ECE482 — Introduction to Operating Systems

Project 1

Manuel — JI (Fall 2024)

Goals of the project

- Get familiar with system calls
- Write a simple shell in C
- Run the shell in Linux

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 ece477 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 the shell should 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
```

Hints to get started:

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

- Mum considers requirements marked with a * as advanced.[1]
- A 50% penalty will be applied if commands are not launched in a new process;
- Any work that is not pushed onto the git server will be ignored;
- Agile development must be used all along the project;
- All the written code must be of good quality;
- 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;
- Prepare a professional README file showing a "build badge" as well as basic installation, running instructions, and the author information. More advanced, but still simple, documentation must be part of a Wiki page. Lack of documentation will lead to a 10% deduction.

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]Advanced requirements should only be completed if you enrolled in ECE4821.

1. Write a working read-parse-execute loop and an `exit` command; **[7.5]**
2. Handle single commands without arguments (e.g. `ls`); **[7.5]**
3. Support commands with arguments (e.g. `apt-get update` or `pkgin update`); **[8]**
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`); **[10]**
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 ece482 | 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: **[6+6+6]**
   - 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+3+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+3]**
   - 11.1. Support requirements 3. and 4. together with subtasks 6.1 to 6.3;
   - 11.2. Extend 11.1 to support requirement 9.;

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+5]**

13.1. For any background job, print out the command line, prepended with the job ID (e.g. if the two lines `/bin/ls &` and `/bin/ls | cat &` are input the output could be the two lines `[1] /bin/ls &` and `[2] /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; To show you she is willing to be flexible she allows you to be reasonably late for the milestones as long the final submission in completed on time. More precisely m1 can be submitted until the deadline of m2, and m2 until the final deadline. Each missed deadline will incur a -30% deduction on the final project grade. Besides for m1 and m2 passing between 25% and 50% of the test cases will lead to -10%, and -20% if less than 25% of the test cases are passed.

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...

A few minutes later she appears at your door and tells you: "By the way, here are some more detailed specs for your shell and some hints on how you can test it!" Before you realise it you see a USB stick falling in front of you. After opening the device you find a file containing the following information. While leaving your room she reminds you: "C only, no C++!"

A First document

This files contains the following information on how you can test your shell.

Continuous Integration

Both code quality and correctness will be tested through Drone.

- Code quality: each time code is pushed, a code quality check will be run and a grade assigned;
- Correctness:
  - JOJ will be triggered on commit messages starting with the keyword `joj`;
  - Check specific tests, e.g. `> git commit -m"joj␣p1m2-1␣3␣5"`
  - When all tasks are ready, check the whole milestone: `> git commit -m"joj␣p1m2"`

Available JOJ test sets: p1m1, p1m2-0, p1m2-1, p1m3-0, p1m3-1. The `0` and `1` tests correspond to ECE4820 and ECE4821, respectively.

A.1 Input

Mum will test your shell with a carefully coded driver, so the input format in the test cases will be taken as the driver's `stdin`.

There are three kinds of commands in the input:

- `WRITELN <line>`: Write `line` to your shell ended with a `\n`.
- `CTRL <C|D>`: Send `SIGINT` with `CTRL C` or `EOF` with `CTRL D`.
- `WAIT <T>`: Wait for `T` ms.

A.2 Output

All of output should be flushed if it wasn't flushed automatically by `\n`. If you don't do so, the order of the output of different processes can be arbitrary, so you will get wrong answer.

The newline character (`\n`), tab character (`\t`) and spaces will be considered as the same in the output.

Mum will redirect your `stdout` and `stderr` to the same file in the driver, so please do not print debug information to `stderr`. The reason why it is done this way it is that a shell should print prompts and error messages to `stderr`, though our project doesn't have this requirement. Some may follow this rule but some will not, so mum has to consider them together.

A.3 Examples

The examples will show input in the left and output in the right. Most of them are from milestone 1 and 2, and the real test cases before. These are published in order to test your shell functionalities rather than sticking to your carefulness on I/O formats.

B Second document

This file provides more detailed information on mum's requirements as well as clearer specification for each task. Milestones and test cases appear into brackets. Each test set features all the previous ones together with a regular and a memory set of tests for the current milestone, e.g. `p1m2-0 2` corresponds to the memory tests for p1m1 and `p1m3-1 5` corresponds to p1m3 regular tests.

B.1 Write a working read-parse-execute loop and an exit command. [m1 1]

The loop time should be reasonable (<10ms), your overall performance will be tested in this part.

Hint. Don't forget to print `exit` when exiting your shell.

Example (Milestone 1 Case 1)

