System Calls and Glibc

Release 1.0

Rishi Agrawal <rishi.b.agrawal@gmail.com>

Apr 28, 2017

CONTENTS:

CHAPTER

ONE

INTRODUCTION

In this white paper we will see how your code interacts with the glibc library and then the system calls in order to get some work done from the computer.

We will go deep into the code and see how it is all organized. How system calls are called from the user space programs. How arguements are passed and how is return value accessed.

We will see the code, we will see the same thing using debugger and then we will write our own small `strace` to see what the `strace` actually does when it lists the paramters to you.

Acknowledgements

Most of the contents in the paper is inspired from the contents in the internet and various blogs.

I wanted to understand the whole process of the glibc and system calls and while doing that - I just documented the whole thing.

Wherever possilble I have given links of the reference point.

CHAPTER TWO

SYSTEM ARCHITECTURE

Todo

Write about System archtecture - little bit about how a process links with the libraries etc.

Shared Library

Todo

Write about Shared library.

Static Library

Todo

Write about Static Library.

System Calls

System calls are API's which the Kernel Provides to the user space applications. The system calls pass some arguements to the kernel space and the kernel acts accordingly on the arguements.

For example: `open()` system call - opens a file so that further read and write operations can be done on the file. The return value of the `open` system call is a `file descriptor` or an `error status`. Sucessful return value allows the user space applications to use the `file descriptor` for further reads and writes.

System calls get executed in the kernel space. Kernel space runs in an elevated prviledge mode. There is a shift of the priviledge modes whenever a system call is called and hence its a bad idea to call system calls without considering the time taken to switch to the elevated priviledge mode.

For example - lets say that you want to copy a file. One way of copying the file is to read each character of the file and for every character read you write the character to another file. This will call two system calls for every character you read and write. As this is expensive in terms of time its a bad design.

Let us see a small demostration of this.

```c
/*
 * In this code we will open the /etc/passwd file and copy the file 1000 times
 * to the output file. We will copy it 1000 times so that we have a good amount
 * data to run our test on.
 */

#include <stdlib.h>
#include <fcntl.h>
#include <stdio.h>
#include <unistd.h>

int main ()
{
    char *src_file = "src_file";
    char *dest_file = "copied_file.txt";

    int dest_fd, src_fd, read_byte, write_byte;
    char read_buf[1];

    dest_fd = open (dest_file, O_WRONLY|O_CREAT);

    if (dest_fd < 0) {
        fprintf (stderr, "Error Opening the destination file.");
        exit(1);
    } else {
        fprintf (stderr, "Successfully opened the destination file..");
    }

    src_fd = open (src_file, O_RDONLY);

    if (src_fd < 0) {
        fprintf (stderr, "Error Opening the source file.");
        exit(1);
    } else {
        fprintf (stderr, "Successfully opened the source file.");
    }


    /*
     * We will start the copy process byte by byte
     */

    while (1) {
        read_byte = read (src_fd, read_buf, 1);
        if (read_byte == 0) {
            fprintf(stdout, "Reached the EOF for src file");
            break;
        }
        write_byte = write (dest_fd, read_buf, 1);

        if (write_byte < 0) {
            perror ("Error Writing File");
            exit (1);
        }
    }

    close(src_fd);
    close(dest_fd);
```

```

    return 0;
}
```

What should instead be done here is that you read a block (set of characters) and then write that block into another file. This will reduce the number of the system calls and thus increase the overall performance of the file copy program.

```
/*
 * In this code we will open the /etc/passwd file and copy the file 1000 times
 * to the output file. We will copy it 1000 times so that we have a good amount
 * data to run our test on.
 */

#include <stdlib.h>
#include <fcntl.h>
#include <stdio.h>
#include <unistd.h>
#include <errno.h>

#define BLOCK_SIZE 4096

int main ()
{
    char *src_file = "src_file";
    char *dest_file = "copied_file.txt";

    int dest_fd, src_fd, read_byte, write_byte;
    char read_buf[BLOCK_SIZE];

    dest_fd = open (dest_file, O_WRONLY|O_CREAT, S_IRWXU|S_IRWXG|S_IROTH);

    if (dest_fd < 0) {
        perror ("\nError opening the destination file");
        exit(1);
    } else {
        fprintf (stderr, "\nSuccessfully opened the destination file..");
    }

    src_fd = open (src_file, O_RDONLY);

    if (src_fd < 0) {
        perror ("\nError opening the source file");
        exit(1);
    } else {
        fprintf (stderr, "Successfully opened the source file.");
    }


    /*
     * We will start the copy process byte by byte
     */

    while (1) {
        read_byte = read (src_fd, read_buf, BLOCK_SIZE);
        if (read_byte == 0) {
            fprintf(stdout, "Reached the EOF for src file");
            break;
        }
```

```c
        write_byte = write (dest_fd, read_buf, BLOCK_SIZE);
        if (write_byte < 0) {
            perror ("Erroo writing file");
            exit(1);
        }
    }

    close(src_fd);
    close(dest_fd);

    return 0;
}
```

```makefile
all:
	gcc -o slow_write slow_write.c -Wall
	gcc -o fast_write fast_write.c -Wall

run:
	time slow_write
	time fast_write

clean:
	rm src_file slow_write fast_write copied_file.txt

setup:
	for i in `seq 1 1000`; do cat /etc/passwd >> src_file; done
```

References

CHAPTER

THREE

WHAT IS GLIBC

`glibc` is a library which has a lot of functions pre-written for you so that you do not have to write the code again and again. Also it statndardizes the way you should be writing your code. It warps a lot of system specific details and all you need to know is to how to call the particular function, and what to be expected from the function and what are the return values the function will give you. It is the `GNU Version of Standard C Library`. All the functions supported in `Standard C Library` can be found there + some added by the `GNU`

Todo

Give an example of a function in Standard C and Not in GNU LibC

Todo

Give an example of a function in GLIBC and not in Standard C.

