CS450 Operating System

Binghan Geng
A20482350
bgeng1@hawk.iit.edu

Programming Assignment 1

Introduction

Implement the sequence command execution operator “;” and the parallel execution operator “&” and more

Build & Run Instructions

```
binghangeng:CS450-Assignment-1_Binghan Geng_A20482350 binghangeng$ gcc assignment1.c   1
binghangeng:CS450-Assignment-1_Binghan Geng_A20482350 binghangeng$ ./a.out assignment1.c   2
$CS450>
```

Changes

1. After your shell has started, it will give a prompt to the user. You should use “$CS450” for the prompt.

```c
// Obtain input, need to ensure that buf is large enough, and the structure may be out of bounds
int getcmd(char *buf, int nbuf)
{
  /**
   * 1) After your shell has started, it will give a prompt to the user. You should use
    "$CS450" for the prompt.
   */
    if (isatty(fileno(stdin))) {
        fprintf(stdout, PREFIX);
    }

    memset(buf, 0, nbuf);
    fgets(buf, nbuf, stdin);
    if (buf[0] == 0) {// EOF
        return -1;
    }
```

$CS450

2.After getting the command line string, fork a child process, parse the string from left to right, the parent process waits for the child process to complete the parsing and execution, and then continue to execute the shell main process to parse the string to determine whether there is; & use these two to Split the string, after parsing all the commands, store them in the global g_exe_cmd.

```
e.g. "echo AA;echo BB & echo CC"

It will create 6 commands:

1. echo AA
2. ;
3. echo BB
4. &
5. echo CC
6. ;
```

It mainly completed by function "parseexec()":

```c
struct cmd *parseexec(char **ps, char *es) {
  char *q, *eq;
  int tok;
  struct execcmd *cmd;
  int prev_spilte = 0;

  cmd = &g_exe_cmd[cmd_index++];

  while (!peek(ps, es, "|")) {
    if ((tok = gettoken(ps, es, &q, &eq)) == 0) {
      SHELL_LOG("parse cmd end\n");
      break;
    }

    if (tok == ';' || tok == '&') {
      SHELL_LOG("get new cmd %d\n\n", cmd_index);
      cmd = &g_exe_cmd[cmd_index++];
      prev_spilte = 1;
      cmd->type = tok;
    } else {
      if (prev_spilte) {
        SHELL_LOG("prev_spilte, get new cmd %d\n\n", cmd_index);
        cmd = &g_exe_cmd[cmd_index++];
        prev_spilte = 0;
      }
    }
    cmd->argv[cmd->argc] = mkcopy(q, eq);
    SHELL_LOG("argv[%d]:%s\n", cmd->argc, cmd->argv[cmd->argc]);
    cmd->argc++;

    if (cmd->argc >= MAXARGS) {
      fprintf(stderr, "too many args\n");
      exit(-1);
    }

  }

  return (struct cmd *) &g_exe_cmd[0];
}
```

3. After parsing the command string, the child process starts to execute all the commands in a loop

3.1 Traverse all the commands under g_exe_cmd, record the start and end positions of the traverse command, for example, when the start and end are equal to 0.

3.2 If the type of the command is &, and a normal command (such as echo A), then put end++.

3.3 If the type of command is ;, then you need to execute the command, because the logic of ; is sequence execution.

```
#1 Traverse all the commands from begin to end at this time, remove the
middle & command

#2 Fork child process and executes all commands from begin to end to
remove & in parallel

#3 The parent process waits for all child processes to complete
```

3.4 Determine whether the execution of all commands is completed. If begin is not equal to end when the execution is completed, then you need to perform action 3

