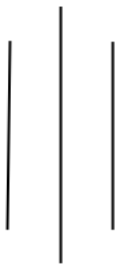
Re-Implementing cat command from GNU Core Utilities

A Project Report



Submitted By

Anukul Adhikari

Roll No. 10

Date: August 17, 2020

Submitted To

Bhaktapur Multiple Campus

Department of Computer Science and Information Technology

Doodhpati, Bhaktapur, Nepal

Under the Supervision of

Mr. Sushant Poudel

Contents

Acknowledgements

All thanks to my adorable parents for their profound help and support during the cause of this project work.Without constant guidance and suggestions, this report would have been nowhere near completion. Finally, I thank to all my teachers and friends, who were the people, who prepared me for this endeavor. I own you all my success.

Introduction

The shell is a program in unix like system that takes command from the input and the operating system performs on that.It was the primary interface to interact with unix like systems but nowadays graphical user interface are primarilly used with addition to command line interfaces such as shell.

Linux system is combination of GNU and linux kernel. GNU default shell is **bash**(which stands for Bourne Again SHell an enhanced version of original unix shell) - acts as shell program. Some other shell programs include : zsh, fish, ksh, tcsh.

A terminal is a program called **terminal emulator**. This program opens window and lets you interact with shell. There are many terminal emulators some major terminal emulators shipped with linux distributions are: gnome-terminal, rxvt, konsole, xterm, st.

A terminal emulator allows user to access to a text terminal and all applications such as command line interfaces and text user interfaces. These helps user to access same machine or different machine commonly using **ssh**.

Today's, general user rely upon graphical user interfaces. However majority of programming and maintenance tasks don't provide graphical interface and still use a command line.

Programs that allow command line interface are generally easier to automate via scripting.

One of the most used coreutility shipped with GNU is **cat**(short term for "concatenate"). The cat command reads each file parameter given in sequence and writes to stdout. If no filename is specified , the cat command reads from stdin (ie. specify filename - for stdin).

System Study

This program uses unix/posix system exclusive headers and functions and is not meant to work on windows based system.

Software

Following compiler and configuration is verified to work with the snippets in this report:

```
Compiler - gcc (GCC) 10.1.0, clang 10.0.1
Compiler target - x86_64-pc-linux-gnu, aarch-linux-android
Thread model: posix
```

Requirement Analysis

Objective

- To take filename/path as argument.
- To read from standard input if no filename or '-' is specified.
- To dynamically allocate required buffer for quick read/write in memory.
- To calculate read/write bytes.
- To use buffer to take input from read file descriptor and output to write file descriptor.
- To make sure every read and write bytes are equal
- To free allocated memory and close open file directory.

Data Flow

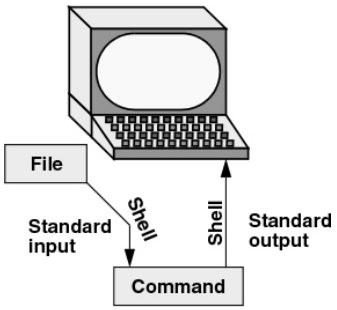


Figure 1: Data flow diagram

```
$ ./cat File
.....
.....
```

The argument File is standard input into the shell to the command and to standard output.
Multiple file can be fed into as arguments, files are opened and stdout in order and is outputted together in link of series.

The file hello.txt contains some text. use cat on hello.txt multiple times.

```
$ ./cat hello.txt hello.txt
Hi my name is Anukul.
Hi my name is Anukul.
```

Implementation

Header Files

- **stdlib.h** is for general purpose standard library, which declares a variety of utility functions for type conversions, memory allocation, process control and other similar tasks.
- **stdio.h** defines several macros and declares three types and many functions for performing input and output.
- **string.h** defines macros, constants and declarations of functions and typrs used not only for string handeling but also various memory handling functions.
- **fcntl.h** is the C POSIX library for c programming language that contains constructs that refer to file control.
- **sys/stat.h** is the C POSIX library for c programming language that contains constructs that facilitate getting information about files attributes.
- **unistd.h** defines miscellaneous symbolic constants and types, and declares miscellaneous functions.
- **err.h** defines the integer variable errno, which is set by system call and some library functions of errors to indicate what error is occurring.

Functions Used

- **int main(int argc, char *argv[ ])]** command line arguments is passed, from main function before running program first argument is the executable file itself and the second one is the argument form user
  ie. argv[0] holds executable name, argv[1] holds first argument name arguments must be passed with space in between

- **fileno()** the function examines the argument and returns the integer file descriptor.
  ie. fileno(stdin) returns 0 later useful in program

- **strcmp()** the function compares two strings if: both string are equal it returns 0, negative if first string is les than second string
  ie. strcmp("hello", "hello") returns 0.