For example: Let us say that we have to find the length of a string. Now this is quite a small code to write and we can write the whole thing ourself, but it is a function which is used a lot of times. So the library gives you an implementation of this. As that function is present in the library you can safely assume that the function will work fine because of millions of people have used it and tested it.

You can add your own code to the library and modify the functions to suit your need.

For the sake of understanding it better we will now go into the code of the library function and see if its similar to our code.

Also we will make some changes to the code so that it stops working incorrectly and then use it in our programs. This exercise is just a demostration of the following.

- We can read the code of glibc
- We can compile the code of glibc ourselves and use the newly compiled library
- We can change the code of glibc
- We can use the changed code of glibc

Download, Extract and walk through `glibc`

Download

The source code of glibc is available at `https://ftp.gnu.org/gnu/libc/`. You can sort the list using `Last Modified` to get the latest tar package.

From the page I got the link as `https://ftp.gnu.org/gnu/libc/glibc-2.24.tar.xz`.

- Let us download this source, see the following snippet for the exact commands.

```
$ wget https://ftp.gnu.org/gnu/libc/glibc-2.24.tar.xz
--2017-01-29 07:50:02--  https://ftp.gnu.org/gnu/libc/glibc-2.24.tar.xz
Resolving ftp.gnu.org (ftp.gnu.org)... 208.118.235.20, 2001:4830:134:3::b
Connecting to ftp.gnu.org (ftp.gnu.org)|208.118.235.20|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 13554048 (13M) [application/x-tar]
Saving to: 'glibc-2.24.tar.xz'

glibc-2.24.tar.xz                    100%[==>]  12.93M   709KB/s    in 21s

2017-01-29 07:50:26 (622 KB/s) - 'glibc-2.24.tar.xz' saved [13554048/13554048]
```

Extract the code

- The downloaded code is a compressed tar file. We need to extract it.

```
rishi@rishi-VirtualBox:~$ tar -xf glibc-2.24.tar.xz
rishi@rishi-VirtualBox:~$ cd glibc-2.24/
rishi@rishi-VirtualBox:~/glibc-2.24$ ls
abi-tags      ChangeLog.3                        ChangeLog.old-ports-mips
aclocal.m4    ChangeLog.4                        ChangeLog.old-ports-powerpc
argp          ChangeLog.5                        ChangeLog.old-ports-tile
assert        ChangeLog.6                        config.h.in
benchtests    ChangeLog.7                        config.make.in
bits          ChangeLog.8                        configure
BUGS          ChangeLog.9                        configure.ac
catgets       ChangeLog.old-ports                conform
ChangeLog     ChangeLog.old-ports-aarch64        CONFORMANCE
ChangeLog.1   ChangeLog.old-ports-aix            COPYING
ChangeLog.10  ChangeLog.old-ports-alpha          COPYING.LIB
ChangeLog.11  ChangeLog.old-ports-am33           cppflags-iterator.mk
ChangeLog.12  ChangeLog.old-ports-arm            crypt
ChangeLog.13  ChangeLog.old-ports-cris           csu
ChangeLog.14  ChangeLog.old-ports-hppa           ctype
ChangeLog.15  ChangeLog.old-ports-ia64           debug
ChangeLog.16  ChangeLog.old-ports-linux-generic  dirent
ChangeLog.17  ChangeLog.old-ports-m68k           dlfcn
ChangeLog.2   ChangeLog.old-ports-microblaze     elf
extra-lib.mk      LICENSES     nscd            stdio-common
extra-modules.mk  locale       nss             stdlib
gen-locales.mk    localedata   o-iterator.mk   streams
gmon              login        po              string
gnulib            mach         posix           sunrpc
grp               Makeconfig   PROJECTS        sysdeps
gshadow           Makefile     pwd             sysvipc
hesiod            Makefile.in  README          termios
hurd              Makerules    resolv          test-skeleton
iconv             malloc       resource        time
iconvdata         manual       rt              timezone
include           math         Rules           version.h
inet              mathvec      scripts         wcsmbs
INSTALL           misc         setjmp          wctype
intl              NAMESPACE    shadow          WUR-REPORT
```

```
io                 NEWS          shlib-versions
libc-abis          nis           signal
libidn             nptl          socket
libio              nptl_db       soft-fp
```

Some string related code is here

```
rishi@rishi-VirtualBox:~/glibc-2.24$ ls string/str*
string/stratcliff.c    string/strcmp.c     string/strerror_l.c      string/strncase_l.
↪c  string/strrchr.c    string/str-two-way.h
string/strcasecmp.c    string/strcoll.c    string/strfry.c          string/strncat.c ␣
↪   string/strsep.c     string/strverscmp.c
string/strcasecmp_l.c  string/strcoll_l.c  string/string.h          string/strncmp.c ␣
↪   string/strsignal.c  string/strxfrm.c
string/strcasestr.c    string/strcpy.c     string/string-inlines.c  string/strncpy.c ␣
↪   string/strspn.c     string/strxfrm_l.c
string/strcat.c        string/strcspn.c    string/strings.h         string/strndup.c ␣
↪   string/strstr.c
string/strchr.c        string/strdup.c     string/strlen.c          string/strnlen.c ␣
↪   string/strtok.c
string/strchrnul.c     string/strerror.c   string/strncase.c        string/strpbrk.c ␣
↪   string/strtok_r.c
```