3.5 end executes all commands, exit the loop of executing commands, exit

It mainly completed by function "run_foreach_cmd()" and "run_do_cmds()":

```
void run_foreach_cmd(struct cmd *cmd) {
  int i;
  struct execcmd *ecmd;
  int bindex, eindex;

  if (cmd == NULL) {
    return;
  }

  SHELL_LOG("\nrun_foreach_cmd have %d cmds\n", cmd_index);
  if (cmd_index == 0) {
    return;
  }

  bindex = eindex = 0;
  while (1) {
```

```
    SHELL_LOG("------------------[%d-%d]-----------------\n", bindex,
eindex);

    if (eindex == cmd_index) {
      SHELL_LOG("eindex %d == cmd_index %d, do cmds end, bindex %d\n",
eindex,
                cmd_index, bindex);

      // Command traversal is complete, execute the remaining commands
      if (bindex < eindex) {
        run_do_cmds(bindex, eindex);
      }

      break;
    }

    ecmd = &g_exe_cmd[eindex];
    SHELL_LOG("bindex %d, eindex %d, ecmd argc %d, type %c\n", bindex,
eindex, ecmd->argc,
              ecmd->type);

    SHELL_LOG("curr cmd:");
    for (i = 0; i < ecmd->argc; i++) {
      SHELL_LOG("%s ", ecmd->argv[i]);
    }
    SHELL_LOG("\n");

    if (ecmd->type == ';') { // sequence execute
      // First execute the command between bindex and eindex, and
continue to parse the command
      SHELL_LOG("do [;] begin, bindex %d, eindex %d\n", bindex,
eindex);
      run_do_cmds(bindex, eindex);

      eindex++;
      bindex = eindex;
      SHELL_LOG("do [;] end, eindex to %d, bindex to %d\n", eindex,
bindex);
    } else if (ecmd->type == '&') {// parallel execute, need to
continue to judge the need to parallel several commands
      eindex++;
      SHELL_LOG("do [&] bindex %d, eindex to %d\n", bindex, eindex);
    } else {// not; &, then you need to continue to judge whether there
is any; &
      eindex++;
```

```
      SHELL_LOG("not [;&] bindex %d, eindex to %d\n", bindex, eindex);
    }
  }
}
```

```
void run_do_cmds(int bindex, int eindex) {
  int i, j;
  int tmp_ecmd_index = 0;
  pid_t ecmd_pid[64];
  struct execcmd *g_tmp_ecmd[64];
  struct execcmd *ecmd;

  tmp_ecmd_index = 0;
  memset(g_tmp_ecmd, 0x00, sizeof(struct execcmd *) * 64);
  memset(ecmd_pid, 0x00, sizeof(pid_t) * 64);

  for (i = bindex; i < eindex; i++) {
    ecmd = &g_exe_cmd[i];

    SHELL_LOG("i %d cmd:", i);
    for (j = 0; j < ecmd->argc; j++) {
      SHELL_LOG("%s ", ecmd->argv[j]);
    }
    SHELL_LOG("\n");

    if (ecmd->type != '&') {
      g_tmp_ecmd[tmp_ecmd_index] = ecmd;
      tmp_ecmd_index++;
    }
  }

  // Execute specific commands
  SHELL_LOG("tmp_ecmd_index %d\n", tmp_ecmd_index);
  for (i = 0; i < tmp_ecmd_index; i++) {
    ecmd = g_tmp_ecmd[i];

    SHELL_LOG("execv cmd:");
    for (j = 0; j < ecmd->argc; j++) {
      SHELL_LOG("%s ", ecmd->argv[j]);
    }
    SHELL_LOG("\n");

    ecmd_pid[i] = fork1();
```

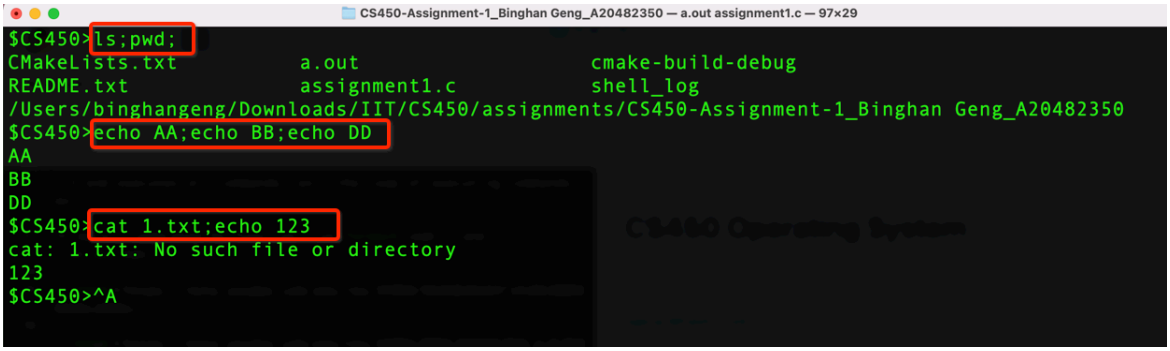
```
    if (ecmd_pid[i] == 0) {
      // The child process executes specific commands
      if (execv(ecmd->argv[0], ecmd->argv) == -1) {
        char mypath[20] = "/bin/";
        strcat(mypath, ecmd->argv[0]);
        if (execv(mypath, ecmd->argv) == -1) {
          strcpy(mypath, "/usr/bin/");
          strcat(mypath, ecmd->argv[0]);
          if (execv(mypath, ecmd->argv) == -1) {
            fprintf(stderr, "Command %s can't find\n", ecmd->argv[0]);
            _exit(0);
          }
        }
      }
    }
  }

  // The parent process waits for all child processes to return
  for (i = 0; i < tmp_ecmd_index; i++) {
    waitpid((pid_t) ecmd_pid[i], NULL, 0);
  }

}
```