```
WRITELN exit                                mumsh $ exit
```

B.2 Handle single commands without arguments. [m1 2-4]

Hint. There are only three files in the working directory, they are `driver`, `mumsh`, `mumsh_mmeory_check`.

Example (Milestone 1 Case 2)

```
WRITELN ls                                  mumsh $ driver
WRITELN exit                                mumsh
                                            mumsh_memory_check
                                            mumsh $ exit
```

B.3 Support commands with arguments. [m1 5-6]

After seeing some writing programs like `cat`, `ls` themselves (totally wrong in a shell), mum has these examples with package managers now.

Example (Milestone 1 Case 5)

```
WRITELN ls -a                               mumsh $ .
WRITELN exit                                ..
                                            driver
                                            mumsh
                                            mumsh_memory_check
                                            mumsh $ exit
```

B.4 File I/O redirection.

This task is based on Task 2 and 3.

Only one output redirection and one input redirection is allowed in a command. The test cases will strictly follow this rule.

B.4.1 Output redirection by overwriting a file. [m1 7-8]

Example (Milestone 1 Case 7)

```
WRITELN echo 321 > 1.txt
WRITELN echo 123 > 2.txt
WRITELN cat 2.txt
WRITELN cat 1.txt
WRITELN ls
WRITELN exit
```

```
mumsh $ mumsh $ mumsh $ 123
mumsh $ 321
mumsh $ 1.txt
2.txt
driver
mumsh
mumsh_memory_check
mumsh $ exit
```

B.4.2 Output redirection by appending to a file. [m1 9-10]

Example (Milestone 1 Case 9)

```
WRITELN echo 321 >> 1.txt
WRITELN cat 1.txt
WRITELN ls
WRITELN exit
```

```
mumsh $ mumsh $ 321
mumsh $ 1.txt
driver
mumsh
mumsh_memory_check
mumsh $ exit
```

B.4.3 Input redirection. [m1 11-12]

Example (Milestone 1 Case 11)

```
WRITELN echo 123 > 1.txt
WRITELN cat < 1.txt
WRITELN ls
WRITELN exit
```

```
mumsh $ mumsh $ 123
mumsh $ 1.txt
driver
mumsh
mumsh_memory_check
mumsh $ exit
```

B.4.4 Combine 4.1 and 4.2 with 4.3. [m1 13-14]

First ensure 4.1 to 4.3 work as expected.

B.5 Support for bash style redirection syntax; [m1 15-18]

The bash style redirection syntax is very complex, You are only required to implement part of it. The following example actually shows all of the requirements:

- An arbitrary amount of space can exist between the redirection symbols and arguments. [16]
- The redirection symbol can occur anywhere in the command. [17]

Example (Milestone 1 Case 18)

```
WRITELN echo 123 > 1.txt                   mumsh $ mumsh $ mumsh $ mumsh $ mumsh $ 123
WRITELN echo 222 > 2.txt                   mumsh $ 222
WRITELN echo 321 > 4.txt                   mumsh $ 222
WRITELN <1.txt>3.txt cat 2.txt 4.txt       321
WRITELN cat 1.txt                          mumsh $ 321
WRITELN cat 2.txt                          mumsh $ 1.txt
WRITELN cat 3.txt                          2.txt
WRITELN cat 4.txt                          3.txt
WRITELN ls                                 4.txt
WRITELN exit                               driver
                                           mumsh
                                           mumsh_memory_check
                                           mumsh $ exit
```

B.6 Pipes.

B.6.1 Basic pipe support. [m2 1-2]

Example (Milestone 2 Case 1)

