Project Members: Zeynep Sıla Kaya (69101)  
  
Part I:  
In this part we only changes given execvp to execv. We need to modify the path like /bin/command\_name. So we concatenate the desired command to initial path. And used the modified path in execv.  
  
Part II:  
shortdir set <name> . This command gives a name to the current directory. Each name should be unique. A name cannot associate with different directories.  
If the user gives another name to existing association, command overwrites it.  
shortdir jump <name> . This command changes the directory which associated with the name in the command.  
shortdir del <name> . This command deletes the name-directory association  
shortdir clear . This command deletes all the associations.  
shortdir list . This command shows all existing name-directory associations.  
  
  
Part III:  
The user should use the command like this: "highlight <desired\_word\_name\_to\_be\_highlighted> <the color> <the file name> This command highlights the desired word in desired color. We created one array for getting the lines of the file(sentences), one array for the copy of the lines(sentencescpy). We created the copy in order to prevent changes in the original one. We have to show the sentences which contains the desired word. Therefore, we created another array called wantedSen to store them. We tokenized the words in the file with punctuations (e.g. awesome! is a token) and highlighted them.  
  
  
Part IV:  
User should use this command : "goodMorning <desired\_time> <path\_of\_the\_music>  
This command plays the desired music at desired time. This implementation uses crontab to control the ryhtmbox application. We get the desired time by spliting the first argument (0th have to be goodMorning) with ".". We created a separate cronFile to write the time and the code for opening the ryhtmbox in, according to the syntax of crontab. With execv() we managed to give the desired input to crontab.  
  
Part V:  
User should use this command "kdiff -a/-b/null file1name/path file2name/path"  
kdiff -a and only kdiff command will function in the same way. In this type of kdiff, user have to provide txt files. If user provides only the name, the file should be in the same directory as the source code. Otherwise, the user should provide the path of the files. It will not accept the other files. It will find which files are different from each other and show them to user with difference count. If they are identical, the user will be informed. In kdiff -a part, line index starts from 0.  
kdiff -b part is for the compare two files by byte to byte and works in binary files. It will only show how much the difference between two files.If they are identical, the user will be informed. If the size of the files are different, the difference between them should be counted. However, since the null terminator of the small file and some byte in the larger file is already compared, we should subtract one.   
  
Part VI:  
User should use this command as : "wordcount <only\_filename>"  
The second argument is for the file name. This file should be txt file but user should only give the name (e.g. without .txt extension)(or path without the .txt extension). For example, "wordcount file1" .The method will find the corresponding txt file. The file should be written in organized way. The file should contain proper amount of spaces.  
This wordcount aims to count the number of words and lines in the given file. It informs the user by printing the results.  
We used the idea of counting the space characters for getting the number of words and lines. For example, for each line there should be a new line char or EOF(if its the last line of the file). Also, there should be a new line, space, EOF etc. for each word.