Some math related code is here

```
$ ls math/w_*
math/w_acos.c    math/w_atanhf.c  math/w_fmodf.c   math/w_j1l.c              math/w_
↪lgammal_r.c     math/w_logf.c        math/w_scalblnl.c
math/w_acosf.c   math/w_atanhl.c  math/w_fmodl.c   math/w_jn.c               math/w_
↪lgamma_main.c   math/w_logl.c        math/w_sinh.c
math/w_acosh.c   math/w_cosh.c    math/w_hypot.c   math/w_jnf.c              math/w_
↪lgamma_r.c      math/w_pow.c         math/w_sinhf.c
math/w_acoshf.c  math/w_coshf.c   math/w_hypotf.c  math/w_jnl.c              math/w_
↪log10.c         math/w_powf.c        math/w_sinhl.c
math/w_acoshl.c  math/w_coshl.c   math/w_hypotl.c  math/w_lgamma.c           math/w_
↪log10f.c        math/w_powl.c        math/w_sqrt.c
math/w_acosl.c   math/w_exp10.c   math/w_ilogb.c   math/w_lgamma_compat.c    math/w_
↪log10l.c        math/w_remainder.c   math/w_sqrtf.c
math/w_asin.c    math/w_exp10f.c  math/w_ilogbf.c  math/w_lgamma_compatf.c   math/w_
↪log1p.c         math/w_remainderf.c  math/w_sqrtl.c
math/w_asinf.c   math/w_exp10l.c  math/w_ilogbl.c  math/w_lgamma_compatl.c   math/w_
↪log1pf.c        math/w_remainderl.c  math/w_tgamma.c
math/w_asinl.c   math/w_exp2.c    math/w_j0.c      math/w_lgammaf.c          math/w_
↪log1pl.c        math/w_scalb.c       math/w_tgammaf.c
math/w_atan2.c   math/w_exp2f.c   math/w_j0f.c     math/w_lgammaf_main.c     math/w_
↪log2.c          math/w_scalbf.c      math/w_tgammal.c
math/w_atan2f.c  math/w_exp2l.c   math/w_j0l.c     math/w_lgammaf_r.c        math/w_
↪log2f.c         math/w_scalbl.c
math/w_atan2l.c  math/w_expl.c    math/w_j1.c      math/w_lgammal.c          math/w_
↪log2l.c         math/w_scalbln.c
math/w_atanh.c   math/w_fmod.c    math/w_j1f.c     math/w_lgammal_main.c     math/w_
↪log.c           math/w_scalblnf.c
```

The header files for the library is here.

```
$ ls include/
aio.h        ctype.h      fenv.h          grp-merge.h        link.h      netinet      ␣
↪resolv.h            spawn.h            syscall.h   utmp.h
```

```
aliases.h   des.h       fmtmsg.h        gshadow.h         list.h      nl_types.h ␣
↪rounding-mode.h  stab.h            sysexits.h  values.h
alloca.h    dirent.h    fnmatch.h       iconv.h           locale.h    nss.h      ␣
↪rpc              stackinfo.h       syslog.h    wchar.h
argp.h      dlfcn.h     fpu_control.h   ifaddrs.h         malloc.h    nsswitch.h ␣
↪rpcsvc           stap-probe.h      tar.h       wctype.h
argz.h      elf.h       ftw.h           ifunc-impl-list.h math.h      obstack.h  ␣
↪sched.h          stdc-predef.h     termios.h   wordexp.h
arpa        endian.h    gconv.h         inline-hashtab.h  mcheck.h    poll.h     ␣
↪scratch_buffer.h stdio_ext.h       tgmath.h    xlocale.h
assert.h    envz.h      getopt.h        langinfo.h        memory.h    printf.h   ␣
↪search.h         stdio.h           time.h
atomic.h    err.h       getopt_int.h    libc-internal.h   mntent.h    programs   ␣
↪set-hooks.h      stdlib.h          ttyent.h
bits        errno.h     glob.h          libc-symbols.h    monetary.h  protocols  ␣
↪setjmp.h         string.h          uchar.h
byteswap.h  error.h     gmp.h           libgen.h          mqueue.h    pthread.h  ␣
↪sgtty.h          strings.h         ucontext.h
caller.h    execinfo.h  gnu             libintl.h         net         pty.h      ␣
↪shadow.h         stropts.h         ulimit.h
complex.h   fcntl.h     gnu-versions.h  libio.h           netdb.h     pwd.h      ␣
↪shlib-compat.h   stubs-prologue.h  unistd.h
cpio.h      features.h  grp.h           limits.h          netgroup.h  regex.h    ␣
↪signal.h         sys               utime.h
```

Walkthrough `strlen`

Todo

write this section.

Walkthrough `div`

Todo

write this section.

Walkthrough `open`

Todo

write this section.

Compiling the code of glibc

Generally compling code on Linux system involves two stages

1. Configuring - running `configure` with right options.
2. Compiling - running `make` with right options.
3. Install - running `make install`.

Configuring

We will get into the glibc-2.24 source directory and run the `configure` script. I have intentionally shown the mistakes which happened so that you also understand the small things which needs to be taken care while configuring and compling.

```
rishi@rishi-VirtualBox:~/glibc-2.24$ ./configure
checking build system type... x86_64-pc-linux-gnu
checking host system type... x86_64-pc-linux-gnu
checking for gcc... gcc
checking for suffix of object files... o
checking whether we are using the GNU C compiler... yes
checking whether gcc accepts -g... yes
checking for readelf... readelf
checking for g++... g++
checking whether we are using the GNU C++ compiler... yes
checking whether g++ accepts -g... yes
checking whether g++ can link programs... yes
configure: error: you must configure in a separate build directory
```

We got an error that we should use a separate directory for running `configure`

```
rishi@rishi-VirtualBox:~/glibc-2.24$ mkdir ../build_glibc

rishi@rishi-VirtualBox:~/glibc-2.24$ cd ../build_glibc/
```

Let us now run the configure command.

```
rishi@rishi-VirtualBox:~/build_glibc$ ../glibc-2.24/configure
checking build system type... x86_64-pc-linux-gnu
checking host system type... x86_64-pc-linux-gnu
checking for gcc... gcc
checking for suffix of object files... o
checking version of sed... 4.2.2, ok
checking for gawk... no

>>>>>>>>>>>>>>>>>SNIP<<<<<<<<<<<<<<<<<<<<<<

checking if gcc is sufficient to build libc... yes
checking for nm... nm
configure: error:
*** These critical programs are missing or too old: gawk
*** Check the INSTALL file for required versions.
```