```
WRITELN ls -a | grep mum                   mumsh $ mumsh
WRITELN exit                               mumsh_memory_check
                                           mumsh $ exit
```

B.6.2 Run all “stages” of piped process in parallel. [m2 3-4]

Your total runtime is limited to 1s for the test.

Example (Milestone 2 Case 3)

```
WRITELN sleep 0.6 | sleep 0.6              mumsh $ mumsh $ success
WRITELN echo success                       mumsh $ exit
WRITELN exit
```

A tricky test of executing the shell itself.

Example (Milestone 2 Case 4)

```
WRITELN echo exit | ./mumsh                mumsh $ mumsh $ exit
WRITELN exit                               mumsh $ exit
```

Hint. the shell is likely to get TLE (Time Limit Exceeded) if not run in parallel.

B.6.3 Extend 6.2 to support requirements 4. and 5. [m2 5-6]

Input redirection can only be found in the first command, and output redirection can only be found in the last command; each of them can only occur once. The test cases will strictly follow this rule.

Example (Milestone 2 Case 5)

```
WRITELN echo 123 > 1.txt
WRITELN echo 321 > 2.txt
WRITELN cat < 1.txt | cat >> 2.txt
WRITELN cat 1.txt
WRITELN cat 2.txt
WRITELN ls
WRITELN exit
```

```
mumsh $ mumsh $ mumsh $ mumsh $ 123
mumsh $ 321
123
mumsh $ 1.txt
2.txt
driver
mumsh
mumsh_memory_check
mumsh $ exit
```

B.6.4 Extend 6.3 to support arbitrarily deep “cascade pipes”. [m2 7-8]

Example (Milestone 2 Case 8)

```
WRITELN echo 123 > 1.txt
WRITELN echo 321 > 2.txt
WRITELN cat < 1.txt | cat | cat | cat | cat | cat | cat | cat | cat | cat | cat | cat | cat | cat | ca
WRITELN cat 1.txt
WRITELN cat 2.txt
WRITELN ls
WRITELN exit
```

```
mumsh $ mumsh $ mumsh $ mumsh $ 123
mumsh $ 321
123
mumsh $ 1.txt
2.txt
driver
mumsh
mumsh_memory_check
mumsh $ exit
```

B.7 Support CTRL-D. [m2 9-10]

If there is an unfinished command, nothing should happen. If the command line is empty, print `exit` and exit the shell.

B.8 Internal commands.

B.8.1 Implement pwd as a built-in command. [m2 11]

Do not use the builtin `pwd` command. Instead, implement it.

B.8.2 Allow changing working directory using cd. [m2 12]

You should consider these:

- Execute `pwd` , then execute `cd ..` followed by `cd .` and another `pwd`

- Execute `cd /etc/../etc/././../etc`
- Execute `cd` alone.
- Execute `cd` with more than 1 argument

B.8.3 Allow pwd to be piped or redirected as specified in requirement 4. [m2 13-14]

Example (Milestone 2 Case 13)

```
WRITELN pwd > 1.txt
WRITELN cat 1.txt
WRITELN ls
WRITELN exit
```

```
mumsh $ mumsh $ /out/package
mumsh $ 1.txt
driver
mumsh
mumsh_memory_check
mumsh $ exit
```

B.9 Support CTRL-C.

Your shell is likely to get TLE (Time Limit Exceeded) if CTRL-C is not handled correctly.

Do not use `kill(0, signal)`, it may destroy the judger. You will be sent to the honor council if you intentionally use it.

B.9.1 Properly handle CTRL-C in the case of requirement 4. [m2 15]

Example (Milestone 2 Case 15)

```
WRITELN sleep 10
CTRL C
WRITELN echo success
CTRL C
WRITELN exit
```

```
mumsh $
mumsh $ success
mumsh $
mumsh $ exit
```

B.9.2 Extend 9.1 to support subtasks 6.1 to 6.3. [m2 16]

Example (Milestone 2 Case 16)

```
WRITELN sleep 10 | echo 123
CTRL C
WRITELN echo success
WRITELN exit
```

```
mumsh $ 123

mumsh $ success
mumsh $ exit
```

B.9.3 Extend 9.2 to support requirement 7, especially on an incomplete input; [m2 17]

Example (Milestone 2 Case 17)

