```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("..");

        /* 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);
        }

    /*go to the specified directory*/
    } else {

        char *token;
        const char s[2] = " ";

        /* get the first token */
        token = strtok(cmdLine, s);
        token = strtok(NULL, s);

        /* grab the token for the new directory*/
        char *new_dir_name = token;

        /* change to the specified directory*/
        int chdir_result = chdir(new_dir_name);

        /* check the result for chdir(......)*/
        if (chdir_result==0) {
            current_dir = getcwd(NULL, 64);
        } else {
            fprintf(stderr, "ERROR: Directory not found\n");
        }
    } /* end if for cd == 0*/

    return 0;
}


int main(int argc, char* argv[])
{
    char *cmdLine = NULL;
    size_t sizecmdLine = 0;
    int len_cmdLine;
    char *current_dir;

    while(1) {
        /*get and print the current directory and its status*/
        current_dir = getcwd(NULL, 64);
        if(current_dir==NULL) {
            perror("pwd");
            exit(0);
        }
        printf("RSI: %s > ", current_dir);

        /*gets the comand from the user and stores as string*/
        len_cmdLine = getline(&cmdLine, &sizecmdLine, stdin);
        cmdLine[len_cmdLine-1] = '\0';
        int cmpS;

        /* check if cd command is initiated  */
        if ((strncmp(cmdLine, "cd", 2)) == 0) {

            /* call the ch_dir function to execute the cd commands
             with input commands and current directory*/
            int ch_dir;
            ch_dir = change_dir( cmdLine, current_dir );

            /*check for the backgroud execution command*/
        } else if ((strncmp(cmdLine, "bg", 2)) == 0) {

            if ((strncmp(cmdLine, "bg ", 3)) == 0) {

            /*count the amount of tokens in the command line*/
                int amt_tokens=0;
                int k = 0;
                    for(k = 0; k < strlen(cmdLine); k++) {
                        if (cmdLine[k]==' ') {
                            amt_tokens++;
                            continue;
                        }
                    }

                char *token;
                const char s[2] = " ";

                /* get the first token */
                token = strtok(cmdLine, s);
                token = strtok(NULL, s);

                char * new_cmd = NULL;
                new_cmd = token;

                int i = 0; // counter for cmdArray

                /* allocate memory for the cmdArray*/
                char** cmdArray = (char**) malloc(sizeof(char*) * (amt_tokens+1));

                /* get the tokens and store them in the cmdArray */
                while( token != NULL ) {
                    cmdArray[i++] = token;
                    token = strtok(NULL, s);
                }

                /* create an new process */
                pid_t pid, w;
                pid = fork();

                /* check the proccess creation for success or failure  */
                if (pid<0) {

                    fprintf(stderr, "Fork Failed - Process not created");
                    exit(-1);

                }  else if (pid == 0) {
                    /* execute the commands given (background execution) */
                    execvp(new_cmd, cmdArray);

                } else {

                    printf("Waitpid reached \n");
                    /*use waitpid(pid, status, opt) for background execution*/
                    w = waitpid(pid, NULL, WNOHANG);
                    printf("Waitpid number: %d, %d \n", w, pid);

                    //error checking for waitput
                    if (w == -1) {
                        perror("waitpid");
                        exit(0);

                    } else if (w == 0) {
                    /*add the process to list data to be printed*/

                    } else if (w == pid) { /*child ended process finished*/
                    /* display the that process finished and update ur bglist */

                    }
                }
                /*print the list of current jobs in the background and its pid*/
            } else if ((strcmp(cmdLine, "bglist")) == 0) {

            printf("Print the jobs in background\n");
            }

        /* execute the basic commands from the library of commands in bin */
        } else {

            /*count the amount of tokens in the command line*/
            int amt_tokens=0;
            int k = 0;
            for(k = 0; k < strlen(cmdLine); k++) {
                if (cmdLine[k]==' ') {
                    amt_tokens++;
                    continue;
                }
            }

            char *token;
            const char s[2] = " ";

            /* get the first token */
            token = strtok(cmdLine, s);

            /* compare if the first line is exceptable
             * with commands for the basic execution*/
            cmpS = strcmp(token, cmdLine);

            /* array of tokens for the execvp */
            int i = 0;
            char** cmdArray = (char**) malloc(sizeof(char*) * (amt_tokens+1));

            /* walk through other tokens */
            while( token != NULL ) {
                cmdArray[i++] = token;
                token = strtok(NULL, s);
            }

            if (cmpS == 0) {
                pid_t pid;
                pid=fork();

                /* create the child process and then execute the basic operation*/
                if (pid<0) {
                    fprintf(stderr, "Fork Failed - Process not created\n");
                    exit(-1);
                }  else if (pid == 0) {
                /* execute the commands given (basic commands)*/
                    execvp(cmdLine, cmdArray);
                } else {
                    wait(NULL);
                }

            }/*end if for cmpS*/
        } /*end basic commands else*/
    } /*end while*/
    return(0);
}

```