The configure step gave errors - let us install `gawk` now.

```
rishi@rishi-VirtualBox:~/build_glibc$ sudo apt-get install gawk
[sudo] password for rishi:
Reading package lists... Done
Building dependency tree
Reading state information... Done
The following additional packages will be installed:
```

```
libsigsegv2
Suggested packages:
gawk-doc
The following NEW packages will be installed:
gawk libsigsegv2

>>>>>>>>>>>>>SNIP<<<<<<<<<<<<<<

Setting up gawk (1:4.1.3+dfsg-0.1) ...
```

Check if the command is present.

```
rishi@rishi-office:~/mydev/publications/system_calls$ which gawk
/usr/bin/gawk
```

Let us run configure again

```
rishi@rishi-VirtualBox:~/build_glibc$ ../glibc-2.24/configure
checking build system type... x86_64-pc-linux-gnu
checking host system type... x86_64-pc-linux-gnu
checking for gcc... gcc
checking for suffix of object files... o
checking whether we are using the GNU C compiler... yes

>>>>>>>>>>SNIP<<<<<<<<<<<<<<<<<<<<<<

running configure fragment for sysdeps/unix/sysv/linux/x86_64
running configure fragment for sysdeps/unix/sysv/linux
checking installed Linux kernel header files... 3.2.0 or later
checking for kernel header at least 2.6.32... ok
*** On GNU/Linux systems the GNU C Library should not be installed into
*** /usr/local since this might make your system totally unusable.
*** We strongly advise to use a different prefix.  For details read the FAQ.
*** If you really mean to do this, run configure again using the extra
*** parameter `--disable-sanity-checks`.
```

- Configure does not want to overwrite the default library and hence we need to give another directory to install the library.
- Let us make a directory and run the configure script.

```
rishi@rishi-VirtualBox:~/build_glibc$ mkdir ../install_glibc
rishi@rishi-VirtualBox:~/build_glibc$ ../glibc-2.24/configure --prefix=/home/rishi/
↪install_glibc/
checking build system type... x86_64-pc-linux-gnu
checking host system type... x86_64-pc-linux-gnu
checking for gcc... gcc
checking for suffix of object files... o
configure: creating ./config.status

>>>>>>>SNIP<<<<<<<<<<<<

config.status: creating config.make
config.status: creating Makefile
config.status: creating config.h
config.status: executing default commands
```

- Configure completed

```
rishi@rishi-VirtualBox:~/build_glibc$ ls
bits  config.h  config.log  config.make  config.status  Makefile
```

- Let us run the `make` command now.

```
rishi@rishi-VirtualBox:~/build_glibc$ make -j 16
make -r PARALLELMFLAGS="" -C ../glibc-2.24 objdir=`pwd` all
make[1]: Entering directory '/home/rishi/glibc-2.24'
LC_ALL=C gawk -f scripts/sysd-rules.awk > /home/rishi/build_glibc/sysd-rulesT \



rishi@rishi-VirtualBox:~/build_glibc$ ls
bits  config.h  config.log  config.make  config.status  Makefile
rishi@rishi-VirtualBox:~/build_glibc$
rishi@rishi-VirtualBox:~/build_glibc$
rishi@rishi-VirtualBox:~/build_glibc$



rishi@rishi-VirtualBox:~/build_glibc$ make -j 16
make -r PARALLELMFLAGS="" -C ../glibc-2.24 objdir=`pwd` all
make[1]: Entering directory '/home/rishi/glibc-2.24'
LC_ALL=C gawk -f scripts/sysd-rules.awk > /home/rishi/build_glibc/sysd-rulesT \

      >>>>>>>>>>>>>>>>>>>>>SNIP<<<<<<<<<<<<<<<<<<<
gcc -nostdlib -nostartfiles -o /home/rishi/build_glibc/elf/pldd    -Wl,-z,combreloc -
↪Wl,-z,relro -Wl,--hash-style=both /home/rishi/build_glibc/csu/crt1.o /home/rishi/
↪build_glibc/csu/crti.o `gcc  --print-file-name=crtbegin.o` /home/rishi/build_glibc/
↪elf/pldd.o /home/rishi/build_glibc/elf/xmalloc.o  -Wl,-dynamic-linker=/home/rishi/
↪install_glibc/lib/ld-linux-x86-64.so.2 -Wl,-rpath-link=/home/rishi/build_glibc:/
↪home/rishi/build_glibc/math:/home/rishi/build_glibc/elf:/home/rishi/build_glibc/
↪dlfcn:/home/rishi/build_glibc/nss:/home/rishi/build_glibc/nis:/home/rishi/build_
↪glibc/rt:/home/rishi/build_glibc/resolv:/home/rishi/build_glibc/crypt:/home/rishi/
↪build_glibc/mathvec:/home/rishi/build_glibc/nptl /home/rishi/build_glibc/libc.so.6 /
↪home/rishi/build_glibc/libc_nonshared.a -Wl,--as-needed /home/rishi/build_glibc/elf/
↪ld.so -Wl,--no-as-needed -lgcc  `gcc  --print-file-name=crtend.o` /home/rishi/build_
↪glibc/csu/crtn.o
make[2]: Leaving directory '/home/rishi/glibc-2.24/elf'
make[1]: Leaving directory '/home/rishi/glibc-2.24'
```

- Make runs successfully.
- Let us check the `install_glibc` directory. It has nothing in it.

```
$ ls ../install_glibc/
```

