List of Files

Filename	Purpose
Builtin.c	Source code for built-in functions for shell
Builtin.h	Header file for builtin.c
Builtin.o	Object file for linking to Unix_Shell
Command.c	Source code for command breakdown for shell
Command.h	Header file for command.c
Command.o	Object file for linking to Unix_Shell
Token.c	Source code to break up a string into tokens for shell
Token.h	Header file for token.c
Token.o	Object file for linking to Unix_Shell
Unix_Shell.c	Source code for the Unix shell
Unix_Shell.o	Object file for Unix_Shell
Unix_Shell	Executable for Unix shell
Makefile	Make file to assemble compile and create executable for Unix shell

The project title and a brief description of the project

Design and implement a simple UNIX shell program using the grammar specified in the later part of this section. Please allow for at least 100 commands in a command line and at least 1000 arguments in each command.

Self-diagnosis and evaluation

Functional features:

- Reconfigurable shell prompt: the user can change the prompt from ‘%’ to a string entered after the built-in command ‘prompt’ [e.g. % prompt john$]
- Built-in command pwd: prints the current directory of the shell process.
- Directory walk: the user can change the directory location using ‘cd’ and if no path is stated will direct to the home directory.
- Wildcard characters: the shell allows wild card characters such as ‘*’ and ‘?’ using the C function ‘glob’
- Shell pipeline: processes can be pipelined into one another using the ‘|’ indicator. Output from one process is inputted into the next.
- Background job execution: processes can be executed in the background using the ‘&’ indicator
- Sequential job execution: processes can be executed in sequential order (one after the other) with the ‘;’ indicator
- The shell environment is inherited from the parent process
- The shell has the build in command ‘exit’ which exits the shell.
- The shell cannot be terminated by the following inputs “CTRL-C”, “CTRL-\”, or “CTRL-Z”

Non-functional features:

- Standard input and output redirection: although input/output redirection works majority of the time there are instances where the commands output unintended error messages, or the redirect will not work at all.

Discussion of your solution

Firstly, the program set a signal mask so that SIGINT, SIGQUIT and SIGTSTP are ignored. After this, it enters a loop which only exits once a command execution returns a status of 1 (exit). In the while loop if first gets the user input from the command line. Once a string is collected it allocates memory to **token. The function tokenise() is called. The tokenise() function separates the string using strtok() with " \t\n" as the eliminators. These tokens are placed into the token array.

Next, the token is separated into commands using the function separateCommands(). SeparateCommands fills into the command data structure with the index of the first token in the token array, the index of the last token, the separator which can be "|", "&" or ";", the standard input and standard output file names. It finally fills in the argv array with the tokens related to the command excluding I/O redirects.
To find the first and last index of tokens it scans each token looking for a separator. If a separator is found the previous token becomes the last index. To find the I/O redirect files it similarly scans each token until it finds a ">" or "<" identifier. Once found the next token becomes the stdin/stdout file.

To pipe between processes the program first counts all pipes within the string and creates a pipe array with the size two times the count. Once the pipes are created it assigns the input and output accordingly using the following function:

```
        int m = 0;
        for (int i=0; i<num; i++) {
            int in = -1;
            int out = -1;
            if (i != 0) { //input
                if (strcmp(cmds[i-1].sep, pipeSep) == 0) {
                    in = pipefds[m]; // 0
                    m+=2;
                }
            }

            if (strcmp(cmds[i].sep, pipeSep) == 0) { //output
                out = pipefds[m+1]; // 1
            }

            status = execute_command(in, out, &cmds[i]);
        }
```

Once assigned the correct pipes it called the execute_command() function. This function first checks to see if there is a built-in command. If there is it executes it using a function pointer:

```
    for (int i=0; i<getBuiltinCount(); i++) {
        if (strcmp(cmd->argv[0], builtin_cmd[i]) == 0) {
            return (*exec_builtin[i])(cmd);
        }
    }
```

If there is no built-in function it will fork the current environment. The child process which checks for I/O redirects and dups the standard input file descriptor and standard output file descriptor, respectively. It will then dup any pipes to the std input and std output. Once complete it will execute the command using the exevp call:

```
execvp(cmd->argv[0], cmd->argv);
```

The parent process will wait for the program to finish if it is not a background process and close the input and output pipes. It will then check for a signal for SIGCHLD which calls the killChild() function. This function kills the child so that zombie process' are reclaimed.

