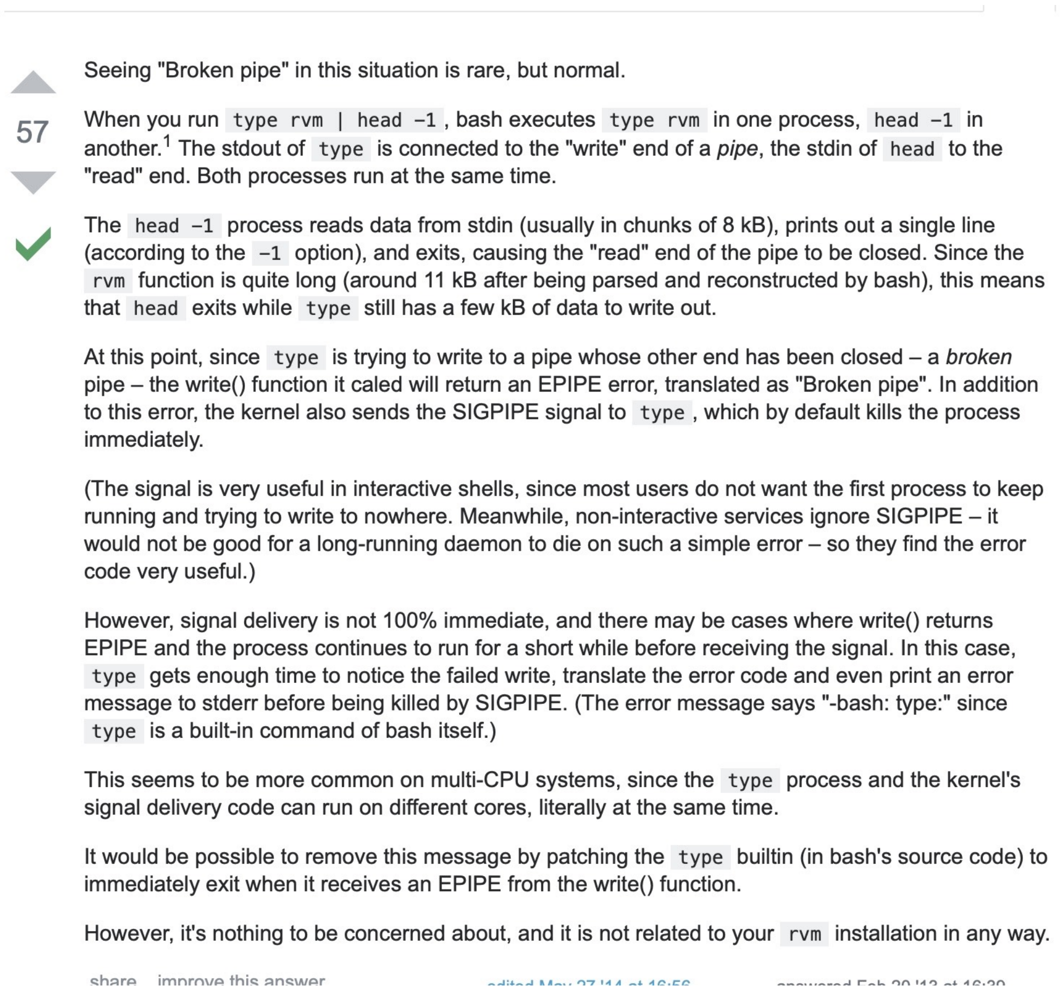
Some useful note here:

1. for "char*" in C, it MUST END WITH '\0'



57

Seeing "Broken pipe" in this situation is rare, but normal.

When you run `type rvm | head -1`, bash executes `type rvm` in one process, `head -1` in another.[1] The stdout of `type` is connected to the "write" end of a *pipe*, the stdin of `head` to the "read" end. Both processes run at the same time.

The `head -1` process reads data from stdin (usually in chunks of 8 kB), prints out a single line (according to the `-1` option), and exits, causing the "read" end of the pipe to be closed. Since the `rvm` function is quite long (around 11 kB after being parsed and reconstructed by bash), this means that `head` exits while `type` still has a few kB of data to write out.

At this point, since `type` is trying to write to a pipe whose other end has been closed – a *broken* pipe – the write() function it caled will return an EPIPE error, translated as "Broken pipe". In addition to this error, the kernel also sends the SIGPIPE signal to `type`, which by default kills the process immediately.

(The signal is very useful in interactive shells, since most users do not want the first process to keep running and trying to write to nowhere. Meanwhile, non-interactive services ignore SIGPIPE – it would not be good for a long-running daemon to die on such a simple error – so they find the error code very useful.)

