National Research University ITMO

Faculty of Software Engineering and Compyter Systems

Labwork №4 [4]

System Software Fundamentals

Ivan Sarzhevskiy

Group P3302

Saint Petersburg

2019 г.

Assignment

Part 1

Implement *tail* utility using C and raw syscalls.

Requirements

1. I/O only via `read(2)` and `write(2)`.
2. Multiple input files standard input handling.
3. Error handling.

Part 2

Rewrite the same program in Perl.

Requirements

1. Use `use strict; use warnings qw(FATAL all);` pragmas.
2. Enable *taint mode* with `#!/usr/bin/perl -T`.

Part 3

Implement *xargs* utility using C and raw syscalls.

Code Listing

Makefile

```
1  GCC = gcc
2  FLAGS = -Wall -Wextra --std=gnu99 -pedantic -Werror -g
3
4  all : tail xargs
5
6  tail : keker_tail.c
7          $(GCC) $(FLAGS) -o keker_tail keker_tail.c
8
9  xargs : keker_xargs.c
10         $(GCC) $(FLAGS) -o keker_xargs keker_xargs.c
11
12 clean :
13         rm keker_tail keker_xargs
```

tail (C implementation)

```c
#include <stdio.h>
#include <unistd.h>
#include <stdlib.h>
#include <string.h>
#include <fcntl.h>
#include <errno.h>
#include <stdbool.h>

#define WORD_MODE_FLAG 1

const int BUFSIZE = 4096;
const char* INVALID_NUM = "Cant parse given number\n";

bool parse_int(const char* str, long* var) {
  char* end;
  *var = strtol(str, &end, 10);
  if (end == str) return false;
  return true;
}

char* get_file_contents(int fd) {
  char buffer[BUFSIZE];
  char* contents = NULL;
  int byte_count, real_buf_size = 0;
  while ((byte_count = read(fd, buffer, BUFSIZE)) > 0) {
    contents = realloc(contents, real_buf_size + byte_count);
    memcpy(contents + real_buf_size, buffer, byte_count);
    real_buf_size += byte_count;
  }
  contents = realloc(contents, real_buf_size + 1);
  contents[real_buf_size] = '\0';
  return contents;
}

void print_last_n_lines(char* contents, int n, int flags) {
  char* end = contents + strlen(contents);
  char* curr_char;
  int curr_n = 0;
  if (*(end - 1) == '\n')
    curr_n--;

  if ((flags & WORD_MODE_FLAG) != 0)
    for (curr_char = end - 2; *curr_char == '\n'; --curr_char)
      curr_n--;

  for (curr_char = end; curr_char != contents; --curr_char) {
    if (*curr_char == '\n' || ((flags & WORD_MODE_FLAG) != 0
          && (*curr_char == ' '
          && *(curr_char + 1) != '\n'
          && *(curr_char - 1) != '\n'
          && *(curr_char + 1) != ' ')))
      curr_n++;
    if (curr_n == n)
      break;
  }
  if (*curr_char == '\n' || (((flags & WORD_MODE_FLAG) != 0) && *curr_char == ' '))
    curr_char++;
  write(STDOUT_FILENO, curr_char, end - curr_char);
}

int main(int argc, char *argv[]) {
  int opt = 0;
  long num_lines = 10;
  int flags = 0;

  while ((opt = getopt(argc, argv, "n:w")) != -1) {
    switch (opt) {
```

```
      case 'n':
        if (!parse_int(optarg, &num_lines)) {
          write(STDERR_FILENO, INVALID_NUM, strlen(INVALID_NUM));
          return 1;
        }
        break;
      case 'w':
        flags |= WORD_MODE_FLAG;
        break;
      default:
        return 1;
    }
  }
  char* contents;

  if (optind != argc && *argv[optind] != '-') {
    if (optind + 1 == argc) {
      int fd = open(argv[optind], O_RDONLY);
      if (errno != 0) {
        char* error = strerror(errno);
        write(STDERR_FILENO, error, strlen(error));
        write(STDERR_FILENO, "\n", 1);
        return 1;
      }
      contents = get_file_contents(fd);
      close(fd);
      print_last_n_lines(contents, num_lines, flags);
    }
    else {
      for (int i = optind; i < argc; ++i) {
        if (*argv[i] == '-') {
          write(STDOUT_FILENO, "==> standard input <==\n", 23);
          contents = get_file_contents(STDIN_FILENO);
          print_last_n_lines(contents, num_lines, flags);
          write(STDOUT_FILENO, "\n", 1);
        }
        else {
          int fd = open(argv[i], O_RDONLY);
          if (errno != 0) {
            char* error = strerror(errno);
            write(STDERR_FILENO, error, strlen(error));
            write(STDERR_FILENO, "\n", 1);
            return 1;
          }
          write(STDOUT_FILENO, "==> ", 4);
          write(STDOUT_FILENO, argv[i], strlen(argv[i]));
          write(STDOUT_FILENO, " <==\n", 5);
          contents = get_file_contents(fd);
          close(fd);
          print_last_n_lines(contents, num_lines, flags);
          write(STDOUT_FILENO, "\n", 1);
        }
      }
    }
  }
  else {
    contents = get_file_contents(STDIN_FILENO);
    print_last_n_lines(contents, num_lines, flags);
  }
}
```