Overall the solution is pretty strong at breaking up a string into its commands however it still contains limitations. It cannot completely handle input and output redirection with pipes which is due to the duplication of file descriptors. It also has to contain a space into between each eliminator which could be improved so it's not needed.

Test evidence

#1 Compilation: program compiled with -Wall flag without any warnings or errors

```
ubuntu@ubuntu-MS-7978:~/Desktop/Assignment 2$ make clean
rm *.o
ubuntu@ubuntu-MS-7978:~/Desktop/Assignment 2$ make
gcc -c -Wall Unix_Shell.c
gcc -c -Wall token.c
gcc -c -Wall command.c
cc    -c -o builtin.o builtin.c
gcc Unix_Shell.o token.o command.o builtin.o -o Unix_Shell
```

#2 Basics: program completed all processes except for "./show a b c". However, when compared to the Ubuntu shell it also cannot display the file so it is unsure if this is intended behaviour or not.

```
ubuntu@ubuntu-MS-7978:~/Desktop/Assignment 2$ Unix_Shell
% cd tests
% ls
a     abcxy      abcxyz.b    bigbig.txt  markingguide.doc
ab    abcxyz     abcxyz.bb   big.txt     show
abc   abcxyz.a   abcxyz.c    foo         show.c
abcx  abcxyz.aa  abcxyz.ccc  m           Test-Cases.txt
% ps
  PID TTY          TIME CMD
 2724 pts/0    00:00:00 bash
 3479 pts/0    00:00:00 Unix_Shell
 3481 pts/0    00:00:00 ps
% ./show a b c
: No such filehow: cannot open /t[illegible]u[illegible]
./show: 1: ./show: [illegible]
                    P[illegible]: not found
./show: 1: ./show: $[illegible]h[illegible]% [illegible] not found
./show: 1: ./show: : not found
./show: 1: ./show: [illegible]_dyld_make_delayed_module_initializer_calls__dyld_mod
_term_funcsCommand: not found
./show: 2: ./show: Command: not found
./show: 3: ./show: Syntax error: "(" unexpected
%
```

Ubuntu shell version

```
ubuntu@ubuntu-MS-7978:~/Desktop/Assignment 2/tests$ ./show a b c
bash: ./show: cannot execute binary file: Exec format error
```

#3 Built-in commands: all commands executed correctly

```
ubuntu@ubuntu-MS-7978:~/Desktop/Assignment 2$ Unix_Shell
% prompt myshell
myshell cd /tmp
myshell pwd
/tmp
myshell cd
myshell pwd
/home/ubuntu
myshell cd ..
myshell pwd
/home
myshell exit
ubuntu@ubuntu-MS-7978:~/Desktop/Assignment 2$
```

#4 Long command: all commands executed however the "./show" command did not which is also not able to be completed in the Ubuntu terminal, so it is unclear if this is intended behaviour.

```
ubuntu@ubuntu-MS-7978:~/Desktop/Assignment 2$ Unix_Shell
% cd tests
% ls -l -t a b c
ls: cannot access 'b': No such file or directory
ls: cannot access 'c': No such file or directory
-rwxr-xr-x 1 ubuntu ubuntu 0 May 19  2008 a
% show a bb ccc dddd 1 22 333 4444 555555
: No such file cannot open /t◆◆[illegible]◆◆[illegible]◆◆[illegible]◆u◆◆
show: 1: show: ◆◆◆◆[illegible]
                    P[illegible]: not found
show: 1: show: $[illegible]◆[illegible]◆[illegible]◆[illegible]◆h[illegible]%[illegible] not found
show: 1: show: : not found
show: 1: show: ◆◆[illegible]◆◆[illegible]_dyld_make_delayed_module_initializer_calls__dyld_mod_ter
m_funcsCommand: not found
show: 2: show: Command: not found
show: 3: show: Syntax error: "(" unexpected
% show  a b c d e f g h I j k l m n o p q r s t u v w x y z 1 2 3 4 5 6 7 8 9 10
 11 12 13 14 15 16 17 18 20 abc123xyz
: No such file cannot open /t◆◆[illegible]◆◆[illegible]◆◆[illegible]◆u◆◆
show: 1: show: ◆◆◆◆[illegible]
                    P[illegible]: not found
show: 1: show: $[illegible]◆[illegible]◆[illegible]◆[illegible]◆h[illegible]%[illegible] not found
show: 1: show: : not found
show: 1: show: ◆◆[illegible]◆◆[illegible]_dyld_make_delayed_module_initializer_calls__dyld_mod_ter
m_funcsCommand: not found
show: 2: show: Command: not found
show: 3: show: Syntax error: "(" unexpected
```