However, signal delivery is not 100% immediate, and there may be cases where write() returns EPIPE and the process continues to run for a short while before receiving the signal. In this case, `type` gets enough time to notice the failed write, translate the error code and even print an error message to stderr before being killed by SIGPIPE. (The error message says "-bash: type:" since `type` is a built-in command of bash itself.)

This seems to be more common on multi-CPU systems, since the `type` process and the kernel's signal delivery code can run on different cores, literally at the same time.

It would be possible to remove this message by patching the `type` builtin (in bash's source code) to immediately exit when it receives an EPIPE from the write() function.

However, it's nothing to be concerned about, and it is not related to your `rvm` installation in any way.

share improve this answer edited May 27 '14 at 16:56 answered Feb 20 '13 at 16:30

2. background process implementation: just dont wait in mum_execute()

3. when messing up with char*str, like access element str[pos-1], MAKE SURE "pos>=1"

VE482 — Introduction to Operating Systems

Project 1

Manuel — UM-JI (Fall 2019)

Goals of the project

- Write a simple shell
- Run the shell in Linux
- Run the shell in Minix 3

1 A tough life...

As you are comfortably sleeping in your soft bed, your alarm clock loudly rings and wakes you up. Already 11 AM... It would be good to sleep a bit longer but you already hear your mum stepping in the room and widely opening the curtains. As you are blinded by the sudden burst of light you start grumbling, but she is not even paying attention to you and just says: "I told you to sleep instead of playing video games all night long! You've graduated two months ago and you haven't found job yet!" The head under your pillow to protect yourself from the light you simply ignore her and start smiling as you think of last year at the same period when you had to wake up early to attend ve477 lectures. But luckily now you are home and you can sleep, so why not enjoying it? As you start to fall asleep again, she admonishes you "Wake up lazy kid! Time for you to to find a job!"

"But mum, I have already sent two CVs last week? What else do you want me to do?" you try to argue.

"What do I want you to do? Hum...I want you to start doing something with your life! If you don't want to find a job then you'll work here!" she orders you.

As she carries on you simply look at her disconcerted. "Yes, I found some exercises for you: you're going to implement a shell," and she punctuates with an ironic smile.

Totally speechless, you look at her wondering how come she knows what a shell is.

As she approaches from your bed she hands in a couple of pages to you just saying "Lets work begin! And be sure that I'll check on your regularly..."

While you regain your composure you manage to articulate "I'm already very busy playing video games!"

To what she firmly replies "Not anymore you'll be busy implementing a shell!" Then she quits the room, leaving you puzzled and bitter. Why has not Lemonion call you already for an interview, you had the perfect profile? Now you are here lying in your bed with a couple of pages entitled *The mumsh shell*.

2 The `mumsh` shell

The main task of a shell is to wait for some user input, parse it and execute a command requested by the user. It should also provide support for input/output redirection and pipelining from a program into another one.

A shell simply consists of a loop that parses the user input. When waiting the shell displays a prompt, here we want our shell to display "`mumsh $ `".

When a command is input by the user it should be launched in a new process and the shell should block, waiting for the command to end.

A command line is composed of a command followed by some arguments. Arguments are space separated. The shell should exit when the user inputs `exit`. In case of error, such as when a command that does not exists is input, the shell should output an error on the standard error output (e.g. `Error: no such file or directory`).

The shell can be tested by comparing its behaviour to the result of the same commands in the regular Linux shell (e.g. sh, bash, zsh). Note that those commands are only for testing purpose, therefore they are far from being optimal and do not encompass all the features that need to be implemented.

```
mumsh $ echo 123 | grep 2
mumsh $ echo 123 > 1.txt
mumsh $ echo 456 >> 1.txt
mumsh $ cat < 1.txt
mumsh $ cat < 1.txt | sort -R > 2.txt
```

The `mumsh` shell is expected to be running in both Linux and Minix 3.

Hints:

- Useful system calls: `fork()`, `execvp()`, `wait/waitpid()`, `dup2/dup()`, `pipe()` and `close()`.
- A command line is not expected to be longer than 1024 characters.
- The use of the command `system` or of `lex` and `yacc` is prohibited.

3 Mum's Grading policy

Your mother has decided that you should follow a total of thirteen requirements, some with sub-tasks. For each requirement the awarded marks are display in bold into square brackets.

Important notes:

- In case of a final grade larger than 100, the extra marks will be saved for a bonus;
- A 50% penalty will be applied if commands are not launched in a new process;
- Any work that is not pushed onto ve482 git server will be ignored;
- Any task that does not exit cleanly, has memory leaks, or undefined behaviors will receive a −10% deduction;
- The shell should only output normal characters (e.g. no escape characters such as `\033` to handle color);

In the following description "requirement *x*" stands for requirement *x*, including all its sub-tasks, if any. A requirement having dependencies is considered completed if and only if it is completed together with all it's dependencies.