- Let us run the `make install` command.

```
$ make install
LC_ALL=C; export LC_ALL; \
make -r PARALLELMFLAGS="" -C ../glibc-2.24 objdir=`pwd` install
make[1]: Entering directory '/home/rishi/glibc-2.24'
make  subdir=csu -C csu ..=../ subdir_lib
make[2]: Entering directory '/home/rishi/glibc-2.24/csu'
make[2]: Leaving directory '/home/rishi/glibc-

>>>>>>>>>>>SNIP<<<<<<<<<<<<<<<
```

```
-f /home/rishi/build_glibc/elf/symlink.list
test ! -x /home/rishi/build_glibc/elf/ldconfig || LC_ALL=C \
  /home/rishi/build_glibc/elf/ldconfig  \
           /home/rishi/install_glibc/lib /home/rishi/install_glibc/lib
           /home/rishi/build_glibc/elf/ldconfig: Warning: ignoring configuration␣
↪file that cannot be opened: /home/rishi/install_glibc/etc/ld.so.conf: No such file␣
↪or directory
           make[1]: Leaving directory '/home/rishi/glibc-2.24'
```

- Let us now check the `install_glibc` directory. It has the required files of the new compiled library.

```
rishi@rishi-VirtualBox:~/build_glibc$ ls ../install_glibc/
bin  etc  include  lib  libexec  sbin
```

Using the new library

Let us now use the above library to link and run our code. We will add a new function to the glibc, change the behaviour of a function in glibc and use the new function and call the changed function.

This will give us a good understanding of how to compile and link with the new library.

Here is the code for adding some changes to the glibc code. See the file `glibc-2.24/stdlib/div.c` and `glibc-2.24/include/stdlib.h`.

Here is the diff

glibc-2.24/stdlib/div.c

- Here we have added a function `my_div` which just returns -1 on invokation and have changed the way the function div behaves. Now when we will pass 99 and 99 to div it will return 100 and 100. Read the default behaviour in the man pages.

```
$ diff glibc-2.24/stdlib/div.c temp/glibc-2.24/stdlib/div.c
51d50
< #include <stdio.h>
59,64d57
<   if (numer == 99 && denom == 99) {
<   printf ("\nValues are 99 and 99");
<   result.quot = 100;
<   result.rem = 100;
<   return result;
<   }
69,74d61
< }
<
<
< int my_div(void) {
<   printf("\n\nCalling my_div() function.");
<   return -1;
```

- Here is the declaration of the new function.

glibc-2.24/stdlib/stdlib.h

```
$ diff glibc-2.24/stdlib/stdlib.h temp/glibc-2.24/stdlib/stdlib.h
753,754d752
<
< extern int my_div(void);
```

- Here is the code which calls the functions.

```
#include <stdio.h>
#include <stdlib.h>


int main () {

	div_t result = div(99, 99);
	int x = my_div();

	printf ("\n\nQuotient %d Remainder %d", result.quot, result.rem);
	return 0;
}
```

- Here is the `Makefile` which will be used to compile the program.

```
TARGET = div
OBJ = $(TARGET).o
SRC = $(TARGET).c
CC = gcc
CFLAGS = -g
LDFLAGS = -nostdlib -nostartfiles -static
GLIBCDIR = /home/rishi/install_glibc/lib/
STARTFILES = $(GLIBCDIR)/crt1.o $(GLIBCDIR)/crti.o `gcc --print-file-name=crtbegin.o`
ENDFILES = `gcc --print-file-name=crtend.o` $(GLIBCDIR)/crtn.o
LIBGROUP = -Wl,--start-group $(GLIBCDIR)/libc.a -lgcc -lgcc_eh -Wl,--end-group
INCDIR = /home/rishi/install_glibc/include

$(TARGET): $(OBJ)
	$(CC) $(LDFLAGS) -o $@ $(STARTFILES) $^ $(LIBGROUP) $(ENDFILES)

$(OBJ): $(SRC)
	$(CC) $(CFLAGS) -c $^ -I `gcc --print-file-name=include` -I $(INCDIR)

clean:
	rm -f *.o *.~ $(TARGET)
	rm test.c.*
	rm a.out


# https://stackoverflow.com/questions/10763394/how-to-build-a-c-program-using-a-custom-version-of-glibc-and-static-linking/10772056#10772056
```

- Run the `make` command.

```
$ make
gcc -g -c div.c -I `gcc --print-file-name=include` -I /home/rishi/install_glibc/include
gcc -nostdlib -nostartfiles -static -o div /home/rishi/install_glibc/lib//crt1.o /home/rishi/install_glibc/lib//crti.o `gcc --print-file-name=crtbegin.o` div.o -Wl,--start-group /home/rishi/install_glibc/lib//libc.a -lgcc -lgcc_eh -Wl,--end-group `gcc --print-file-name=crtend.o` /home/rishi/install_glibc/lib//crtn.o
```

- Run the statically linked code

```
$ ./div

Values are 99 and 99

Calling my_div() function.

Quotient 100 Remainder 100
```

- See the size of the staticically linked code. The huge size is due to static linking. We will now link it dynamically and then see the size.

```
$ ls -lh div
-rwxrwxr-x 1 rishi rishi 3.3M Jan 29 20:00 div
```

- Run the dynamically linked code.

Todo

Link it dynamically.

Error: Unable to do it dynamically.

- See the sizes of the files

```
rishi@rishi-VirtualBox:~/test_code$ ls -l dynamic-test static-test
-rwxrwxr-x  1 rishi rishi   8600 Jan 29 12:13 dynamic-test
-rwxrwxr-x  1 rishi rishi 909048 Jan 29 12:13 static-test
```

Todo

Link it dynamically.

- Check the file type of the executables.