Ubuntu shell version:

```
ubuntu@ubuntu-MS-7978:~/Desktop/Assignment 2/tests$ ls -l  -t   a b c
ls: cannot access '-l': No such file or directory
ls: cannot access 'b': No such file or directory
ls: cannot access 'c': No such file or directory
a
ubuntu@ubuntu-MS-7978:~/Desktop/Assignment 2/tests$ show a  bb  ccc  dddd 1  22
 333  4444  555555
bash: ./show: cannot execute binary file: Exec format error
ubuntu@ubuntu-MS-7978:~/Desktop/Assignment 2/tests$ show  a b c d e f g h I j k
l m n o p q r s t u v w x y z 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 20 ab
c123xyz
bash: ./show: cannot execute binary file: Exec format error
```

#5 Wildcards: all wildcard command executed correctly.

```
ubuntu@ubuntu-MS-7978:~/Desktop/Assignment 2$ Unix_Shell
% cd tests
% ls -l *.c
-rwxr-xr-x 1 ubuntu ubuntu   0 May 19  2008 abcxyz.c
-rwxr-xr-x 1 ubuntu ubuntu 252 May 19  2008 show.c
% ls -l a*.c
-rwxr-xr-x 1 ubuntu ubuntu 0 May 19  2008 abcxyz.c
% ls -l abc.?
ls: cannot access 'abc.?': No such file or directory
% ls -l abc*.?
-rwxr-xr-x 1 ubuntu ubuntu 0 May 19  2008 abcxyz.a
-rwxr-xr-x 1 ubuntu ubuntu 0 May 19  2008 abcxyz.b
-rwxr-xr-x 1 ubuntu ubuntu 0 May 19  2008 abcxyz.c
% ls -l *
-rwxr-xr-x 1 ubuntu ubuntu        0 May 19  2008 a
-rwxr-xr-x 1 ubuntu ubuntu        0 May 19  2008 ab
-rwxr-xr-x 1 ubuntu ubuntu        0 May 19  2008 abc
-rwxr-xr-x 1 ubuntu ubuntu        0 May 19  2008 abcx
-rwxr-xr-x 1 ubuntu ubuntu        0 May 19  2008 abcxy
-rwxr-xr-x 1 ubuntu ubuntu        0 May 19  2008 abcxyz
-rwxr-xr-x 1 ubuntu ubuntu        0 May 19  2008 abcxyz.a
-rwxr-xr-x 1 ubuntu ubuntu        0 May 19  2008 abcxyz.aa
-rwxr-xr-x 1 ubuntu ubuntu        0 May 19  2008 abcxyz.b
-rwxr-xr-x 1 ubuntu ubuntu        0 May 19  2008 abcxyz.bb
-rwxr-xr-x 1 ubuntu ubuntu        0 May 19  2008 abcxyz.c
-rwxr-xr-x 1 ubuntu ubuntu        0 May 19  2008 abcxyz.ccc
-rw-r--r-- 1 ubuntu ubuntu 13534035 Nov  7  2008 bigbig.txt
-rw-r--r-- 1 ubuntu ubuntu  4511345 Nov  7  2008 big.txt
-rwxr-xr-x 1 ubuntu ubuntu       63 May 19  2008 foo
-rwxr-xr-x 1 ubuntu ubuntu   148992 Nov  7  2008 m
-rwxr-xr-x 1 ubuntu ubuntu   150016 Nov  7  2008 markingguide.doc
-rwxr-xr-x 1 ubuntu ubuntu    13344 May 19  2008 show
-rwxr-xr-x 1 ubuntu ubuntu      252 May 19  2008 show.c
-rwxr-xr-x 1 ubuntu ubuntu     1313 May 19  2008 Test-Cases.txt
```

