```c
/*
 * Mshell.c
 *
 *  Created on: Oct 22, 2017
 *  Author: scott
 *  brief: Mshell (Mini SHell)
 */
#include <sys/wait.h>
#include <unistd.h>
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include <errno.h>
#include <fcntl.h>
#include <sys/time.h>
#include <sys/resource.h>
#include <sys/types.h>
char ** Mshell_split_line(char *line,char ** ReIn,char ** Re,char ** ReErr,char ** ReApp,
    char ** ReAErr, int *estatus);
int IOredir(const char* path, int fdnew, int oflags, mode_t mode);
int Mshell_processline(char *line, int *estatus,int *errcode);

int main(int argc, char **argv){
  //initialization
  FILE *fdIN;
  char *line=NULL;
  int readin=0,linenum=0,errcode=0,estatus=0;
  size_t buffersize=0;//let getline realloc()
  //check arguments and set fdIN
  if(argc>1){
    if(argc!=2) fprintf(stderr,"Warning:Too␣many␣arguments␣provided:␣only␣first␣one␣will␣be␣
        processed:%s",argv[1]);
    if((fdIN=fopen(argv[1], "r"))==NULL){
    fprintf(stderr,"Critial␣Error␣in␣opening␣target␣file␣in␣read␣mode:%s:%s\n",argv[1],
        strerror(errno));
    exit(EXIT_FAILURE);
    }
  }else if(argc==1){
    fdIN=stdin;
    fputs("$␣",stdout);//change PS1
  }
  //reading from target file discripter and processing
  while((readin=getline(&line, &buffersize, fdIN))>=0){//return -1 on EOF or error
    if(fdIN==stdin) fputs("$␣",stdout);
    linenum++;
    if(readin<=1||line[0]=='#'||line[readin-1]!='\n'){  //emptyline,comment,or not newline
        delimited
      errno=0; //setting errono for error check
      continue; //skip this line
    }
    Mshell_processline(line,&estatus,&errcode);//got line parse and put it into list
      if(estatus!=0){
          fprintf(stderr,"\nError:Execuation␣Error␣existed␣for␣line␣number␣#␣%d:␣likely␣
              abortted.\n",linenum);
      }
  }
  if(errno!=0){ //error occurs
    fprintf(stderr, "Error␣excuting␣getline()␣for␣line:␣%s,line␣number:%i\n",strerror(errno)
        ,linenum);
    return errno;
  }else{
    fprintf(stderr, "End␣of␣file␣read,␣exiting␣shell␣with␣exit␣code:%i\n",errcode);
    printf("\nExecuation␣Completed.\n");
  }
    return errcode;
}
```

```
61  int IOredir(const char* path, int fdnew, int oflags, mode_t mode){
62    int fdold;
63    if((fdold=open(path,oflags,mode))<0){
64      fprintf(stderr,"Warning:Error␣in␣opening␣target␣file␣:%s:%s\n",path, strerror (errno));
65      return EXIT_FAILURE;       //skipping redirection
66    }
67    if (dup2 (fdold, fdnew) < 0) {
68      fprintf(stderr, "Warning:Error␣in␣dup2␣target␣file␣discripter:=%d␣dup2()␣failure:␣%s\n",
            fdold, strerror (errno));
69      return EXIT_FAILURE;
70    }
71    if (close(fdold)<0){
72      fprintf (stderr, "Warning:Error␣in␣closing␣file(%s):%s[dangling␣file␣discripter␣exits]",
            path, strerror (errno));
73      return EXIT_FAILURE;
74    }
75    return 0;
76  }
77  int Mshell_processline(char *line, int* estatus,int *errcode){
78    char * ReIn = NULL, * Re = NULL, * ReErr = NULL, * ReApp = NULL, * ReAErr = NULL; //IO
          keys
79    char ** tokens=Mshell_split_line(line,&ReIn,&Re,&ReErr,&ReApp,&ReAErr,estatus);
80    pid_t pid;
81    struct rusage rusage;
82    struct timeval t1, t2;
83    if (!strcmp (tokens[0], "cd")) {
84        if(tokens[1] == NULL){
85          if(chdir (getenv ("HOME")) < 0){  //defualt by shell[cd ]
86            fprintf (stderr, "ERROR-->cd␣failure␣in␣chdir:␣%s\n", strerror (errno));
87            *estatus = 1;
88            return EXIT_FAILURE;
89          }
90        }else{
91          if(chdir (tokens[1])<0){
92            fprintf (stderr, "ERROR-->cd␣failure␣in␣chdir:␣%s\n", strerror (errno));
93            *estatus = 1;
94            return EXIT_FAILURE;
95          }
96        }
97      return EXIT_SUCCESS;
98    }
99    if (!strcmp (tokens[0], "exit")) {
100     if(tokens[1] != NULL) *errcode =atoi(tokens[1]);
101     if(tokens[2] != NULL) fprintf (stderr, "Warning:␣only␣first␣argument(%s)␣will␣be␣set␣to␣
            the␣error␣code␣for␣command␣exit\n",tokens[1]);
102     exit(*errcode); //last errcode unless specified by the commmand
103   }
104   //get timestamp
105   if (gettimeofday(&t1, NULL) < 0) {
106       fprintf (stderr, "ERROR:␣gettimeofday␣failure␣for␣command[%s]:␣%s\n", tokens[0],
              strerror (errno));
107       *estatus = 1;
108       return EXIT_FAILURE;
109   }
