```c
/* Created By: Abhinandan (Abhi) Jagdev
 * Last Modified: 10-10-2014
 * Description: This project allows the user to execute some basic commands like ls,
                    open etc in the basic shell format; along with that it
                    also allows the user to change directories and
                    executes some basic commands in background.
 * Some Functions Used:
 *      wait(): used to wait for basic command execution after fork()
 *      fork(): used to create child process to create/allow
                execution of the programs
 *      waitpid(): used for keeping track of background operations
 */



// Includes
#include <unistd.h>     // Symbolic Constants
#include <sys/types.h>  // Primitive System Data Types
#include <errno.h>      // Errors
#include <stdio.h>      // Input/Output
#include <sys/wait.h>   // Wait for Process Termination
#include <stdlib.h>     // General Utilities
#include <time.h>
#include <string.h>


int change_dir(char *cmdLine, char *current_dir ){

    /* cd == cd~ takes you to the home directory of the system*/
    if ((strcmp(cmdLine, "cd")) == 0) {

        /* get the environemnt of the system to go home when "cd" is pressed*/
        char *current_env = getenv("HOME");

        /* error occurred */
        if (current_env==NULL){
            fprintf(stderr, "ERROR: Could not find the home directory\n");
            exit(1);
        }

        /* change the direcotry to home*/
        int chdir_result = chdir(current_env);

        /* check the result for chdir(......)*/
        if (chdir_result==0) {
            current_dir = getcwd(NULL, 64);
        } else {
            fprintf(stderr, "ERROR: Could not change the directory\n");
            exit(1);
        }

    /* execute change directory and only go back one directory*/
    } else if ( (strcmp(cmdLine, "cd ..")) == 0) {
        int chdir_result = chdir("..");

```

```
56          /* check the result for chdir(......)*/
57          if (chdir_result==0) {
58              current_dir = getcwd(NULL, 64);
59          } else {
60              fprintf(stderr, "ERROR: Could not change the directory\n");
61              exit(1);
62          }
63
64      /*go to the specified directory*/
65      } else {
66
67          char *token;
68          const char s[2] = " ";
69
70          /* get the first token */
71          token = strtok(cmdLine, s);
72          token = strtok(NULL, s);
73
74          /* grab the token for the new directory*/
75          char *new_dir_name = token;
76
77          /* change to the specified directory*/
78          int chdir_result = chdir(new_dir_name);
79
80          /* check the result for chdir(......)*/
81          if (chdir_result==0) {
82              current_dir = getcwd(NULL, 64);
83          } else {
84              fprintf(stderr, "ERROR: Directory not found\n");
85          }
86      } /* end if for cd == 0*/
87
88      return 0;
89  }
90
91
92  int main(int argc, char* argv[])
93  {
94      char *cmdLine = NULL;
95      size_t sizecmdLine = 0;
96      int len_cmdLine;
97      char *current_dir;
98
99      while(1) {
100         /*get and print the current directory and its status*/
101         current_dir = getcwd(NULL, 64);
102         if(current_dir==NULL) {
103             perror("pwd");
104             exit(0);
105         }
106         printf("RSI: %s > ", current_dir);
107
108         /*gets the comand from the user and stores as string*/
109         len_cmdLine = getline(&cmdLine, &sizecmdLine, stdin);
110         cmdLine[len_cmdLine-1] = '\0';
```

```
111         int cmpS;
112 
113         /* check if cd command is initiated  */
114         if ((strncmp(cmdLine, "cd", 2)) == 0) {
115 
116             /* call the ch_dir function to execute the cd commands
117              with input commands and current directory*/
118             int ch_dir;
119             ch_dir = change_dir( cmdLine, current_dir );
120 
121             /*check for the backgroud execution command*/
122         } else if ((strncmp(cmdLine, "bg", 2)) == 0) {
123 
124             if ((strncmp(cmdLine, "bg ", 3)) == 0) {
125 
126             /*count the amount of tokens in the command line*/
127                 int amt_tokens=0;
128                 int k = 0;
129                     for(k = 0; k < strlen(cmdLine); k++) {
130                         if (cmdLine[k]==' ') {
131                             amt_tokens++;
132                             continue;
133                         }
134                     }
135 
136                 char *token;
137                 const char s[2] = " ";
138 
139                 /* get the first token */
140                 token = strtok(cmdLine, s);
141                 token = strtok(NULL, s);
142 
143                 char * new_cmd = NULL;
144                 new_cmd = token;
145 
146                 int i = 0; // counter for cmdArray
147 
148                 /* allocate memory for the cmdArray*/
149                 char** cmdArray = (char**) malloc(sizeof(char*) * (amt_tokens+1));
150 
151                 /* get the tokens and store them in the cmdArray */
152                 while( token != NULL ) {
153                     cmdArray[i++] = token;
154                     token = strtok(NULL, s);
155                 }
156 
157                 /* create an new process */
158                 pid_t pid, w;
159                 pid = fork();
160 
161                 /* check the proccess creation for success or failure  */
162                 if (pid<0) {
163 
164                     fprintf(stderr, "Fork Failed - Process not created");
165                     exit(-1);
```

```
166
167                 }  else if (pid == 0) {
168                     /* execute the commands given (background execution) */
169                     execvp(new_cmd, cmdArray);
170
171                 } else {
172
173                     printf("Waitpid reached \n");
174                     /*use waitpid(pid, status, opt) for background execution*/
175                     w = waitpid(pid, NULL, WNOHANG);
176                     printf("Waitpid number: %d, %d \n", w, pid);
177
178                     //error checking for waitput
179                     if (w == -1) {
180                         perror("waitpid");
181                         exit(0);
182
183                     } else if (w == 0) {
184                     /*add the process to list data to be printed*/
185
186                     } else if (w == pid) { /*child ended process finished*/
187                     /* display the that process finished and update ur bglist */
188
189                     }
190                 }
191                 /*print the list of current jobs in the background and its pid*/
192             } else if ((strcmp(cmdLine, "bglist")) == 0) {
193
194             printf("Print the jobs in background\n");
195             }
196
197         /* execute the basic commands from the library of commands in bin */
198         } else {
199
200             /*count the amount of tokens in the command line*/
201             int amt_tokens=0;
202             int k = 0;
203             for(k = 0; k < strlen(cmdLine); k++) {
204                 if (cmdLine[k]==' ') {
205                     amt_tokens++;
206                     continue;
207                 }
208             }
209
210             char *token;
211             const char s[2] = " ";
212
213             /* get the first token */
214             token = strtok(cmdLine, s);
215
216             /* compare if the first line is exceptable
217              * with commands for the basic execution*/
218             cmpS = strcmp(token, cmdLine);
219
220             /* array of tokens for the execvp */
```

```
221             int i = 0;
222             char** cmdArray = (char**) malloc(sizeof(char*) * (amt_tokens+1));
223 
224             /* walk through other tokens */
225             while( token != NULL ) {
226                 cmdArray[i++] = token;
227                 token = strtok(NULL, s);
228             }
229 
230             if (cmpS == 0) {
231                 pid_t pid;
232                 pid=fork();
233 
234                 /* create the child process and then execute the basic operation*/
235                 if (pid<0) {
236                     fprintf(stderr, "Fork Failed - Process not created\n");
237                     exit(-1);
238                 }  else if (pid == 0) {
239                 /* execute the commands given (basic commands)*/
240                     execvp(cmdLine, cmdArray);
241                 } else {
242                     wait(NULL);
243                 }
244 
245             }/*end if for cmpS*/
246         } /*end basic commands else*/
247     } /*end while*/
248     return(0);
249 }
250 
```