#6 Sequential execution ';': all commands were executed sequentially.

```
ubuntu@ubuntu-MS-7978:~/Desktop/Assignment 2$ Unix_Shell
% sleep 10 ; echo hello
hello
% sleep 10 ; echo hello1 ; sleep 10 ; echo hello2
hello1
hello2
%
```

#7 Concurrent execution: all commands executed correctly

```
ubuntu@ubuntu-MS-7978:~/Desktop/Assignment 2$ Unix_Shell
% echo hello & echo world
hello
world
% sleep 10 & echo hello
hello
% ps & ls
builtin.c  builtin.o  command.h  makefile  token.c  token.o     Unix_Shell.c
builtin.h  command.c  command.o  tests     token.h  Unix_Shell  Unix_Shell.o
  PID TTY          TIME CMD
 2724 pts/0    00:00:00 bash
 4110 pts/0    00:00:10 Unix_Shell
 4124 pts/0    00:00:00 ps
% echo ps-command & ps & echo ls-command & ls -l
ps-command
ls-command
total 84
-rw-rw-r-- 1 ubuntu ubuntu  1091 Nov  4 16:49 builtin.c
-rw-rw-r-- 1 ubuntu ubuntu    98 Nov  1 16:26 builtin.h
-rw-r--r-- 1 ubuntu ubuntu  3312 Nov  4 16:49 builtin.o
-rw-rw-r-- 1 ubuntu ubuntu  4437 Nov  4 12:57 command.c
-rw-rw-r-- 1 ubuntu ubuntu  1111 Nov  2 19:23 command.h
-rw-r--r-- 1 ubuntu ubuntu  4848 Nov  4 16:48 command.o
-rw-rw-r-- 1 ubuntu ubuntu   372 Nov  4 16:32 makefile
drwxr-xr-x 2 ubuntu ubuntu  4096 Nov  4 12:57 tests
-rw-rw-r-- 1 ubuntu ubuntu   404 Nov  4 12:52 token.c
-rw-rw-r-- 1 ubuntu ubuntu   143 Nov  4 12:46 token.h
-rw-r--r-- 1 ubuntu ubuntu  1728 Nov  4 16:48 token.o
-rwxr-xr-x 1 ubuntu ubuntu 18672 Nov  4 16:49 Unix_Shell
-rw-rw-r-- 1 ubuntu ubuntu  3979 Nov  4 16:35 Unix_Shell.c
-rw-r--r-- 1 ubuntu ubuntu  6160 Nov  4 16:48 Unix_Shell.o
  PID TTY          TIME CMD
 2724 pts/0    00:00:00 bash
 4110 pts/0    00:00:10 Unix_Shell
 4127 pts/0    00:00:00 ps
% sleep 10 &
```

#8 Standard input redirection '<': input redirected worked correctly except for the last command. It showed an error however the command still completed.

```
ubuntu@ubuntu-MS-7978:~/Desktop/Assignment 2$ Unix_Shell
% cd tests
% ls -l > junk
% cat foo > junk2
cat: junk2: input file is output file
%
```

#9 Standard output redirection '>': all command worked as intended.

```
ubuntu@ubuntu-MS-7978:~/Desktop/Assignment 2$ Unix_Shell
% cd tests
% cat < foo
this is file foo
file foo - line 2
line 3
line 4
the last line
% grep line < foo
file foo - line 2
line 3
line 4
the last line
%
```

Open junk2 ~/Desktop/Assignment 2/tests Save

```
this is file foo
file foo - line 2
line 3
line 4
the last line
```

```
Open ▾   junk
         ~/Desktop/Assignment 2/tests   Save
-rwxr--r-- 1 ubuntu ubuntu 0 Nov  4 17:02 junk
```

#10 Simple shell pipeline: all pipeline commands worked as intended.

```
ubuntu@ubuntu-MS-7978:~/Desktop/Assignment 2$ Unix_Shell
% cd tests
% cat foo | cat
this is file foo
file foo - line 2
line 3
line 4
the last line
% cat foo | grep line
file foo - line 2
line 3
line 4
the last line
% cat foo | sort
file foo - line 2
line 3
line 4
the last line
this is file foo
% cat foo | sort -r
this is file foo
the last line
line 4
line 3
file foo - line 2
%
```