110   int waitStatus,T; // T for time difference, errcode check child return signal
111   pid = fork();  //fork
112   if (pid == 0) {
113   // Child process:IO redirection
114     if (ReAErr != NULL) { //aborting if fail
115       if (IOredir (ReAErr, 2, O_RDWR | O_APPEND | O_CREAT, 0666)){
116         *estatus=1;
117         return EXIT_FAILURE;
118       }
119     } else if (ReErr != NULL)
120       if (IOredir (ReErr, 2, O_RDWR | O_TRUNC | O_CREAT, 0666)){
121         *estatus=1;
122         return EXIT_FAILURE;
123       }
```

```
124       if (ReApp != NULL) {
125         if (IOredir (ReApp, 1, O_RDWR | O_APPEND | O_CREAT, 0666)){
126           *estatus=1;
127           return EXIT_FAILURE;
128         }
129       } else if (Re != NULL)
130         if (IOredir (Re, 1, O_RDWR | O_TRUNC | O_CREAT, 0666)){
131           *estatus=1;
132           return EXIT_FAILURE;
133         }
134       if (ReIn != NULL && IOredir (ReIn, 0, O_RDONLY, 0666)){
135         *estatus=1;
136         return EXIT_FAILURE;
137       }
138       if (execvp (tokens[0],tokens)==-1) { //exec
139         fprintf (stderr, "ERROR-->execvp␣failure␣for[%s]:␣%s\n", tokens[0], strerror (errno));
140         *estatus=1;
141         return EXIT_FAILURE;
142       }
143       exit(EXIT_FAILURE);//should never reach here
144     } else if (pid < 0) {
145     // Error forking
146       fprintf (stderr, "ERROR-->fork␣failure␣for␣[%s]:␣%s\n", tokens[0], strerror (errno));
147       *estatus = 1;
148       return EXIT_FAILURE;
149     } else {
150     // Parent process
151     if (wait4 (pid, &waitStatus, 0, &rusage) > 0) { //wait for the specific process
152       if ((*errcode = WEXITSTATUS (waitStatus)) != 0) *estatus = 1;
153       if (gettimeofday(&t2, NULL) < 0) {
154             fprintf (stderr, "ERROR:␣gettimeofday␣failure␣for␣command[%s]:␣%s\n", tokens[0],
                  strerror (errno));
155         *estatus = 1;
156       }
157       //Printing all the info
158       T = (t2.tv_sec * 1000000 + t2.tv_usec) - (t1.tv_sec * 1000000 + t1.tv_usec);
159       fprintf (stderr, "\n[%s]Command␣returned␣with␣return␣code:\t%d\n",tokens[0], *errcode);
160       fprintf (stderr,"Consuming␣Time:\n");
161       fprintf (stderr, "␣TIME->real:\t%d.%04ds\n", T / 1000000, T % 1000000);
162       fprintf (stderr, "␣TIME->usr:\t%ld.%04ds\n", rusage.ru_utime.tv_sec, rusage.ru_utime.
              tv_usec);
163       fprintf (stderr, "␣TIME->sys:\t%ld.%04ds\n", rusage.ru_stime.tv_sec, rusage.ru_stime.
              tv_usec);
164     }else{
165       fprintf (stderr, "ERROR:␣wait4␣failure␣for␣[pid=%d␣]:␣%s\n", pid, strerror (errno));
166     }//fork exec concluded here;
167     }
168     return EXIT_SUCCESS;
169   }
170   char ** Mshell_split_line(char *line,char ** ReIn,char ** Re,char ** ReErr,char ** ReApp,
          char ** ReAErr, int *estatus){
171     int bufsize = 1024, position = 0;
172     char* offset=0;
173     char **tokens = malloc(bufsize * sizeof(char*));
174     char *token, **tokens_reserve;
175     if (tokens==NULL) {
176       fprintf(stderr, "Critical␣Error:␣Malloc␣failture-->␣Not␣enough␣space␣left␣for␣processing
              ");
177       *estatus=1;
178       exit(EXIT_FAILURE);
179     }
180     line[strlen (line) - 1] = 0; /* remove \n -->ls\n is no a command*/
181     token = strtok(line, "␣");
182     while (token != NULL) {
183       if ((offset=strstr (token, "<")) && offset==token)  *ReIn = token + 1;
184       else if ((offset=strstr (token, ">")) && offset==token)  *Re = token + 1;
185       else if ((offset=strstr (token, "2>")) && offset==token) *ReErr = token + 2;
186       else if ((offset=strstr (token, ">>")) && offset==token) *ReApp = token + 2;
```

```
187     else if ((offset=strstr (token, "2>>")) && offset==token) *ReAErr = token + 3;//ignore
            2>>
188     else tokens[position++] = token;
189     if (position >= bufsize) {
190         bufsize += 1024;
191         tokens_reserve = tokens;
192         tokens = realloc(tokens, bufsize * sizeof(char*));
193         if (tokens==NULL) {
194         free(tokens_reserve);
195           fprintf(stderr, "Critical␣Error:␣Malloc␣failture-->␣Not␣enough␣space␣left␣for␣
                processing");
196           *estatus=1;
197         exit(-1);
198         }
199     }
200     token = strtok(NULL, "␣");
201   }
202   tokens[position] = NULL;//append the null terminator
203   return tokens;
204 }
```