1. Write a working read/parse/execute loop and an `exit` command; **[5]**
2. Handle single commands without arguments (e.g. `ls`); **[5]**
3. Support commands with arguments (e.g. `apt-get update` or `pkgin update`); **[5]**
4. File I/O redirection: **[5+5+5+2]**

4.1. Output redirection by overwriting a file (e.g. `echo 123 > 1.txt`);
4.2. Output redirection by appending to a file (e.g. `echo 465 >> 1.txt`);
4.3. Input redirection (e.g. `cat < 1.txt`);
4.4. Combine 4.1 and 4.2 with 4.3;

5. Support for bash style redirection syntax (e.g. `cat < 1.txt 2.txt > 3.txt 4.txt`); **[8]**
6. Pipes: **[5+5+5+10]**
   6.1. Basic pipe support (e.g. `echo 123 | grep 1`);
   6.2. Run all 'stages' of piped process in parallel. (e.g. `yes ve482 | grep 482`);
   6.3. Extend 6.2 to support requirements 4. and 5. (e.g. `cat < 1.txt 2.txt | grep 1 > 3.txt`);
   6.4. Extend 6.3 to support arbitrarily deep "cascade pipes" (e.g. `echo 123 | grep 1 | grep 1 | grep 1`)

   Note: the sub-processes must be reaped in order to be awarded the marks.
7. Support `CTRL-D` (similar to bash, when there is no/an unfinished command); **[5]**
8. Internal commands: **[5+5+5]**
   8.1. Implement `pwd` as a built-in command;
   8.2. Allow changing working directory using `cd`;
   8.3. Allow `pwd` to be piped or redirected as specified in requirement 4.;
9. Support `CTRL-C`: **[5+3+2+10]**
   9.1. Properly handle `CTRL-C` in the case of requirement 4.;
   9.2. Extend 9.1 to support subtasks 6.1 to 6.3;
   9.3. Extend 9.2 to support requirement 7., especially on an incomplete input;
   9.4. Extend 9.3 to support requirement 6.;
10. Support quotes: **[5+2+3+5]**
    10.1. Handle single and double quotes (e.gm. `echo "de'f' ghi" '123"a"bc' a b c`);
    10.2. Extend 10.1 to support requirement 4. and subtasks 6.1 to 6.3;
    10.3. Extend 10.2 in the case of incomplete quotes (e.g. Input `echo "de`, hit enter and input `cd"`);
    10.4. Extend 10.3 to support requirements 4. and 6., together with subtask 9.3;
11. Wait for the command to be completed when encountering >, <, or |: **[3+2]**
    11.1. Support requirements 3. and 4. together with subtasks 6.1 to 6.3;
    11.2. Extend 11.1 to support requirement 10.;
12. Handle errors for all supported features. **[10]**

    Note: a list of test cases will be published at a later stage. Marks will be awarded based on the number of cases that are correctly handled, i.e. if only if:

    - A precise error message is displayed (e.g. simply saying "error happened!" is not enough);
    - The program continues executing normally after the error is identified and handled;
13. A command ending with an `&` should be run in background. **[10]**

13.1. For any background job, the shell should print out the command line, prepended with the job ID and the process ID (e.g. if the two lines `/bin/ls &` and `/bin/ls | cat &` are input the output could be the two lines `[1] (32757) /bin/ls &` and `[2] (32758) (32759) /bin/ls | cat &`);

13.2. Implement the command `jobs` which prints a list of background tasks together with their running states (e.g. in the previous case output the two lines `[1] done /bin/ls &` and `[2] running /bin/ls | cat &`);

4 Mum's schedule

As your mother wants to closely monitor you and ensure you do not play video games in secret instead of working on her shell she has decided to enforce a tight schedule:

Milestone 1: tasks 1 to 5 must be completed;

Milestone 2: tasks 6 to 9 must be completed;

Final shell: fully functional shell;

It seems your mother has thought of everything, she has even organised to setup an Online Judge to verify that you are respecting the milestones; and she will be pretty tough: **if a task fails to pass the very basic simple milestone tests it will receive a -50% deduction!**

Feeling resigned you decide to follow her advice, so you grab your computer and start working in your bed thinking it makes a very comfortable desk...

JOJ Online Judge

Important reminders regarding the Online Judge (OJ):

- Save all the files, without any folder structure, into a `.tar` archive;
- Strictly stick to the specifications;
- The OJ only checks the correctness of the code not its quality;
- Use the provided `Makefile` or `CMakeLists.txt` files;