#11 Long shell pipeline: all commands executed correctly except for the line with output redirect. The output was not redirected to the file junk.

```
ubuntu@ubuntu-MS-7978:~/Desktop/Assignment 2$ Unix_Shell
% cd tests
% cat foo | sort | sort -r | grep line
the last line
line 4
line 3
file foo - line 2
% cat | cat | cat | cat | cat | cat | cat | cat | cat | cat | cat
hi
yes
wd
hi
yes
wd
% cat | cat | cat | cat | cat | cat | cat | cat | cat | cat | cat > junk
this is a sentence
cat: junk: input file is output file
% cat | cat | cat | cat | cat | cat | cat | cat | cat | cat | grep line
this is a line
this is not
this is a line
%
```

#12 combinations: all commands worked as intended. It states that ls -lt is not a command which is verified by the Ubuntu shell.

```
ubuntu@ubuntu-MS-7978:~/Desktop/Assignment 2$ Unix_Shell
% ls -l   > junk  ;  cat < junk ;  /bin/ls  -lt  /dev/tty*  |  grep tty  |  sort
  |  head  > junk2  &  sleep 10  ;  cat  < junk2
-rwxr--r-- 1 ubuntu ubuntu 47 Nov  4 17:13 junk
% ls -lt | cat > junk ;  ps | sort  &  echo ps-output ; sleep 10 & echo wait-for
-10seconds ; cat junk | cat | grep a | sort -r
ls: cannot access '-lt': No such file or directory
cat: junk: input file is output file
ps-output
 2724 pts/0    00:00:00 bash
 4770 pts/0    00:00:09 Unix_Shell
 4897 pts/0    00:00:00 ps
  PID TTY          TIME CMD
wait-for-10seconds
%
```

Ubuntu shell version:

```
ubuntu@ubuntu-MS-7978:~/Desktop/Assignment 2/tests$ ls -lt | cat > junk ;  ps |
sort  &  echo ps-output ; sleep 10 & echo wait-for-10seconds ; cat junk | cat |
grep a | sort -r
ls: cannot access '-lt': No such file or directory
[1] 4914
ps-output
[2] 4915
wait-for-10seconds
ubuntu@ubuntu-MS-7978:~/Desktop/Assignment 2/tests$  3442 pts/1    00:00:00 bash
 4913 pts/1    00:00:00 ps
 4914 pts/1    00:00:00 sort
 4915 pts/1    00:00:00 sleep
  PID TTY          TIME CMD
```

```
Open ▾   junk  ~/Desktop/Assignment 2/tests   Save
total 17956
-rwxr-xr-x 1 ubuntu ubuntu        0 May 19  2008 a
-rwxr-xr-x 1 ubuntu ubuntu        0 May 19  2008 ab
-rwxr-xr-x 1 ubuntu ubuntu        0 May 19  2008 abc
-rwxr-xr-x 1 ubuntu ubuntu        0 May 19  2008 abcx
-rwxr-xr-x 1 ubuntu ubuntu        0 May 19  2008 abcxy
-rwxr-xr-x 1 ubuntu ubuntu        0 May 19  2008 abcxyz
-rwxr-xr-x 1 ubuntu ubuntu        0 May 19  2008 abcxyz.a
-rwxr-xr-x 1 ubuntu ubuntu        0 May 19  2008 abcxyz.aa
-rwxr-xr-x 1 ubuntu ubuntu        0 May 19  2008 abcxyz.b
-rwxr-xr-x 1 ubuntu ubuntu        0 May 19  2008 abcxyz.bb
-rwxr-xr-x 1 ubuntu ubuntu        0 May 19  2008 abcxyz.c
-rwxr-xr-x 1 ubuntu ubuntu        0 May 19  2008 abcxyz.ccc
-rw-r--r-- 1 ubuntu ubuntu 13534035 Nov  7  2008 bigbig.txt
-rw-r--r-- 1 ubuntu ubuntu  4511345 Nov  7  2008 big.txt
-rwxr-xr-x 1 ubuntu ubuntu       63 May 19  2008 foo
-rwxr--r-- 1 ubuntu ubuntu        0 Nov  4 17:13 junk
-rwxr--r-- 1 ubuntu ubuntu       63 Nov  4 17:02 junk2
-rwxr-xr-x 1 ubuntu ubuntu   148992 Nov  7  2008 m
-rwxr-xr-x 1 ubuntu ubuntu   150016 Nov  7  2008 markingguide.doc
-rwxr-xr-x 1 ubuntu ubuntu    13344 May 19  2008 show
-rwxr-xr-x 1 ubuntu ubuntu      252 May 19  2008 show.c
-rwxr-xr-x 1 ubuntu ubuntu     1313 May 19  2008 Test-Cases.txt
```

```
Open ▾   junk2  ~/Desktop/Assignment 2/tests   Save
crw------- 1 root   root    5,  3 Nov  4 16:26 /dev/ttyprintk
crw-rw---- 1 root   dialout 4, 64 Nov  4 16:26 /dev/ttyS0
crw-rw---- 1 root   dialout 4, 65 Nov  4 16:26 /dev/ttyS1
crw-rw---- 1 root   dialout 4, 66 Nov  4 16:26 /dev/ttyS2
crw-rw---- 1 root   dialout 4, 67 Nov  4 16:26 /dev/ttyS3
crw-rw---- 1 root   dialout 4, 68 Nov  4 16:26 /dev/ttyS4
crw-rw---- 1 root   dialout 4, 69 Nov  4 16:26 /dev/ttyS5
crw-rw---- 1 root   dialout 4, 70 Nov  4 16:26 /dev/ttyS6
crw-rw---- 1 root   dialout 4, 71 Nov  4 16:26 /dev/ttyS7
crw-rw---- 1 root   dialout 4, 72 Nov  4 16:26 /dev/ttyS8
```