- **open()** the function opens file by specified pathname if not it may create the file in that path it returns file descriptor greater that 2 because they are 0,1,2 are used by stdin,stdout,stderr open commonly used flags are O_RDONLY (read-only), O_WRONGLY (write-only), O_RDWR (read/write) these are commonly used file permissions
  ie. open(*argv, O_RDONLY) opening pathname given by *argv in readonly mode.

- **fstat()** the function obtains information about file associated with file descriptor and write it to area pointed by buffer.
  ie. fstat(wfd, &sbuf) is used to access information from stat structure on wfd.

- **malloc()** the function allocates size bytes and returns a pointer to the allocated memory
  ie. malloc(bsize) returns pointer to appropriate allocated blocksize

- **read()** the function reads upto bsize bytes from file descriptor to the buffer
  ie. read(rfd, buf, bsize) the file indicated by rfd is input to memory area indicated by buffer with calculated blocksize

- **write()** the function writes upto bsize bytes from buffer to file referred by file descriptor.
  ie. write(wfd, buf, bsize) the file indicated by the wfd is output to the memory area indicated by the buffer with calculated blocksize

- **free()** the function releases allocated memory
  ie. free(buf) freeing used buffer memory

- **close()** the function closes file descriptor so it no longer refers to any file. ie. close(fd) closing opened file descriptor.

Memory Leak & Performance

Memory Leak is when memory is allocated and not freed after use, or when the pointer to a memory allocation is deleted, rendering the memory no longer usable. Memory leak degrade performance due to increased paging, and overtime causes a program to crash. When program runs out of memory it may cause Linux kernel to crash.

Compiler doesn't show memory leak so one must use other tools to detect memory leak. The straightforward way is to use gnu debugger with valgrind both are debugging tool native to linux system. Detailed memory leak check required some additional compiler arguments.

```
$ gcc -Wall -ggdb3 -o cat cat.c
```

- **ggdb3** produces extra debugging information including macro definitions and line number same as syntax error shows line number in compiler.

Debugging took most time and all the issues arised was due to non freed allocated memory. This caused system to crash due to low system memory and larger file size. Later found out it was due to memory not freed before exiting the program.

