```
#include <string.h>
#include <stdio.h>
#include <stdlib.h>
#include <errno.h>
#include <sys/wait.h>
#include <sys/time.h>
#include <sys/resource.h>
#include <fcntl.h>

ssize_t lineSize = 0;
char *line = NULL;

int numwords;
char *options;
//==================================
char **splitLine(char *line){

  int pos = 0;
  char **words = malloc(64 *sizeof(char*));
  char *word;
  word = strtok(line, " \t\n");
  while( word != NULL){

    words[pos] = word;
    pos++;

    word = strtok(NULL, " \t\n");
  }
  words[pos+1] = NULL;
  numwords = pos;

  return words;
}
//==================================
char **checkRedirect(char **wordes){
  int redIN = -1;
  int redOUT = -1;
  int redERR = -1;
  int i =0;
  char file[64];
  int fdIN = 0;
  int fdOUT = 0;
  int fdERR = 0;
  for(i = 0; i< numwords; i++){
    strcpy(file, wordes[i]);

    if(wordes[i][0] == '<'){

      strcpy(file,&file[1]);
      if( (fdIN = open(file,O_RDONLY)) < 0){
        fprintf(stderr, "Error '%s': %s, Errno: %d\n", file, strerror(errno), errno);
        exit(EXIT_FAILURE);
      }

      if(dup2(fdIN,0) <0){
        fprintf(stderr, "Error '%s': %s, Errno: %d\n", file, strerror(errno), errno);
        exit(EXIT_FAILURE);
      }
      redIN = i;
    }

    if(file[0] == '>' && file[1] != '>'){
      strcpy(file,&file[1]);
      printf("file to REDIRECT STDOUT: %s\n", file);
      if((fdOUT = open(file, O_CREAT | O_TRUNC| O_WRONLY)) <0){
        fprintf(stderr, "Error '%s': %s, Errno: %d\n", file, strerror(errno), errno);
        exit(EXIT_FAILURE);
```

```
      }
      if(dup2(fdOUT,1)<0){
        fprintf(stderr, "Error '%s': %s, Errno: %d\n", file, strerror(errno), errno);
        exit(EXIT_FAILURE);
      }
      redOUT = i;
     }

    if((file[0] == '>') && (file[1] == '>')){
      strcpy(file,&file[2]);
      printf("file to REDIRECT STDOUT: %s\n", file);
      if((fdOUT = open(file, O_CREAT | O_APPEND| O_WRONLY)) <0){
        fprintf(stderr, "Error '%s': %s, Errno: %d\n", file, strerror(errno), errno);
         exit(EXIT_FAILURE);
      }
      if(dup2(fdOUT,1)<0){
        fprintf(stderr, "Error '%s': %s, Errno: %d\n", file, strerror(errno), errno);
        exit(EXIT_FAILURE);
      }
      redOUT = i;
    }

    if((file[0] == '2') && (file[1] == '>')){
      strcpy(file,&file[2]);
      printf("file to REDIRECT STDERR: %s\n", file);
      if((fdERR = open(file, O_CREAT | O_TRUNC| O_WRONLY)) <0){
        fprintf(stderr, "Error '%s': %s, Errno: %d\n", file, strerror(errno), errno);
         exit(EXIT_FAILURE);
      }
      if(dup2(fdERR,2)<0){
        fprintf(stderr, "Error '%s': %s, Errno: %d\n", file, strerror(errno), errno);
        exit(EXIT_FAILURE);
      }
      redERR = i;
    }

    if((file[0] == '2') && (file[1] == '>') && (file[2] == '>')){
      strcpy(file,&file[2]);
      printf("file to REDIRECT STDERR: %s\n", file);
      if((fdERR = open(file, O_CREAT | O_APPEND| O_WRONLY)) <0){
        fprintf(stderr, "Error '%s': %s, Errno: %d\n", file, strerror(errno), errno);
         exit(EXIT_FAILURE);
      }
      if(dup2(fdERR,2)<0){
        fprintf(stderr, "Error '%s': %s, Errno: %d\n", file, strerror(errno), errno);
        exit(EXIT_FAILURE);
      }
      redERR = i;
    }

    }//END OF FOR LOOP
    char **retWords = malloc(64*sizeof(char*));
    if(!retWords)
      exit(EXIT_FAILURE);
    int e = 0;
    for(e = 0; e < numwords; e++){
      //printf("foor loop #%d\n", e);
      if(!((redOUT == e)||(redIN ==e) ||(redERR == e))){
        retWords[e] = wordes[e];
      }
    }


  return retWords;
}
//==================================
int parseline(char **wordes){
```

```
  if(numwords > 1){
    if(wordes[1][0] == '-'){
      options = wordes[1];
    }
  }
  int j = 0;
  char **parsedW = malloc(64 * sizeof(char*));
  /*for(j = 0; j<numwords; j++){


  }*/
  return 1;
}
//================================================
int launchNewProcess(char **wordes){

  pid_t childPID, parentPID;
  struct rusage ru;
  int status;
  char** commands;

  childPID = fork();
  //printf("childPID:%d\n", (int) childPID);
  if(childPID == 0){
    commands = checkRedirect(wordes);
//    printf("commands[0] = %s\n", commands[0]);
    int n = sizeof(commands)/sizeof(char*);

    //printf("n = %d\n", n);
    //printf("In the child process\n");

    if(execvp(commands[0],commands) < 0){
      fprintf(stderr, "Error from child '%s': %s, Errno: %d\n", commands[0],strerror(errno), errno);
    exit(EXIT_FAILURE);
    }
  }
  else if(childPID < 0){
    fprintf(stderr, "Error: %s, Errno: %d\n", strerror(errno), errno);
  } else {

      if(wait3(&status,0,&ru) ==-1)
        fprintf(stderr, "Error 'wait3': %s, Errno: %d\n", strerror(errno), errno);
      else
        fprintf(stderr, "CPU:  %ld.%03d | User: %ld.%03d | Real:%ld.%03d\n",    ru.ru_stime.tv_sec,
(int)ru.ru_stime.tv_usec, ru.ru_utime.tv_sec,(int)ru.ru_utime.tv_usec, ru.ru_stime.tv_sec +
ru.ru_utime.tv_sec, (int)ru.ru_stime.tv_usec + (int)ru.ru_utime.tv_usec);
  }

  return 1;
}

int main (){
  int builtin = 0;
  while(1){
    builtin = 0;
    if( getline(&line, &lineSize, stdin) < 0)
      perror( strerror(errno));
    char **wordes = NULL;

    wordes = splitLine(line);


    if(wordes[0][0] == '#'){
      printf("comment encountered\n");
      continue;
    }
```

```
    if(!strcmp(wordes[0],"cd")){
      builtin = 1;
      if(chdir(wordes[1]))
        fprintf(stderr,"Error: %s, Errno: %d\n", strerror(errno),errno);
      else
        fprintf(stderr, "Directory changed to '%s'\n", wordes[1]);
    }
    if(!strcmp(wordes[0],"exit")){
      builtin = 1;
      if(wordes[1] != NULL)
        exit (atoi(wordes[1]));
      exit(0);
    }

    parseline(wordes);

    if(builtin == 0)
      launchNewProcess(wordes);

  } // END OF infinite while


  return 0;
}
```