#13 Ignore Ctrl-C, Ctrl-\ and Ctrl-Z: all required signals were ignored

```
ubuntu@ubuntu-MS-7978:~/Desktop/Assignment 2$ Unix_Shell
% ^C
% ^Z
% ^\
%
```

#14 Claim zombie processes: zombie processes were claimed and checked using the command “ps aux | grep “defunct”

```
ubuntu@ubuntu-MS-7978:~/Desktop/Assignment 2$ Unix_Shell
% sleep 1 &
% sleep 1 &
% sleep 1 &
% sleep 1 &
% sleep 1 &
% sleep 1 &
%

ubuntu@ubuntu-MS-7978:~/Desktop/Assignment 2$ ps aux |grep "defunct"
ubuntu     5187  0.0  0.0  14424  1104 pts/1    S+   17:32   0:00 grep --color=auto defunct
```

#15 Handline slow system calls: slow system calls were handled by the following function.

```
int again = 1;
while (again) {
    again = 0;
    i = getline(&str, &len, stdin);
    str[i-1] = '\0';
    if (str == NULL)
        if (errno == EINTR)
            again = 1;        // signal interruption, read again
}
```

Source code listing

Builtin.c

```
#include "command.h"
#include <stdio.h>
#include <string.h>
#include <unistd.h>
#include <stdlib.h>

char *builtin_cmd[] = {
    "prompt",
    "pwd",
    "cd",
    "exit"
};

int getBuiltinCount() {
    return sizeof(builtin_cmd) / sizeof(char *);
}

///change prompt
int prompt_builtin(Command *cmd)
{
    extern char *prompt;
    strcpy(prompt, cmd->argv[1]);;

    return 0;
}

int pwd_builtin(Command *cmd)
{
```

```
    char cwd[256];
    if (getcwd(cwd, sizeof(cwd)) != NULL) {
                printf("%s\n", cwd);
    } else {
       perror("pwd err");
       return 1;
    }
    return 0;
}

int cd_builtin(Command *cmd)
{
    if (cmd->argv[1] == NULL) {
        if (chdir(getenv("HOME")) != 0) {
            perror("cd directory err");
        }
    }
    else
    {
        if (chdir(cmd->argv[1]) != 0) { //change directory
            perror("cd directory err");
        }
    }
    return 0;
}

int exit_builtin(Command *cmd)
{
    return 1;
}

//builtin pointer array
int (*exec_builtin[]) (Command *cmd) = {
    &prompt_builtin,
    &pwd_builtin,
    &cd_builtin,
    &exit_builtin
};
```

Builtin.h

```
extern char *builtin_cmd[];

int getBuiltinCount();

extern int (*exec_builtin[]) (Command *cmd);
```