```
CTRL C                                     mumsh $
CTRL C                                     mumsh $
WRITELN exit                               mumsh $ exit
```

B.9.4 Extend 9.3 to support requirement 6. [m2 18]

Example (Milestone 2 Case 18)

```
WRITELN sleep 10 | sleep 10 | sleep 10 | sleep 10 | sleep 10 | sleep 10 | sleep 10 | sleep 10 | sleep
                                           mumsh $ 123
CTRL C
CTRL C                                     mumsh $
WRITELN echo success                       mumsh $ success
CTRL C                                     mumsh $
WRITELN exit                               mumsh $ exit
```

B.10 Support quotes.

B.10.1 Handle single and double quotes. [m3 1-2]

Hint. the program name can also be quoted.

Example (Final Pretest Case 2)

```
WRITELN "echo" 'abc def' "xm""fxh"         mumsh $ abc def xmfxh
WRITELN exit                               mumsh $ exit
```

B.10.2 Extend 10.1 to support requirement 4 and subtasks 6.1 to 6.3. [m3 3-4]

Example (Final Pretest Case 3)

```
WRITELN echo 123 > 1.txt                   mumsh $ mumsh $ mumsh $ 123
WRITELN "echo" "<1.'txt'" < 1.txt > "2.""txt"
                                           mumsh $ <1.'txt'
WRITELN cat 1.txt                          mumsh $ 1.txt
WRITELN cat 2.txt                          2.txt
WRITELN ls                                 driver
exit                                       mumsh
                                           mumsh_memory_check
                                           mumsh $ exit
```

Example (Final Pretest Case 4)

```
WRITELN "echo" "123" "|" | cat > 1.txt    mumsh $ mumsh $ 123 |
WRITELN cat 1.txt                         mumsh $ 1.txt
WRITELN ls                                driver
exit                                      mumsh
                                          mumsh_memory_check
                                          mumsh $ exit
```

B.10.3 Extend 10.2 in the case of incomplete quotes. [m3 5-6]

This part is similar to some functions in task 11. Consider them together. You should output "> " after the user hits enter.

Example (Final Pretest Case 6)

```
WRITELN echo 123 > 1.txt                  mumsh $ mumsh $ > mumsh $ 123
WRITELN echo "<1.'txt'                    mumsh $ <1.'txt'
WRITELN " < 1.txt >> 2."'txt'"
WRITELN cat 1.txt                         mumsh $ 1.txt
WRITELN cat 2."'txt'"                     2.'txt'
WRITELN ls                                driver
WRITELN exit                              mumsh
                                          mumsh_memory_check
                                          mumsh $ exit
```

B.10.4 Extend 10.3 to support requirements 4 and 6, together with subtask 9.3. [no pre-test]

Usually you do not need to modify many things for this task if you design your shell well. It is not tested in the final pretest, but will be tested in the final shell.

B.11 Wait for the command to be completed when encountering >, <, or |.

You should output "> " after the user hits enter.

B.11.1 Support requirements 3 and 4 together with subtasks 6.1 to 6.3. [m3 7-12]

Hint: ">>" does not need to be supported. You should output "> " after the user hits enter.

Test cases content:

- Cases 7-8: >;
- Cases 9-10: <;
- Cases 11-12: |;

Example (Final Pretest Case 7)

```
WRITELN echo 123 >                          mumsh $ > mumsh $ 123
WRITELN 1.txt                               mumsh $ 1.txt
WRITELN cat 1.txt                           driver
WRITELN ls                                  mumsh
WRITELN exit                                mumsh_memory_check
                                            mumsh $ exit
```

Example (Final Pretest Case 9)

```
WRITELN echo 321 > 1.txt                    mumsh $ mumsh $ > 321
WRITELN cat <                               mumsh $ 1.txt
WRITELN 1.txt                               driver
WRITELN ls                                  mumsh
WRITELN exit                                mumsh_memory_check
                                            mumsh $ exit
```

Example (Final Pretest Case 11)