Todo

correct the following.

```
rishi@rishi-VirtualBox:~/test_code$ file static-test
static-test: ELF 64-bit LSB executable, x86-64, version 1 (GNU/Linux), statically
↪linked, for GNU/Linux 2.6.32,
↪BuildID[sha1]=866f4fe367915159ae62cc80a0ae614059d67153, not stripped
```

```
rishi@rishi-VirtualBox:~/test_code$ file dynamic-test
dynamic-test: ELF 64-bit LSB executable, x86-64, version 1 (SYSV), dynamically linked,
↪ interpreter /home/rishi/install_glibc/lib/ld-linux-x86-64.so.2, for GNU/Linux 2.6.
↪32, BuildID[sha1]=c0f8ac9a77a879e6adc855333d6bc88c5078ffd3, not stripped
```

CHAPTER

FOUR

HOW IS A SYSTEM CALL CALLED ON X86_64 ARCHITECTURE FROM USER SPACE

There are three parts to calling a system call.

1. Setting up the arguements to be passed to the kernel space.
2. Call the system call using the `syscall` assembly instruction.
3. Get back the return value.

In the sections below we will see each of them in detail.

Setting Up Arguements

Note: The following text is copied verbatim from the document `System V Application Binary Interface AMD64 Architecture Processor 57 Supplement Draft Version 0.99.6`, Section `AMD64 Linux Kernel Conventions`

Todo

Check if we are infringing copyright here.

```
Calling Conventions

The Linux AMD64 kernel uses internally the same calling conventions as user-
level applications (see section 3.2.3 for details). User-level applications that like
to call system calls should use the functions from the C library. The interface
between the C library and the Linux kernel is the same as for the user-level appli-
cations with the following differences:

1. User-level applications use as integer registers for passing the sequence
%rdi, %rsi, %rdx, %rcx, %r8 and %r9. The kernel interface uses %rdi,
%rsi, %rdx, %r10, %r8 and %r9.
2. A system-call is done via the syscall instruction. The kernel destroys
registers %rcx and %r11.
3. The number of the syscall has to be passed in register %rax.
4. System-calls are limited to six arguments, no argument is passed directly on
the stack.
5. Returning from the syscall, register %rax contains the result of the
system-call. A value in the range between -4095 and -1 indicates an error,
```

```
it is -errno.
6. Only values of class INTEGER or class MEMORY are passed to the kernel.
```

See the `System V Application Binary Interface AMD64 Architecture Processor Supplement Draft Version 0.99.6.` Section `AMD64 Linux Kernel Conventions` for the details.

Reiterating The Above Again

Hence when we have called any function in user space we will have the following state of the registers when we are in the called function.

Table 4.1: "Arguements Passing In Linux"

Register	Argument User Space	Argument Kernel Space
%rax	Not Used	System Call Number
%rdi	Arguement 1	Arguement 1
%rsi	Arguement 2	Arguement 2
%rdx	Arguement 3	Arguement 3
%r10	Not Used	Arguement 4
%r8	Arguement 5	Arguement 5
%r9	Arguement 6	Arguement 6
%rcx	Arguement 4	Destroyed
%r11	Not Used	Destroyed

Note: This table summarizes the differences when a function call is made in the user space, and when a system call is made. This will be more clear in coming texts. Right now make a note of it

Passing arguements

1. Arguements are passed in the registers. The called function then uses the register to get the arguements.
2. The arguements are passed in the following sequence `%rdi, %rsi, %rdx, %r10, %r8 and %r9.`
3. Number of arguements are limited to `six`, no arguements will be passed on the stack.
4. Only values of class INTEGER or class MEMORY are passed to the kernel.
5. Class `INTEGER` This class consists of integral types that fit into one of the general purpose registers.
6. Class `MEMORY` This class consists of types that will be passed and returned in mem- ory via the stack. These will mostly be strings or memory buffer. For example in `write()` system call, the first parameter is `fd` which is of class `INTEGER` while the second argument is the `buffer` which has the data to be written in the file, the class will be `MEMORY` over here. The third parameter which is the count - again has the class as `INTEGER`.

Note: The above information is sourced from AMD64 Architecture Processor Supplement Draft Version 0.99.6

Calling the System Call

1. A system-call is done via the `syscall` assembly instruction. The kernel destroys registers `%rcx` and `%r11`.

2. The number of the syscall has to be passed in register `%rax`.

Retrieving the Return Value

1. Returning from the `syscall`, register `%rax` contains the result of the system-call. A value in the range between `-4095` and `-1` indicates an error, it is `-errno`.

CHAPTER

FIVE

SETTING UP ARGUMENTS

Introduction

In the above section we have see the theory part related to passing arguements to the system call interface of the kernel. Now we will see some assignements related to it.

We will see if how the above concepts being implemented in actual code. By this time we know that the we need to link the code to the `glibc` in order to use the system calls. In the following sections we will see this

We will do it in three different ways.

1. Walk through `open` system call in `glibc` library.
2. See it using debugger.
3. Use `ptrace` system call and see the state of the registers. Code related to this can be found in the appendix.

Walk through `open` system call in `glibc`

In this assignment we will download the source code of `glibc` and then walk through the code to find out where exactly the code is calling the `syscall` assembly instruction and where is it moving the arguements to the registers.

We will do this with `open` system call and `write` system calls.

Note: If you have not understood above concepts. Do not worry, keep reading on and then re-read the whole thing once more.

How `open()` system call is called using `glibc`

1. All the above theory should match with the code which is written in `glibc`.
2. We will now read the code in the `glibc` to find out if the theory matches what is written in the code.
3. We will also do some assignments to get a better understanding of the above theory.
4. Now the question is `open` system call - how will it turn to a `syscall` instruction.
5. Now we need to find out what happens to the `open` system call when compiled.
6. File where sys call numbers are mentioned `/usr/include/x86_64-linux-gnu/asm/unistd_64.h`
7. File where `SYS_write` maps to `NR_Write` `/usr/include/x86_64-linux-gnu/bits/syscall.h`