Command.c

```c
#include <string.h>
#include <stdlib.h>
#include <stdio.h>
#include <glob.h>
#include "command.h"

void searchRedirection(char *token[], Command *cp)
{
    int i;
    for (i=cp->first; i<=cp->last; ++i) {
        if (strcmp(token[i], "<") == 0) {   // standard input redirecti
on
            cp->stdin_file = token[i+1];
            ++i;
        } else if (strcmp(token[i], ">") == 0) { // standard output red
irection
            cp->stdout_file = token[i+1];
            ++i;
        }
    }
}

int separator(char *token)
{
    int i=0;
    char *commandSeparators[] = {pipeSep, conSep, seqSep, NULL};

    while (commandSeparators[i] != NULL) {
        if (strcmp(commandSeparators[i], token) == 0) {
            return 1;
        }
        ++i;
    }

    return 0;
}

void fillCommandStructure(Command *cp, int first, int last, char *sep)
{
     cp->first = first;
     cp->last = last - 1;
     cp->sep = sep;
}

void buildCommandArgumentArray(char *token[], Command *cp)
{
    glob_t globResult;
```

```
    int n = 0;
    for (int t=cp->first; t<=cp->last; ++t ) {
        if (strcmp(token[t], ">") != 0 && strcmp(token[t], "<") != 0) {
            glob(token[t], GLOB_NOCHECK, 0, &globResult);
            n += globResult.gl_pathc;
            globfree(&globResult);
        }
    }

    n += 1; //null terminated

     cp->argv = (char **) realloc(cp->argv, sizeof(char *) * n);
     if (cp->argv == NULL) {
         perror("realloc");
         exit(1);
     }

     int i;
     int k = 0;
     for (i=cp->first; i<= cp->last; ++i ) {
         if (strcmp(token[i], ">") != 0 && strcmp(token[i], "<") != 0)
{
             for (int n=0; n<globResult.gl_pathc; n++) {
                 glob(token[i], GLOB_NOCHECK, 0, &globResult);
                 cp-
>argv[k] = malloc(sizeof(char) * (strlen(globResult.gl_pathv[n])+1));
                 strcpy(cp->argv[k], globResult.gl_pathv[n]);
                 globfree(&globResult);
                 ++k;
             }
         }
     }

     cp->argv[k] = NULL;
}

int separateCommands(char *token[], Command command[])
{
     int i;
     int nTokens;

     i = 0;
     while (token[i] != NULL) ++i;
     nTokens = i;

     if (nTokens == 0)
          return 0;
```

```
     if (separator(token[0]))
          return -3;

     if (!separator(token[nTokens-1])) {
        token[nTokens] = seqSep;
        ++nTokens;
     }

     int first=0;   // points to the first tokens of a command
     int last;      // points to the last  tokens of a command
     char *sep;     // command separator at the end of a command
     int c = 0;         // command index
     for (i=0; i<nTokens; ++i) {
         last = i;
         if (separator(token[i])) {
             sep = token[i];
             if (first==last)  // two consecutive separators
                 return -2;
             fillCommandStructure(&(command[c]), first, last, sep);
             ++c;
             first = i+1;
         }
     }

     // check the last token of the last command
     if (strcmp(token[last], pipeSep) == 0) { // last token is pipe sep
arator
          return -4;
     }

     int nCommands = c;

     for (i=0; i<nCommands; ++i) {
         searchRedirection(token, &(command[i]));
         buildCommandArgumentArray(token, &(command[i]));
     }

     return nCommands;
}
```

Command.h

```
#define MAX_NUM_COMMANDS  1000

// command separators
#define pipeSep  "|"
#define conSep   "&"
#define seqSep   ";"
```

```c
struct CommandStruct {
   int first;      // index to the first token in the array "token" of
the command
   int last;       // index to the first token in the array "token" of
the command
   char *sep;      // the command separator that follows the command,
must be one of "|", "&", and ";"
   char **argv;
   char *stdin_file;
   char *stdout_file;
};

typedef struct CommandStruct Command;  // command type

int separateCommands(char *token[], Command command[]);
```

Token.c

```c
#include <string.h>
#include "token.h"

int tokenise (char line[], char *token[])
{
      char *tk;
      int i=0;

      tk = strtok(line, tokenSeparators);
      token[i] = tk;

      for (i=1; tk != NULL; i++) {
          if (i>=MAX_NUM_TOKENS) {
              i = -1;
              break;
          }

          tk = strtok(NULL, tokenSeparators);
          token[i] = tk;
      }

      return i;
}
```