```
WRITELN echo 321 |                          mumsh $ > 321
WRITELN cat                                 mumsh $ exit
WRITELN exit
```

B.11.2 Extend 11.1 to support requirement 10. [m3 13-14]

Hint. Actually 10.3 already has this functionality, so you only need to extend to 10.1, 10.2, 10.4.

Example (Final Pretest Case 13)

```
WRITELN echo 123 > 1.txt                    mumsh $ mumsh $ > > > mumsh $ 123
WRITELN "echo" "<                           mumsh $ <
WRITELN 1.'txt'" <                          1.'txt'
WRITELN 1.txt >                             mumsh $ 1.txt
WRITELN "2.""txt"                           2.txt
WRITELN cat 1.txt                           driver
WRITELN cat 2.txt                           mumsh
WRITELN ls                                  mumsh_memory_check
exit                                        mumsh $ exit
```

B.12 Handle errors for all supported features. [m3 15-22]

There are eight kinds of errors to be handled. In the test cases of other tasks, none of these errors will be included. You will always get a correct command then. At most one error will occur in a command, so you do not need to consider about the order of errors.

The output format of your errors should strictly follow the format listed below. It is very similar to the bash style error output format.

1. Non-existing program [m3 15]
   - input: `non-exist abc def`
   - input: `echo abc | non-exist`
   - output: `non-exist: command not found`
2. Non-existing file in input redirection [m3 16]
   - input: `cat < non-existing.txt`
   - output: `non-existing.txt: No such file or directory`
3. Failed to open file in output redirection [m3 17]
   - input: `echo abc > /dev/permission_denied`
   - output: `/dev/permission_denied: Permission denied`
4. Duplicated input redirection [m3 18]
   - input: `echo abc < 1.txt < 2.txt`
   - input: `echo abc | grep abc < 1.txt`
   - output: `error: duplicated input redirection`
5. Duplicated output redirection [m3 19]
   - input: `echo abc > 1.txt > 2.txt`
   - input: `echo abc > 1.txt >> 2.txt`
   - input: `echo abc > 1.txt | grep abc`
   - output: `error: duplicated output redirection`
6. Syntax Error [m3 20]
   - input: `echo abc > > > >`
   - output: ``syntax error near unexpected token `>'``
   - input: `echo abc > < 1.txt`
   - output: ``syntax error near unexpected token `<'``
   - input: `echo abc > | grep abc`
   - output: ``syntax error near unexpected token `|'``
7. Missing program [m3 21]
   - input: `echo 123 | | grep 123`
   - output: `error: missing program`
8. `cd` to non-existing directory [m3 22]
   - input: `cd non-existing`
   - output: `non-existing: No such file or directory`

B.13 A command ending with an & should be run in background.

B.13.1 For any background job, print out the command line, prepended with the job ID [m3 23]

Hints.

- Do not print the process ID.

- Do not print complete information or ”mumsh $” after the job is finished (like ‘bash‘ or ‘zsh‘).
- The job ID starts from 1.

Example (Final Pretest Case 23)

```
WRITELN /bin/ls &                          mumsh $ [1] /bin/ls &
WRITELN sleep 0.2                          mumsh $ driver
WRITELN /bin/ls | cat &                    mumsh
WRITELN sleep 0.2                          mumsh_memory_check
WRITELN exit                               mumsh $ [2] /bin/ls | cat &
                                           mumsh $ driver
                                           mumsh
                                           mumsh_memory_check
                                           mumsh $ exit
```

B.13.2 Implement the command jobs which prints a list of background tasks together with their running states [m3 24]

Hint. Print all of done and running background processes in this shell.

Example (Final Pretest Case 24)

```
WRITELN /bin/ls &                          mumsh $ [1] /bin/ls &
WRITELN sleep 0.2                          mumsh $ driver
WRITELN sleep 0.2 &                        mumsh
WRITELN jobs                               mumsh_memory_check
WRITELN sleep 0.4                          mumsh $ [2] sleep 0.2 &
WRITELN exit                               mumsh $ [1] done /bin/ls &
                                           [2] running sleep 0.2 &
                                           mumsh $ mumsh $ exit
```