Excercise Sheet 4

Johannes Koch

May 1, 2018

1 Task 2

```
#include <stdlib.h>
#include <stdio.h>
#include <unistd.h>
#include <sys/wait.h>
#include <sys/types.h>

int main(int argc, char  *argv[]){

  pid_t ls, grep;
  int pipefd[2];

  // initialize unnamed pipe
  if(pipe(pipefd) == -1){
    perror("pipe");
    return EXIT_FAILURE;
  }

  // create ls and grep processes
  if( (ls = fork()) ){
    if( (grep = fork()) );
  }

  // check if forks successful
  if(ls == -1 || grep == -1){
    perror("fork");
    return EXIT_FAILURE;
  }

  // ls execs ls and redirects its output in the pipe
  if(ls == 0){
    // call ls and get output into pipe
    close(pipefd[0]);   // close read end
    dup2(pipefd[1], 1); // redirect stdout to pipe
    close(pipefd[1]);

    // call ls
    execlp("ls", "ls", NULL);
    // exit on error
    perror("execlp");
    _exit(EXIT_FAILURE);
  }

  // grep execs grep on pipe content
  if(grep == 0){

    close(pipefd[1]);   // close write end
    dup2(pipefd[0], 0); // use pipe read end as stdin
    close(pipefd[0]);

    // call ls
    //execlp("grep", "grep", argv[1], NULL);

    char *parameters[argc + 1];
    parameters[0] = "grep";
    for(int i = 1; i < (argc - 1); i++){
      parameters[i] = argv[i + 1];
    }
    parameters[argc] = NULL;
    execvp("grep", parameters);

    // exit on error
    //perror("execlp");
    perror("execv");
    _exit(EXIT_FAILURE);
  }

  close(pipefd[0]);
  close(pipefd[1]);

  // wait for children
  while(wait(NULL) > 0);

  return EXIT_SUCCESS;
}
```