Design Document: bobcat

Perry David Ralston Jr.
CruzID: pdralsto

October 9, 2020

1 Goals

The goal of this program is to mimic the functionality of `cat(1)`. This program will handle reading and writing from/to standard input and files without using `FILE`* functions. `bobcat` is expected to work in the Ubuntu 20.04 environment.

2 Design

The design of this program consists of processing the inputs from the command line and then using system calls to complete the tasks as described by the user commands.

2.1 Handling Arguments

In accordance with `cat(1)`, no argument, or an argument of `-`, should be considered as reading from standard input. bobcat will not handle any additional flags outside of the stdin FILE designation, `-`. All `-` arguments after the first `-` are ignored. Paths, given as a string argument, to bobcat will result in the contents of the file of the given path to be printed to standard output. Initialization is detailed in Algorithm 1.

```
Input : Argument count: argc
Input : Array of arguments: argv
Output : returns 0 upon successful completion, non-zero
    otherwise
if argc == 1
  if (read_file(0) == -1 or close(fd) == -1)
    print error to stderr and exit(errno)
for file in argv[1 -> argc-1]
  if file == '-' and stdin_read == false
    fd = 0
    stdin_read = true
  if file != '-'
    fd = open(file)
  if fd != null
    if (fd == -1 or read_file(fd) == -1 or close(fd) == -1)
      print error to stderr and exit(errno)
exit(0)
```

Algorithm 1: Initialization

2.2 Printing

Printing will be handled by calling the local `read_file(file_descriptor)` function. This function is passed the file descriptor of the opened file and uses syscalls to `read(2)` and `write(2)` to complete the task. A global constant `BUFF_SIZE` is used to create the local variable buffer and buffer is used to store the information read from the file. As long as there is input read into the buffer, the buffer will be passed to the `write(2)` function with the global constant `STDOUT`, representing the fd for stdout, to print the buffer to stdout. When an error is detected, a non-zero value is passed back to the caller for handling. See Algorithm 2 for details.

```
Input : File descriptor: fd
Output : returns 0 upon successful completion, -1 otherwise

bytes_read = read(fd, buffer, BUFFER_SIZE)
while bytes_read > 0
  if write(STDOUT, buffer, BUFFER_SIZE) == -1
    return(-1)
if bytes_read = -1 or write(STDOUT, buffer, BUFF_SIZE) == -1
  return(-1)
return(0)
```

Algorithm 2: Printing