Token.h

```c
#define MAX_NUM_TOKENS  1000
#define tokenSeparators " \t\n"    // characters that separate tokens

int tokenise (char line[], char *token[]);
```

Unix_Shell.c

```c
#include "token.h"
#include "command.h"
#include "builtin.h"
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <sys/wait.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <errno.h>

void killChild() {
    int count = 1;
    pid_t pid;
    int status;

    while (count) {
        pid = waitpid(-1, &status, WNOHANG);
        if (pid < 0)
            count = 0;
    }
}

int execute_command(int input, int output, Command *cmd) {
    if (cmd->argv == NULL) {
        printf(" NO ARGS");
        return 0;
    }

    for (int i=0; i<getBuiltinCount(); i++) {
        if (strcmp(cmd->argv[0], builtin_cmd[i]) == 0) {
            return (*exec_builtin[i])(cmd);
        }
    }

    pid_t pid;
    int status;

    if ((pid = fork()) < 0) {
        perror("fork");
        exit(1);
    }
    else if (pid == 0) {
        if (cmd->stdin_file != NULL) { //redirect input
            int fd0 = open(cmd->stdin_file, O_RDONLY, 0);
```

```
            dup2(fd0, STDIN_FILENO);
            close(fd0);
        }

        if (cmd->stdout_file != NULL) { //redirect ouput
            int fd1 = open(cmd->stdout_file, O_WRONLY|O_CREAT, 0766);
            dup2(fd1, STDOUT_FILENO);
            close(fd1);
        }

        if (input != -1)
            dup2 (input, 0);

        if (output != -1)
            dup2 (output, 1);


        execvp(cmd->argv[0], cmd->argv);


        exit(1);
    } else {
        if (strcmp(cmd->sep, conSep) != 0) {
            wait(&status);
        }

        if (input != -1)
            close (input);

        if (output != -1)
            close (output);

        signal(SIGCHLD, killChild);
        return 0;
    }
}

char *prompt;

int main(void)
{
    sigset_t sigs;

    sigemptyset(&sigs);
    sigaddset(&sigs, SIGINT);
    sigaddset(&sigs, SIGQUIT);
    sigaddset(&sigs, SIGTSTP);

    sigprocmask(SIG_SETMASK, &sigs, NULL);
```

```
    if (!(prompt = malloc(256 * sizeof(char))))
        return 1;
    strcpy(prompt, "%");

    char *str;
    size_t len = 256;
    size_t i = 0;
    int status;
    do {
       printf("\033[0;33m");
       printf("%s ", prompt);
       printf("\033[0m");

       int again = 1;
       while (again) {
            again = 0;
            i = getline(&str, &len, stdin);
            str[i-1] = '\0';
            if (str == NULL)
                if (errno == EINTR)
                     again = 1;        // signal interruption, read aga
in
       }

        char **token = calloc(1000, sizeof(char*));
        tokenise(str, token);

        Command cmds[MAX_NUM_COMMANDS];
        for (int i=0; i<MAX_NUM_COMMANDS; i++)
        {
            cmds[i].first = 0;
            cmds[i].last = 0;
            cmds[i].sep = NULL;
            cmds[i].argv = NULL;
            cmds[i].stdin_file = NULL;
            cmds[i].stdout_file = NULL;
        }

        int num = separateCommands(token, cmds);

        //get pipe count
        int pipeCount = 0;
        for (int x=0; x<num; x++) {
            if (strcmp(cmds[x].sep, pipeSep) == 0) {
                pipeCount += 1;
            }
        }
```

```
        //initialise pipes
        int pipefds[2*pipeCount];
        for(int i = 0; i < pipeCount; i++ ){
            if( pipe(pipefds + i*2) < 0 ){
            }
        }

        int m = 0;
        for (int i=0; i<num; i++) {
            int in = -1;
            int out = -1;
            if (i != 0) { //input
                if (strcmp(cmds[i-1].sep, pipeSep) == 0) {
                    in = pipefds[m]; // 0
                    m+=2;
                }
            }

            if (strcmp(cmds[i].sep, pipeSep) == 0) { //output
                out = pipefds[m+1]; // 1
            }

            status = execute_command(in, out, &cmds[i]);
        }
    } while (!status);

    return(0);
}
```