Function

It starts by parsing the config file and storing the ip addresses of the nodes in the node_ip[] array. Then the function connect_nodes() is called to establish a connection between the main node and all the active nodes. It then enters an infinite loop that starts with taking the input from the user.It then checks whether the command needs to be executed on the main node or on other nodes and then calls the respective functions to do their job. The commands are executed using the execvp () system call with the first argument “-sh” i.e we execute the shell and pass our commands as arguments.The main function then waits for a reply from the last node to execute the command.

Input format
Avoid any spaces unless required in the command. The commands that need to be executed on one node should only be separated by a pipe.for example if we want to execute two commands “ls” and “wc” on node 1, we should write **n1.ls|w**c and not **n1.ls|n1.wc**

Clustershell_server.c

This code should be run on the secondary node. While executing, the id of the node, the ip address of the node and path to the config file should be given as input at the command line argument.
Important functions and data structures

1)**has_pipe(int len,char *read_buf)** checks if we the command needs to further be sent to other nodes for computation and returns the beginning from the pipe after which the command for the next node starts. Eg. if the command is **“n1.ls|n2.wc”** the

function will return “**|n2.wc”**. It returns null if there is no need to execute commands on other nodes.

2) **cd_exec(char *input)** this is used to execute the cd command using the chdir system call, which is used to change the directory of a process.

3) **char *arg_list[]** containing the arguments that needs to be passed to execvp)() system call. The first 2 values are **“sh”** and **“-c”** respectively.

4) **get_arguments(char *input)** This populates the global data structure arg_list with the commands that are required to run on a particular node.The first two entries of this data structure are as described above. The third entry is the command that needs to be executed.

5)**char *node_ip[MAXLEN]** stores the respective ip address of the nodes. node_ip[i]=ip address of node i.

6) **parse_ip(FILE *fp)** This performs the same functions as described in the previous section.Populates the **node_ip[]** and **Node_address[]** data structure.

Main Function
Each secondary node executes the server code. It has the following structure.

1)**Creator process** which is responsible for accepting a connection and create a new process to handle that connection

2)**Principal process** which is responsible for communicating with the main node. It executes the given commands and then either

passes them back to the main node or to some other node with other required information.

3)**Side process** which is created when connection is established with someone other than the main node. This executes the command and then either passes the information to some other node or back to the main node.

The main process starts by parsing the config file and creating a socket and the address structure. The ip is as specified in the command line arguments and the port number is 1024 +id which is as specified by the command line arguments. The socket is bound to this address. The main function enters into an infinite loop which denotes the creator process as specified above. It accepts connections from other nodes. The first connection is supposed to be from the main node. It accepts this connection and creates the **principal process** as mentioned above which waits for a message from the main node.

If a command is received from the main node. If computation in other nodes is required, It carries out the following steps.

1) Extract the command from the second node onwards. i.e. If the command is **n1.ls|n2.wc** it will extract **n2.wc**
2) Connect to the other node and receive an acknowledgement for the connection, and then send the command to the next node. After sending this command, It creates a new process and executes the command assigned to it and the output of this command is directly sent to the other node. Meanwhile, the other node is waiting for data from the previous process.

If the received command(from the main node) does not require computation on any other node, the computation is carried out and the data is directly sent to the main node. In both cases the

main node is waiting for the data from the last node that does the computation.

If the creator process receives a connection from a node other than the main node. It creates a **side process** as described before. This process performs the same tasks as the principal node. But, It first receives the further commands to be executed and then receives the data on which it has to execute its own command. It then performs the steps similar to the principal process.Of checking whether computation on other nodes is needed or not. Note that the correct socket fd for communication with the main node is saved as soon as the principal process is created hence all the subsequent processes can communicate directly with the main node.

For execution of the cd command, a special function named **cd_exec(char *input)** is created which uses the system call **chdir()** and sends the message "**directory changed"** to the main node.

Notes on execution
For running the files, the following instructions should be followed.

Client.c
The path to the config file should be given as the command line argument.

Server.c
The first command line argument is the id of the node, i.e the number using which we identify the node. The next argument should be the IP address of the node and the third argument should be the path to the config file.