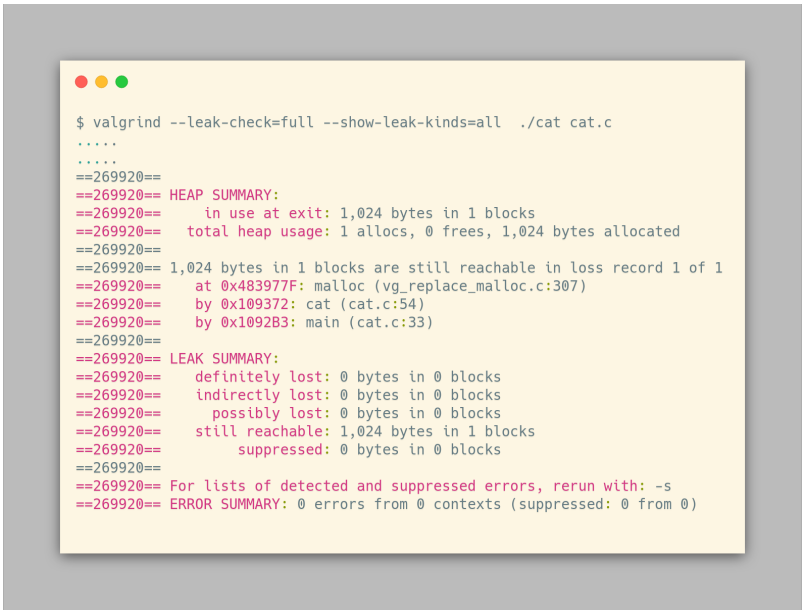


Figure 2: Memory Leak caused by not freeing malloc block

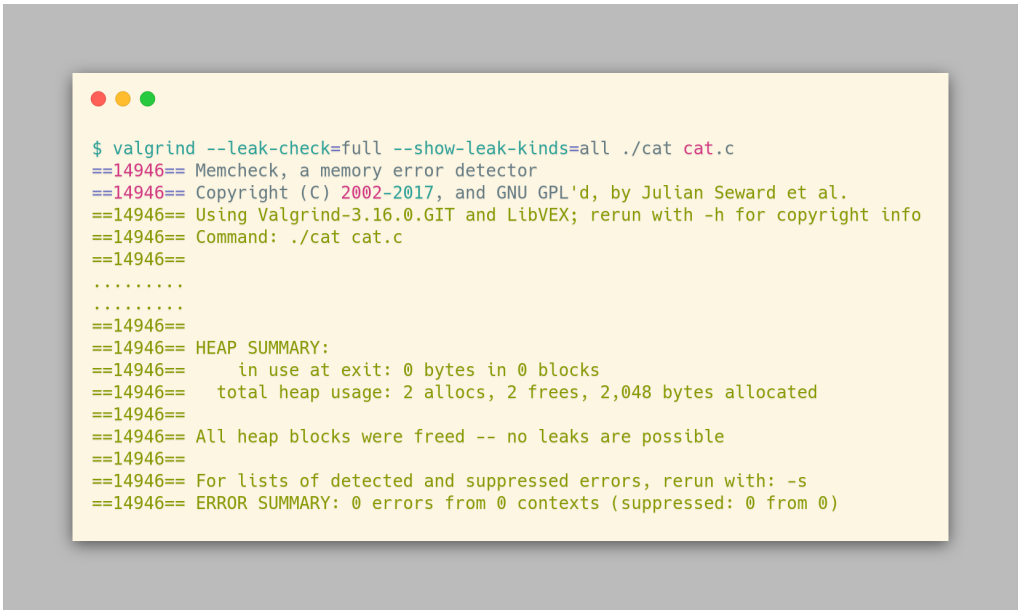


Figure 3: No Memory Leak after freeing malloc block

Lets compare the re-implemented cat and original gnu coreutil cat.

```
$ time ./cat cat.c
.................................
.................................

        }
        nr = read(rfd, buf, bsize);
    }
    free(buf);
}

real    0m0.003s
user    0m0.000s
sys     0m0.003s

$ time cat cat.c
.................................
.................................

        nr = read(rfd, buf, bsize);
    }
    free(buf);
}

real    0m0.003s
user    0m0.003s
sys     0m0.000s
```

It is **identical**.

Source Code

```c
#include<stdlib.h>
#include<stdio.h>
#include<string.h>
#include<fcntl.h>
#include<sys/stat.h>
#include<unistd.h>
#include<err.h>

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

    ++argv;

    int fd;
    fd = fileno(stdin);

    while(*argv != 0) {

        if(strcmp(*argv, "-") == 0) {
            fd = fileno(stdin);
        }
        else {
            fd = open(*argv, O_RDONLY);
        }
        if(fd < 0) {
            err(1, "%s", *argv);
        }
        ++argv;
        cat(fd);
        if (fd != fileno(stdin)) {
            close(fd);
```

```
        }
    }
    cat(fd);
    close(fd);
}

void cat(int rfd) {
    int wfd;
    static char *buf;
    static size_t bsize;
    struct stat sbuf;

    wfd = fileno(stdout);
        if (fstat(wfd, &sbuf)) {
            err(1, "stdout");
        }

        bsize = sbuf.st_blksize;
        buf = malloc(bsize);

        if(buf == NULL) {
            err(1, 0);
        }

    ssize_t nr, nw;
    int offset = 0;

    nr = read(rfd, buf, bsize);
    while(nr > 0) {
        for (offset = 0; nr > 0; nr -= nw, offset += nw) {
            nw = write(wfd, buf+offset, nr);
            if (nw < 0) {
                err(1, "stdout");
            }
        }
        nr = read(rfd, buf, bsize);
    }
    free(buf);
}
```

Bibliography

[1] M. Sobell, *Practical Guide to Ubuntu Linux (Versions 8.10 and 8.04)*. Pearson Education, 2008.

[2] D. Spinellis, "unix-history-repo." `https://github.com/dspinellis/unix-history-repo/blob/BSD-Release/usr/src/bin/cat/cat.c`, 1995.

[3] A. Allain, "Using valgrind to find memory leaks and invalid memory use."

[4] Srijan, "Pointers in c explained – they're not as difficult as you think."

[5] *ERR(3) Linux Programmer's Manual*, July 2020.

[6] *FERROR(3) Linux Programmer's Manual*, March 2019.

[7] *STRCMP(3) Linux Programmer's Manual*, April 2020.

[8] *OPEN(2) Linux Programmer's Manual*, June 2020.

[9] *FSTAT(3P) POSIX Programmer's Manual*, March 2013.

[10] *MALLOC(3) Linux Programmer's Manual*, June 2020.

[11] *CLOSE(2) Linux Programmer's Manual*, June 2020.

[12] *unistd.h(0P) POSIX Programmer's Manual*, June 2013.

[13] *READ(2) Linux Programmer's Manual*, February 2018.

[14] *WRITE(2) Linux Programmer's Manual*, October 2019.