Test Cases

1. Sequence Operation

test cases:

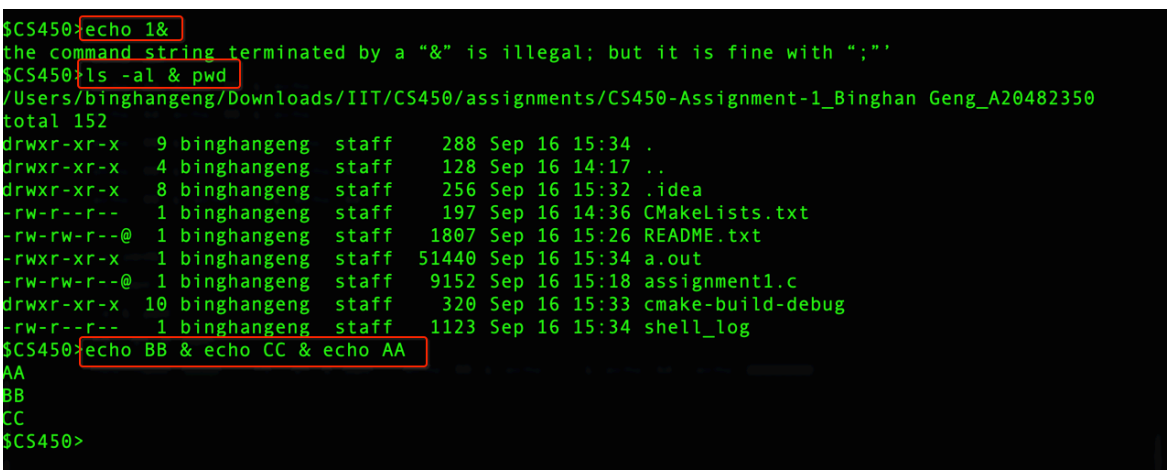
```
1. ls;pwd
2. echo AA;echo BB;echo DD
3. cat 1.txt;echo 123
```



2. Parallel Operation

test cases:

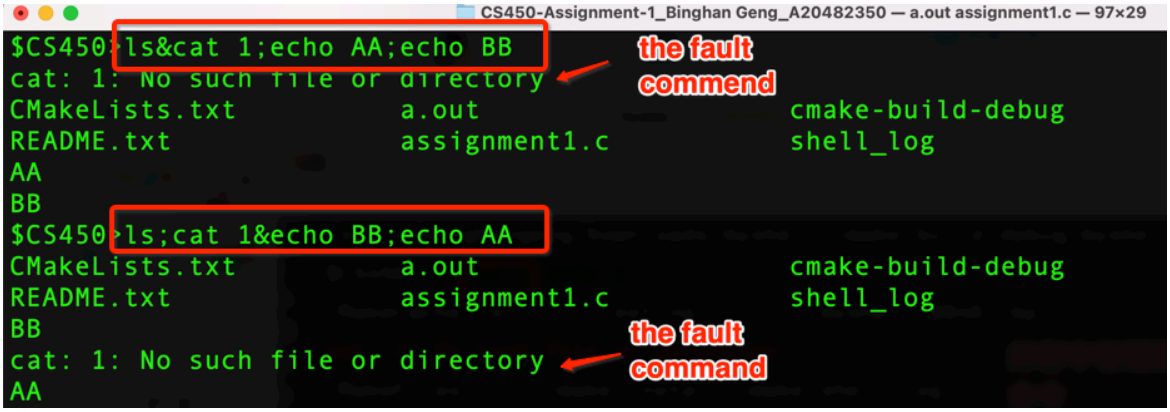
```
1. echo 1&
2. ls -al & pwd
3. echo BB & echo CC & echo AA
```



3. Sequence & Parallel Operation

test cases:

```
1. ls&cat 1;echo AA;echo BB
2. ls;cat 1&echo BB;echo AA
```


CS450-Assignment-1_Binghan Geng_A20482350 — a.out assignment1.c — 97×29
$CS450>ls&cat 1;echo AA;echo BB
cat: 1: No such file or directory
CMakeLists.txt a.out cmake-build-debug
README.txt assignment1.c shell_log
AA
BB
$CS450>ls;cat 1&echo BB;echo AA
CMakeLists.txt a.out cmake-build-debug
README.txt assignment1.c shell_log
BB
cat: 1: No such file or directory
AA
the fault commend
the fault command