tail (Perl implementation)

```
#!/usr/bin/perl -T

use strict;
use warnings qw(FATAL all);
use Getopt::Long;

my $num_lines = 10;
my $word_mode;
my $first_out = 0;

# cmd line args parsing
GetOptions ("n=i" => \$num_lines,
            "w"   => \$word_mode)
or die("Error in command line arguments\n");

foreach my $filename (@ARGV) {
  my $fh;
  if ($filename eq "-") {
    $fh = "STDIN";
  }
  else {
    open($fh, "<", $filename)
      or die("Cannot open $filename\n");
  }
  if (scalar @ARGV != 1) {
    if ($filename eq "-") {
      if ($first_out == 0) {
        print "==> standard input <==\n";
        $first_out++;
      }
      else {
        print "\n==> standard input <==\n";
      }
    }
    else {
      if ($first_out == 0) {
        print "==> $filename <==\n";
        $first_out++;
      }
      else {
        print "\n==> $filename <==\n";
      }
    }
  }
  if ($word_mode) {
    my @words;
    my $processed = 0;
    while (my $line = <$fh>) {
      my @c_words = split /\s+/, $line;
      foreach my $word (@c_words) {
        if ($processed < $num_lines) {
          push (@words, "$word ");
          $processed++;
        }
        else {
          shift @words;
          push @words, "$word ";
        }
      }
      my $last_word = pop @words;
      push @words, "$last_word\n";
      if (eof) {
        foreach my $word (@words) {
          print $word;
        }
```

```
        @words = ();
      }
    }
  }
  else {
    my @lines;
    my $processed = 0;
    while (my $line = <$fh>) {
      if ($processed < $num_lines) {
        push (@lines, $line);
        $processed++;
      }
      else {
        shift @lines;
        push (@lines, $line);
      }
      if (eof) {
        foreach my $line (@lines) {
          print $line;
        }
        @lines = ();
      }
    }
  }
}
```

xargs (C implementation)

```
#include <unistd.h>
#include <stdlib.h>
#include <string.h>

#define BUF_SIZE 4096
#define MAX_ARGS_LEN 2090996

void parse_arguments(char* command_buffer) {
  int init_position = strlen(command_buffer);

  char inbuf[BUF_SIZE];
  int bytes_read;
  while (init_position < MAX_ARGS_LEN - 1 &&
         (bytes_read = read(STDIN_FILENO, inbuf, BUF_SIZE)) > 0)
    for (int pos = 0; pos < bytes_read && init_position < MAX_ARGS_LEN - 1; ++pos)
      command_buffer[init_position++] = inbuf[pos] == '\n' ? ' ' : inbuf[pos];
}

int main(int argc, char** argv) {
  char* command_buffer = calloc(1, MAX_ARGS_LEN);

  if (argc == 1) {
    strcat(command_buffer, "echo ");
  }
  else {
    for (int i = 1; i < argc; ++i) {
      strcat(command_buffer, argv[i]);
      strcat(command_buffer, " ");
    }
  }

  parse_arguments(command_buffer);
  return system(command_buffer);
}
```