/\*

\* run\_command.c : do the fork, exec stuff, call other functions

\*/

#include <stdio.h>

#include <stdlib.h>

#include <sys/types.h>

#include <signal.h>

#include <errno.h>

#include <sys/wait.h>

#include "shell.h"

#include <unistd.h>

#include <string.h>

void run\_command(char \*\*myArgv) {

pid\_t pid;

int stat;

int run\_in\_background;

/\*

\* Check for background processing.

\* Do this before fork() as the "&" is removed from the argv array

\* as a side effect.

\*/

run\_in\_background = is\_background(myArgv);

set\_timer();

switch(pid = fork()) {

/\* Error. \*/

case -1 :

perror("fork");

exit(errno);

/\* Parent. \*/

default :

if (!run\_in\_background) {

//waitpid(pid,&stat,0); /\* Wait for child to terminate. \*/

if(waitpid(pid,&stat,0)>0){

stop\_timer();

}

if (WIFEXITED(stat) && WEXITSTATUS(stat)) {

fprintf(stderr, "Child %d exited with error status %d: %s\n",

pid, WEXITSTATUS(stat), (char\*)strerror(WEXITSTATUS(stat)));

} else if (WIFSIGNALED(stat) && WTERMSIG(stat)) {

fprintf (stderr, "Child %d exited due to signal %d: %s\n",

pid, WTERMSIG(stat), (char\*)strsignal(WTERMSIG(stat)));

}

}

return;

/\* Child. \*/

case 0 : /\*//fd[0]=read child process , fd[1]=write child process \*/

/\* Redirect input and update argv. \*/

redirect\_in(myArgv);

/\* Redirect output and update argv. \*/

redirect\_out(myArgv);

pipe\_and\_exec(myArgv);

exit(errno);

}

}