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

Part I

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`.

Part II

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 III

Write a C program that mimics the behavior of the *xargs* utility without optional arguments.

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

Makefile

```
1  CC=gcc
2  CFLAGS=-std=c11 -g -Wall -Wextra -pedantic -Werror
3
4  all: head xargs
5
6  head: head.c
7          $(CC) $(CFLAGS) -o head head.c
8
9  xargs: xargs.c
10         $(CC) $(CFLAGS) -o xargs xargs.c
11
12 clean:
13         @rm head xargs
```

head (C)

```c
#define _POSIX_C_SOURCE 2
#include <unistd.h>
#include <stdlib.h>
#include <stdio.h>
#include <fcntl.h>
#include <stdbool.h>
#include <stdarg.h>
#include <errno.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) {
    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) {
  errno = 0;
  char* endptr;
  *val = strtoul(str, &endptr, 10);
  return *str != '-' && errno == 0 && 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 {
      errno = 0;
      int fd = open(argv[optind], O_RDONLY);
      if (fd == -1) {
        const char* error_fmt = "%s: Cannot open %s for reading\n";
```

```
        switch (errno) {
          case EACCES:
            error_fmt = "%s: Cannot open %s for reading (access denied)\n";
            break;
          case ENOENT:
            error_fmt = "%s: Cannot open %s for reading (file not found)\n";
            break;
        }
        fdprintf(STDERR_FILENO, error_fmt, argv[0], argv[optind]);
      }
      else {
        fdprintf(STDOUT_FILENO, HEADER_FILE_FMT + header_offset, argv[optind]);
        print_lines_buffered(fd, num_lines);
        close(fd);
      }
    }
  }
  while (++optind < argc);
}

int main(int argc, char** argv) {
  unsigned int num_lines = 10;

  int c;
  while ((c = getopt(argc, argv, "n:")) != EOF) {
    switch (c) {
      case 'n':
        if (!try_parse_uint(optarg, &num_lines)) {
          fdprintf(STDERR_FILENO, "%s: invalid number of lines: '%s'\n", argv[0], optarg);
          return 1;
        }
        break;
      case '?':
        fdprintf(STDERR_FILENO, "Usage: %s [-n num-lines] [file...]\n", argv[0]);
        return 1;
    }
  }

  if (optind == argc)
    print_lines_buffered(STDIN_FILENO, num_lines);
  else
    handle_filename_args(argc, argv, num_lines);

  return 0;
}
```

head (Perl)

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

# Run
# cpan Getopt::Long
# to fetch script dependencies

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

my $num_lines = 10;

GetOptions("n=i" => \$num_lines) or die("Usage: $0 [-n num-lines] [file...]\n");
$num_lines > 0 or die("$0: invalid number of lines: '$num_lines'\n");

my $printed = 0;
my $first_file = 1;

while (1) {
  if ($printed == 0) {
```

```
    if (not @ARGV and $first_file) {
      # No arguments = don't print the header, read from stdin
      undef $first_file;
    }
    elsif (@ARGV) {
      my $leading_newline = $first_file ? "" : "\n";
      my $filename = $ARGV[0] eq "-" ? "standard input" : $ARGV[0];
      print "$leading_newline==> $filename <==\n";
      undef $first_file;
    }
    else {
      last;
    }
  }
  if ($printed++ < $num_lines) {
    my $line = <>;
    print $line;
  }
  if ($printed == $num_lines or eof) {
    $printed = 0;
    close ARGV;
  }
}
```

xargs (C)

```
#define _POSIX_C_SOURCE 2
#include <unistd.h>
#include <stdlib.h>
#include <string.h>

#define BUF_SIZE 4096
#define MAX_ARG_SIZE 1024 * 1024

void read_arg_string(char* arg_buf) {
  unsigned int args_pos = strlen(arg_buf);

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

int main(int argc, char** argv) {
  char* arg_string = calloc(1, MAX_ARG_SIZE);

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

  read_arg_string(arg_string);
  return system(arg_string);
}
```