Quash Tutorial

EECS 678 Staff

Quash in a Nutshell

中文理解： https://www.cnblogs.com/wuyuegb2312/p/3399566.html

后台运行（background）：让父进程不等待子进程退出而直接读入用户的下一步操作即可，不执行wait()
foreground：waitpid（）

quash.c

```
main ( … ) {
    while (is_running) {
        CommandHolder *script = parse(&state);

        run_script (script);
    }
}
```

execute.c

```
run_script (CommandHolder* holders) {

    if ( end_condition_reached[1] ) {
        is_running = false;
    }

    for each holder in holders {
        create_process (holder);
    }

    if (holder_contains_forground_jobs) {
        wait_for_all_the_proceses_in_the_job_to_complete;
    } else {
        push_the_job_in_background_job_queue;
    }
}

create_process (CommandHolder holder) {

    Setup_pipes_and_io_redirection_based_on_flags;

    fork_a_child ();

    if (in_child) {
        run_child_command(holder.cmd);
        exit (EXIT_SUCCESS);
    } else {
        run_parent_command(holder.cmd);
    }
}

run_*****_command (Command cmd) {

    switch based on command type {
        case command_type:
            run_command_action (command_arguments);
            break;
        …
        default:
             fprintf (stderr, "Unknown Command\n");
    }
}
```

Essential Data Structures

从右往左看：从小往大；
redirect_in 重定向输入<

command.h

```
struct CommandHolder {
     char* redirect_in,
     char* redirect_out,
     int flags,
     Command cmd;
}
```

command.h

```
union Command {
     SimpleCommand simple;
     GenericCommand generic;
     EchoCommand echo;
     ExportCommand export;
     CDCommand cd;
     KillCommand kill;
     PWDCommand pwd;
     JobsCommand jobs;
     ExitCommand exit;
     EOCCommand eoc;
} Command;
```

Example

command.h

```
struct CDCommand {
     CommandType type;
     char* dir;
} CDCommand;
```

Quash Invocation

Example - 1

```
>> ./quash
[<QUASH_PROMPT>] cd /home/
```

flag：是否运行在background

```
typedef enum CommandType {
EOC = 0, // pseudo-command for marking the end of a script
    GENERIC,
    ECHO,
    EXPORT,
    KILL,
    CD,
    PWD,
    JOBS,
    EXIT
} CommandType;
```

quash.c

```
main ( … ) {
    while (is_running) {
        CommandHolder *script = parse(&state);

        run_script (script);
    }
}

-------------------------------------------------
After 3rd line CommandHolder structure array
pointed to by script looks like the following:

script[0].redirect_in = 0;
script[0].redirect_out = 0;
script[0].flags = 0;
script[0].cmd =>
    script.cmd.type = 5;
    script.cmd.dir = "/home/";
```

Quash Invocation

Example - 2

[<QUASH_PROMPT>] cd /home/ | ls –ll /home/

quash.c

```
main ( … ) {
    while (is_running) {
        CommandHolder *script = parse(&state);

        run_script (script);
    }
}
```

After 3rd line CommandHolder structure array pointed to by script looks like the following:

```
script[0].redirect_in = 0;
script[0].redirect_out = 0;
script[0].flags = 0x10;
script[0].cmd =>
    script.cmd.type = CD;
    script.cmd.dir = "/home/";


script[1].redirect_in = 0;
script[1].redirect_out = 0;
script[1].flags = 0x10;
script[1].cmd =>
    script.cmd.type = GENERIC;
    script.cmd.args = ["ls", "-ll", "/home/"];
```

cmd.args: execute funciton

[<QUASH_PROMPT>] cd /home/ | ls –ll /home/

代表一个job有两个process；

如果需要多个jobs，必须需要打多个commands 最后以&结尾

quash.c

```
main ( … ) {
    while (is_running) {
        CommandHolder *script = parse(&state);

        run_script (script);
    }
}
```

After 3rd line CommandHolder structure array pointed to by script looks like the following:

```
script[0].redirect_in = 0;
script[0].redirect_out = 0;
script[0].flags = 0x10;
script[0].cmd =>
    script.cmd.type = CD;
    script.cmd.dir = "/home/";

script[1].redirect_in = 0;
script[1].redirect_out = 0;
script[1].flags = 0x10;
script[1].cmd =>
    script.cmd.type = GENERIC;
    script.cmd.args = ["ls", "-ll", "/home/"];
```

执行第一个 cd(change directory)

执行第二个 ls(list directory)

提到了需要两个queue 来完成

execute.c

```
run_script (CommandHolder* holders) {

    if ( end_condition_reached[1] ) {
        is_running = false;
    }

    for each holder in holders {
        create_process (holder);
        ----------------------------------
        Iteration-0: Create process for CD
        Iteration-1: Create process for ls
        ----------------------------------
    }

    if (holder_contains_forground_jobs) {
        wait_for_all_the_proceses_in_the_job_to_complete;
        ------------------------------------------------
        NOTE: You need a queue here which is populated
        by create_process () function; to track the
        pids of created processes and wait for them
        to exit
        ------------------------------------------------
    } else {
        push_the_job_in_background_job_queue;
        ------------------------------------------------
        NOTE: Another queue required to accomplish this
        task
        ------------------------------------------------
    }
}

create_process (CommandHolder holder) {

    Setup_pipes_and_io_redirection_based_on_flags;

    pid = fork_a_child ();

    if (pid == 0) {
        run_child_command(holder.cmd);
        exit (EXIT_SUCCESS);
    } else {
        ------------------------------------------------
        NOTE: This is a good place to populate the pid
        queue
        ------------------------------------------------
        run_parent_command(holder.cmd);
    }
}
```

commands: jobs — see all stopped or background processes
fg bring a background process to foreground
bg: restart a stopped background process

Command Handling

background(jobs)and foreground(process)

- Parent Side
  - EXPORT
  - CD
  - KILL

像cd这样的命令实际并非可执行程序，（如果想在自己编写的shell里使用）需要自己来实现为内建命令。那么，对于这种命令，肯定是不能exec()了，需要进行分析和额外处理。而且可以看出，它的执行并不需要建立子进程。

- Child Side
  - GENERIC
  - ECHO
  - PWD
  - JOBS

Quash Milestones

- get_current_directory
- create_process (Step-1)
  just uncomment child and parent run process functions
- lookup_env
- run_pwd
- run_cd
- run_export
- run_echo
- run_generic
- create_process (Step-2)
  Setup pipes to establish IO redirection among children
- run_script (Step-1)
  Implement PID-queue, update create_process to track the pids of children. After returning from process creation and if the job is foreground, pop processes one by one from the queue and wait for each of them to exit
- create_process (Step-3)
  Setup file redirection for child process outputs
- run_script (Step-2)
  Implement background job handling
- run_kill
  Implement signal handling in quash to process kill signal

Debugging

- Don’t write a whole bunch of code and then start debugging
- Make progress in small steps!

Add Comments!

If either of these apply to you:

```
// no comments for you
// it was hard to write
// so it should be hard to read
```

```
//When I wrote this, only God and I understood what I was doing
//Now, God only knows
```

Then I won’t hold it against you if you don’t add comments in your code. ☺