8. From the objdump we saw that `__libc_open` was called. This called `__open_nocancel` and it had a `syscall` instruction. It means that the path to the kernel is in this function.
9. See the `object dump`, offset `433e0e`. This dump is taken from a code where we had a `open` system call and was compiled.

```
0000000000433e09 <_open_nocancel>:
433e09:   b8 02 00 00 00          mov    $0x2,%eax

433e0e:   0f 05                   syscall        <<<<<<<<<<<<<<<<<<

433e10:   48 3d 01 f0 ff ff       cmp    $0xfffffffffffff001,%rax
433e16:   0f 83 f4 46 00 00       jae    438510 <__syscall_error>
433e1c:   c3                      retq
433e1d:   48 83 ec 08             sub    $0x8,%rsp
433e21:   e8 ca 2f 00 00          callq  436df0 <__libc_enable_asynccancel>
433e26:   48 89 04 24             mov    %rax,(%rsp)
433e2a:   b8 02 00 00 00          mov    $0x2,%eax
433e2f:   0f 05                   syscall
433e31:   48 8b 3c 24             mov    (%rsp),%rdi
433e35:   48 89 c2                mov    %rax,%rdx
433e38:   e8 13 30 00 00          callq  436e50 <__libc_disable_asynccancel>
433e3d:   48 89 d0                mov    %rdx,%rax
433e40:   48 83 c4 08             add    $0x8,%rsp
433e44:   48 3d 01 f0 ff ff       cmp    $0xfffffffffffff001,%rax
433e4a:   0f 83 c0 46 00 00       jae    438510 <__syscall_error>
433e50:   c3                      retq
433e51:   66 2e 0f 1f 84 00 00    nopw   %cs:0x0(%rax,%rax,1)
433e58:   00 00 00
433e5b:   0f 1f 44 00 00          nopl   0x0(%rax,%rax,1)
```

1. Now, when in glibc-2.3 dir I started finding the code for the function `__open_nocancel` I found this
2. File is `sysdeps/unix/sysv/linux/generic/open.c`

```
int __open_nocancel (const char *file, int oflag, ...)
{
    int mode = 0;

    if (__OPEN_NEEDS_MODE (oflag))
    {
        va_list arg;
        va_start (arg, oflag);
        mode = va_arg (arg, int);
        va_end (arg);
    }

    return INLINE_SYSCALL (openat, 4, AT_FDCWD, file, oflag, mode);
}
```

1. So INLINE_SYSCALL is being called by this function. This is defined in the file `glibc-2.3/sysdeps/unix/sysv/linux/x86_64/sysdep.h`

```
# define INLINE_SYSCALL(name, nr, args...) \
    ({                                          \
     unsigned long int resultvar = INTERNAL_SYSCALL (name, , nr, args);        \
     if (__glibc_unlikely (INTERNAL_SYSCALL_ERROR_P (resultvar, )))        \
     {                                        \
     __set_errno (INTERNAL_SYSCALL_ERRNO (resultvar, ));             \
```

```
    resultvar = (unsigned long int) -1;                        \
    }                                            \
    (long int) resultvar; })
```

1. Thus it calls `INTERNAL_SYSCALL` which is defined as

```
# define INTERNAL_SYSCALL(name, err, nr, args...) \
  INTERNAL_SYSCALL_NCS (__NR_##name, err, nr, ##args)
```

1. Now let us see the `INTERNAL_SYSCALL_NCS` in the file `./sysdeps/unix/sysv/linux/x86_64/sysdep.h` here see the macro `INTERNAL_SYSCALL_NCS`. This is the exact macro which is calling the `syscall` assembly instruction. You can see the `asm` instructions in the code.

```
# define INTERNAL_SYSCALL_NCS(name, err, nr, args...) \
   ({                                           \
    unsigned long int resultvar;                          \
    LOAD_ARGS_##nr (args)                             \
    LOAD_REGS_##nr                                \
    asm volatile (                                \
        "syscall\n\t"                                 \
        : "=a" (resultvar)                                \
        : "0" (name) ASM_ARGS_##nr : "memory", REGISTERS_CLOBBERED_BY_SYSCALL);    \
    (long int) resultvar; })
```

1. Thus here we enter the kernel using the `syscall` assembly instruction.
2. Also, we need to figure out how - `open()` call went to be called as `__open_nocancel`

Todo

`open` call called `__open_nocancel`, How.

Todo

The above section is not very well written, do it.

#. We have redone the whole thing with the `write` system call in the appendix. You can see that as well to get more clarity.

How is `write` system call implemented in `glibc`

Todo

Write this part - do it in the appendix. This will make the paper better organized.

Check Arguements Using A Debugger

In the above example we saw how the code calls the `syscall` instruction to enter the kernel and call the required functionality. Write the following code and compile it with `gcc -g filename.c`

`-g` flag adds the debugging information to the execuatable.

```c
#include <fcntl.h>
#include <string.h>

int main ()
{
    char filename[] = "non_existent_file";
    int fd;
    fd = open (filename, O_CREAT|O_WRONLY);

    fd = write (fd, filename, strlen(filename));
    close (fd);
    unlink (filename);
    return 0;
}
```

- Once done, run the code in the debugger `gdb ./a.out`
- Set the breakpoint in the call on write `break write`
- According to the calling conventions the register `$rdi` should have the file descriptor. `$rdi` should have the string's address and the `$rdx` should have the length of the string.
- Using `print` command will confirm these values.

```
(gdb) b write
Breakpoint 1 at 0x400560
(gdb) r
Starting program: /home/rishi/mydev/books/crash_book/code_system_calls/01/aaa/a.out

Breakpoint 1, write () at ../sysdeps/unix/syscall-template.S:81
81  ../sysdeps/unix/syscall-template.S: No such file or directory.
(gdb) print $rdi
$1 = 3
(gdb) print (char *) $rsi
$2 = 0x7fffffffdeb0 "non_existent_file"
(gdb) print $rdx
$3 = 17
(gdb)
```

Using `ptrace` to see the variables passed

Todo

add code for this. Better to add it as a appendix.

