# CS450 Operating System

Binghan Geng A20482350

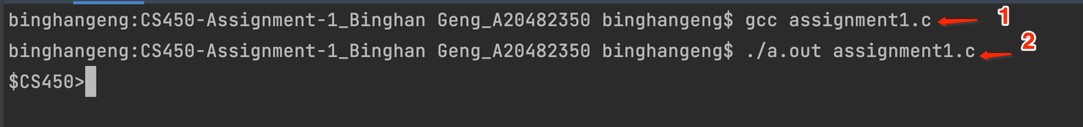
[bgeng1@hawk.iit.edu](mailto:bgeng1@hawk.iit.edu)

# Programming Assignment 1

## Introduction

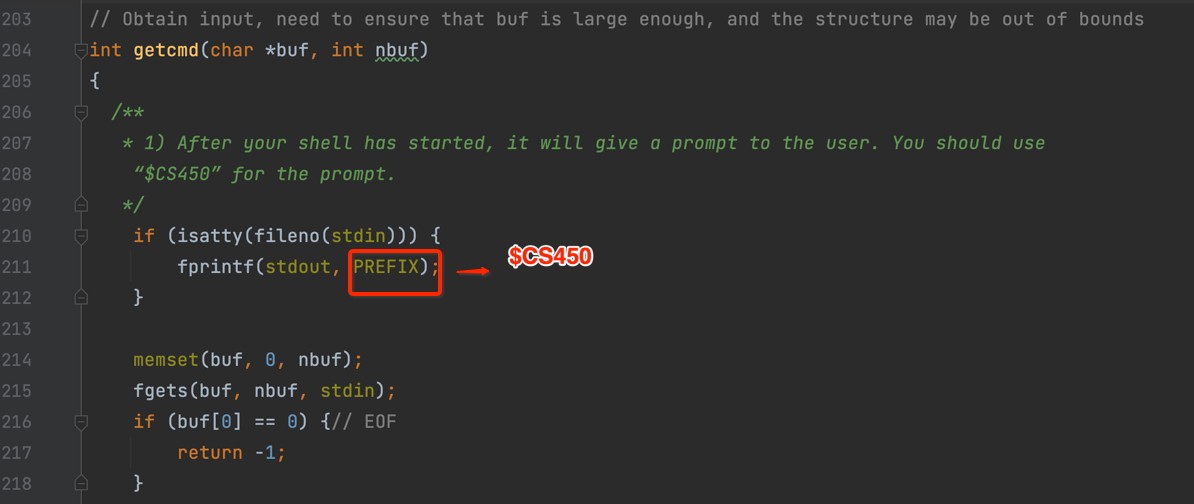
Implement the sequence command execution operator “;” and the parallel execution operator “&” and more

## Build & Run Instructions



Changes

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



1. 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()":

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];

}

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

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

* 1. 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
  2. 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);

}

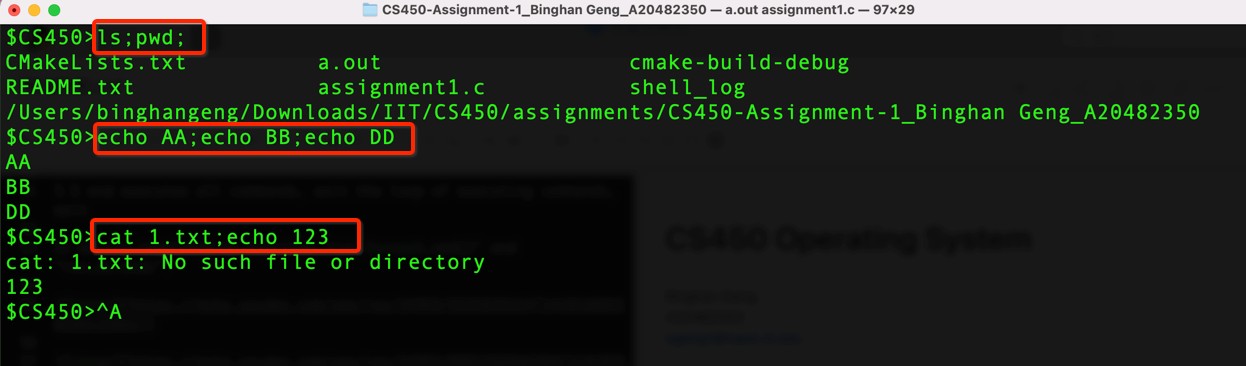
}

## Test Cases

### Sequence Operation

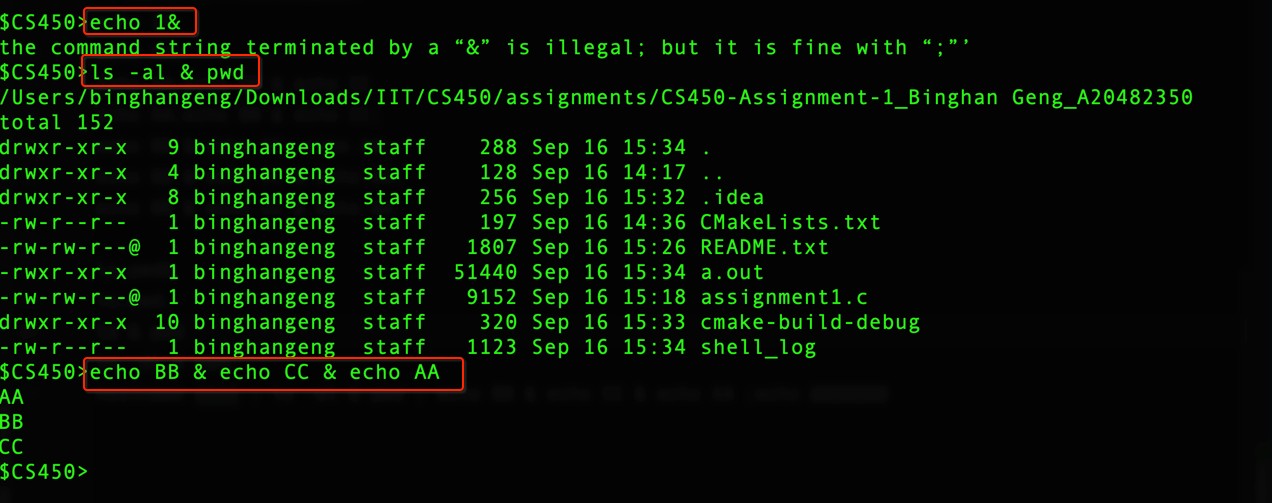
test cases:

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



### Parallel Operation

test cases:



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

### Sequence & Parallel Operation

test cases:

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

