ITMO UNIVERSITY

National Research University ITMO

Faculty of Software Engineering and Computer Systems

SYSTEM SOFTWARE FUNDAMENTALS

Lab Work #4 (3)

System Calls

Timothy Labushev

Group P3302

Saint Petersburg

2019

Assignment

Using raw system calls, write a C program that mimics the behavior of the *head* utility.

Requirements

1. The program should perform input and output via `read(2)` and `write(2)`.
2. The program should accept multiple input files and use standard input when - is given as a filename.
3. The program should handle errors and print informative messages to `stderr`.

Relevant Sections from POSIX.1-2008

Synopsis

`head [-n` *`number`*`] [`*`file`*`...]`

Description

The *head* utility shall copy its input files to the standard output, ending the output for each file at a designated point.

Copying shall end at the point in each input file indicated by the **-n** *number* option. The option-argument number shall be counted in units of lines.

Options

The following option shall be supported:

-n *number*

The first *number* lines of each input file shall be copied to standard output. The application shall ensure that the number option-argument is a positive decimal integer.

When a file contains less than number lines, it shall be copied to standard output in its entirety. This shall not be an error.

If no options are specified, *head* shall act as if **-n 10** had been specified.

Stdout

The standard output shall contain designated portions of the input files.

If multiple *file* operands are specified, *head* shall precede the output for each with the header:

`"\n==> %s <==\n", <pathname>`

except that the first header written shall not include the initial <newline>.

Code Listing

```c
#define _POSIX_C_SOURCE 2
#include <unistd.h>
#include <stdlib.h>
#include <stdio.h>
#include <fcntl.h>
#include <stdbool.h>
#include <stdarg.h>

#define BUF_SIZE 4096

void print_lines_buffered(int fd, unsigned int num_lines) {
  unsigned int printed = 0;

  char inbuf[BUF_SIZE];
  int bytes_read;
  while (printed < num_lines && (bytes_read = read(fd, inbuf, BUF_SIZE)) > 0) {
    unsigned int pos;
    for (pos = 0; pos < bytes_read; ++pos)
      if (inbuf[pos] == '\n' && ++printed == num_lines) break;

    write(STDOUT_FILENO, inbuf, pos);
  }

  write(STDOUT_FILENO, "\n", sizeof("\n"));
}

bool try_parse_uint(const char* str, unsigned int* val) {
  char* endptr;
  *val = strtoul(str, &endptr, 10);
  return *str != '-' && endptr != str && *endptr == '\0';
}

// An fprintf()-like wrapper over write().
// Rationale: the assignment requires we only use raw syscalls to perform IO.
void fdprintf(int fd, const char* fmt, ...) {
  #define MSG_BUF_SIZE 512
  char buf[MSG_BUF_SIZE];
  va_list args;
  va_start(args, fmt);
  int msg_len = vsnprintf(buf, MSG_BUF_SIZE, fmt, args);
  va_end(args);
  write(fd, buf, msg_len < MSG_BUF_SIZE ? msg_len : MSG_BUF_SIZE);
}

void handle_filename_args(int argc, char** argv, int num_lines) {
  // According to POSIX,
  // If multiple file operands are specified, head shall precede the output for each with the header:
  // "\n==> %s <==\n", <pathname>
  // except that the first header written shall not include the initial <newline>.
  #define HEADER_STDIN "\n==> standard input <==\n"
  #define HEADER_FILE_FMT "\n==> %s <==\n"

  bool first_header = true;
  do {
    int header_offset = first_header ? 1 : 0;
    first_header = false;

    if (*argv[optind] == '-') {
      write(STDOUT_FILENO, HEADER_STDIN + header_offset, sizeof(HEADER_STDIN) - header_offset);
      print_lines_buffered(STDIN_FILENO, num_lines);
    }
    else {
      int fd = open(argv[optind], O_RDONLY);
      if (fd == -1) {
        fdprintf(STDERR_FILENO, "%s: Cannot open %s for reading", argv[0], argv[optind]);
      }
      else {
```

```
68          fdprintf(STDOUT_FILENO, HEADER_FILE_FMT + header_offset, argv[optind]);
69          print_lines_buffered(fd, num_lines);
70          close(fd);
71        }
72      }
73    }
74    while (++optind < argc);
75  }
76
77  int main(int argc, char** argv) {
78    unsigned int num_lines = 10;
79
80    int c;
81    while ((c = getopt(argc, argv, "n:")) != EOF) {
82      switch (c) {
83        case 'n':
84          if (!try_parse_uint(optarg, &num_lines)) {
85            fdprintf(STDERR_FILENO, "%s: invalid number of lines: '%s'\n", argv[0], optarg);
86            return 1;
87          }
88          break;
89        case '?':
90          fdprintf(STDERR_FILENO, "Usage: %s [-n num-lines] [file...]\n", argv[0]);
91          return 1;
92      }
93    }
94
95    if (optind == argc)
96      print_lines_buffered(STDIN_FILENO, num_lines);
97    else
98      handle_filename_args(argc, argv, num_lines);
99
100   return 0;
101 }
```