Operating Systems Lab-Course

Project 2: *Command Shell*

Robert Engelke
Stephan Rudolph
John Trimpop
Sven Berger

Content

1. Introduction
2. Main Features
3. What is a parser?
4. First steps
5. Realization of task a: Pipes

1. Introduction

Main task:

- *Implementation of own command shell in Unix/Linux*
- *Must support the given commands*

Subtask a)

- *Concatenation of different commands by using a pipe (|)*

1. Introduction

Environment:

- Ubuntu Linux with default terminal and gcc

Responsibilities:

- Everyone: *Presentation and documentation*
- Robert Engelke: *Implementation of Pipes*
- Stephan Rudolph: *Implementation of Background processes*
- John Trimpop, Sven Berger: *Implementation Commands and Parser*

2. Main Features

- Sequential execution: `foo ; bar`
- Ignore comments until end of line: `# foo bar`
- Protected Characters: `'foo bar'` or `/foo`
- Background process: `foo &`
- Abort foreground process: `^C`
- Read from file/ print to file: `< file` or `> file`

2. Main Features

- Declare variable: `setenv foo`(variable) `bar`(value)
- Change path directory: `cd foo/bar`
- Pipes: `foo | bar`
- Exit Shell: `exit`

3. What is a parser?

- Component of interpreter or compiler
- Checks for correct syntax and builds data structure
- Programmed by hand or via generator tools

- Command shell needs a parser for interpreting certain commands
- Prepared for our project
- To do: Connection between parser and our shell

4. First steps

- First step: adressing the parser
- Approach via `exit` command

```
[...]
#include "parser.h"

main(void){
    char line[512];

    printf("\nSuperMegaShell> ");
    scanf("%s", &line);

    cmds *cmd = parser_parse(line);
    if (cmd != NULL){
        switch(cmd->kind){
            case EXIT:
                printf("\nClosing SuperMegaShell...\n");
                Return 0;
[...]
```

4. First steps

- Important: to compile, we have to use:

```
gcc shell.c -o shell.o parser.c
```

- (Otherwise methods of this file cannot be found)
- Now other capabilities of our shell may follow

5. Realization of task a: Pipes

- Pipes, `foo | bar` i.e.:
  1. Execute command foo
  2. Set result of `foo` as input of `bar`
  3. Execute command `bar` and print result

- Also possible as background process:
  `(foo | bar)&`

5. Realization of task a: Pipes

- **Possible steps:**
  - *Creating two child processes*
  - *In- and output*
  - *Both processes can read and write at beginning*
  - *Reading for process 1 has to be blocked*
  - *Process 1 then writes to pipe*
  - *Writing for process 2 has to be blocked*
  - *Process 2 then reads from pipe*
  - *Resulting to output (wether if fore- or background)*