CHAPTER

SIX

SYSTEM CALL IMPLEMENTATION IN THE USER SPACE

There are two ways system calls are being called in the user space. Both of them will eventually call the `syscall` instruction but `glibc` provides a wrapper around that instruction using a function call.

- **`glibc` library call which does the work which needs to be done before** calling the `syscall` instruction.
- `syscall` assembly instruction to enter the priviledged mode. This allows the process to move to the priviledge mode.

CHAPTER SEVEN

CALLING SYSTEM CALLS

Glibc `syscall()` interface

1. There is a library function in `glibc` named as `syscall`, you can read about it in the man pages by the command `man 2 syscall`.
2. We already have the code of `glibc` with us.
3. See the function in the file `glibc-2.23/sysdeps/unix/sysv/linux/x86_64/syscall.S`
4. On reading the code you will see that the function is moving the arguement values to the registers and then calling the assembly instruction `syscall`.
5. As `syscall` here is a user space `glibc` library function, first the arguements will be in the registers used for calling user space functions. Once this is done, as the system call is being called, the arguements will be used into the registers where the kernel wishes to find the arguments.
6. Code for `syscall(2)` library function.

Note: Remember the note above. As `syscall` is a function which we called in user space, the registers are different. We now need to pick and place the registers in a way that the system call understands it. THis is shown in the code below.

```
.text

    ENTRY (syscall)
        movq %rdi, %rax             /* Syscall number -> rax.  */
        movq %rsi, %rdi             /* Shift the arg2 to arg1 for syscalls */
        movq %rdx, %rsi             /* Shift the arg3 to arg2 for syscalls */
        movq %rcx, %rdx             /* Shift the arg4 to arg3 for syscalls */
        movq %r8, %r10              /* Shift the arg5 to arg4 for syscalls */
        movq %r9, %r8               /* Shift the arg6 ro arg5 for syscalls */
        movq 8(%rsp),%r9            /* Shift the arg7 from the stack to arg6 for␣
↪syscalls */
        syscall                     /* Do the system call.  */
        cmpq $-4095, %rax           /* Check %rax for error. %rax has the return␣
↪value  */
        jae SYSCALL_ERROR_LABEL     /* Jump to error handler if error.  */
        ret                         /* Return to caller.  */
    PSEUDO_END (syscall)
```

Todo

The above code is not getting highlighted, maybe due to the use of incorrect lexer. See this page http://pygments.org/docs/lexers/ and hightlight the above code. use code block for this.

Assembly Instruction for calling system call.

We know now that for calling a system call we just need to set the right arguements in the register and then call the `syscall` instruction.

Register `%rax` needs the `system call number`. So where are the `system call numbers` defined. Here we can see the `glibc` code to see the mapping of the number and the system call. Or you can see this in a header file in the system's include directory.

System call numbers will never change, if they do there will be a lot of porting efforts which will need to be done else a lot of applications will break.

Let us see a excerpt from the file `/usr/include/x86_64-linux-gnu/asm/unistd_64.h`

```
#define __NR_read 0
#define __NR_write 1
#define __NR_open 2
#define __NR_close 3
#define __NR_stat 4
```

Here you can see that the system calls have numbers associated with them.

Now armed with the knowledge of how to call system calls let us write some assembly code where we call a system call.

Before doing this excercise let us see the write system call a bit. In the follwing code we will write `hello world` on the screen. We will not use `printf` for this, rather we will use `2` (the standard descriptor for writing to the terminal) and `write` system call for it.

We need to do this so that we understand our assembly level program a bit better.

```
#include <fcntl.h>

int main ()
{
    write (1, "Hello World", 11);
    return 0;
}
```

You should go through the assembly code of the C file. Use command `gcc -S filename.c` This will generate the assembly file with `.s` extension. If you go through the assembly code you will see a call to `write` function. This function is defined in the `glibc`. We will see the source of `write` system call in sometime. At that time you can refer this and understand it better.

Note: When I am compiling the code I can see the assembly code only using the eax register and not rax, why?

Todo

We should explain the assembly code generated above.

Now we will do the same using the `syscall` intergface which the `glibc` provides.

```c
#include <unistd.h>
#include <sys/syscall.h>


int main ()
{
    syscall (1, 1, "Hello World", 11);
    return 0;
}
```

You should go through the assembly code of the C file. Use command `gcc -S filename.c` This will generate the assembly file with `.s` extension. If you go through the assembly code you will see a call to `syscall` function. This function is defined in the `glibc`. We will see the source of `syscall` system call in sometime. At that time you can refer this and understand it better.

Note: When I am compiling the code I can see the assembly code only using the eax register and not rax, why?

Todo

We should explain the assembly code generated above.

Now we will do the same in our assembly code.

```nasm
section .text
        global _start
        _start:                 ; ELF entry point
         ; 1 is the number for syscall write ().

    mov rax, 1
    ; 1 is the STDOUT file descriptor.

    mov rdi, 1

     ; buffer to be printed.

    mov rsi, message

    ; length of buffer

    mov rdx, [messageLen]

    ; call the syscall instruction
    syscall

    ; sys_exit
        mov rax, 60

    ; return value is 0
        mov rdi, 0

    ; call the assembly instruction
        syscall

section .data
        messageLen: dq message.end-message
```

```
        message: db 'Hello World', 10
.end:
```

Makefile for assembling the code.

```
all:
	nasm -felf64 hello.asm
	ld hello.o

clean:
	rm -rf *.o

```

CHAPTER

EIGHT

RETURN VALUES

Return Value Status in the register

Todo

add content to show the return values as well and the error codes.

Conclusion

Hence we now know the following stuff

Todo

Write the conclusion.

CHAPTER

NINE

WALK THROUGH `WRITE` SYSTEM CALL

Todo

Write this section.

CHAPTER

TEN

USING `PTRACE` SYSTEM CALL TO SEE THE VARIABLES

CHAPTER

ELEVEN

INDICES AND TABLES

- genindex
- modindex
- search