ELEC 424/553
Mobile & Embedded Systems

Lecture 7 - System Calls

http://www.cnn.com/2010/TECH/mobile/07/09/cooper.cell.phone.inventor/index.html

Google Slides Link:
https://docs.google.com/presentation/d/10X89S5DPI8txxseBbGyLh-HgPZX7X8yM_gype2_STa0/edit?usp=sharing

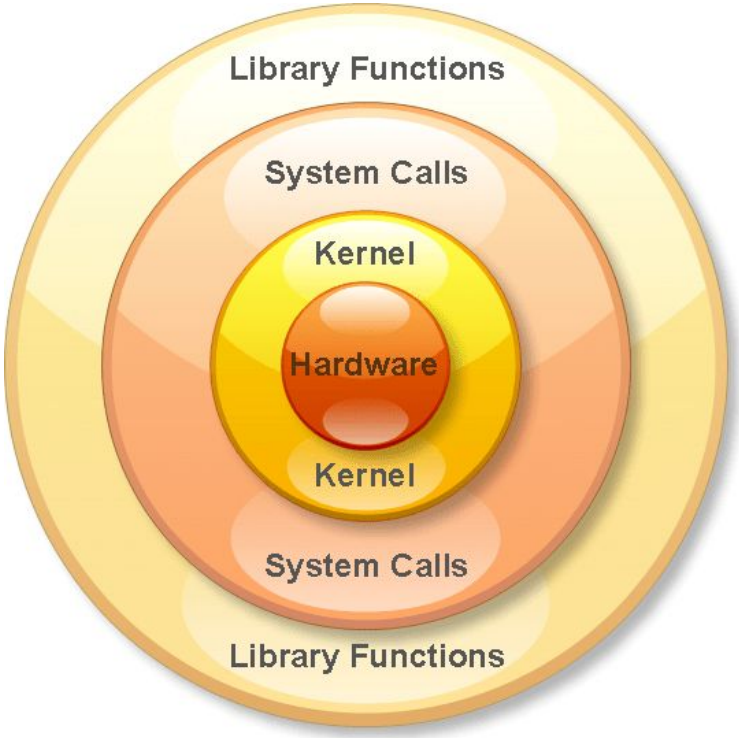
Housekeeping

ARM has hardware support for OS privilege levels similar to x86

Note from prior exercise on factorial code:

- In C, ++n is pre-increment and n++ is post-increment
- For the condition of a while loop, ++n will happen before the condition is checked and n++ will happen after the condition is checked

Soldering session, extra office hours



Balakumaran Kannan, “Anatomy of Linux system call in ARM64”. URL: https://eastrivervillage.com/Anatomy-of-Linux-system-call-in-ARM64/

Questions In The Past

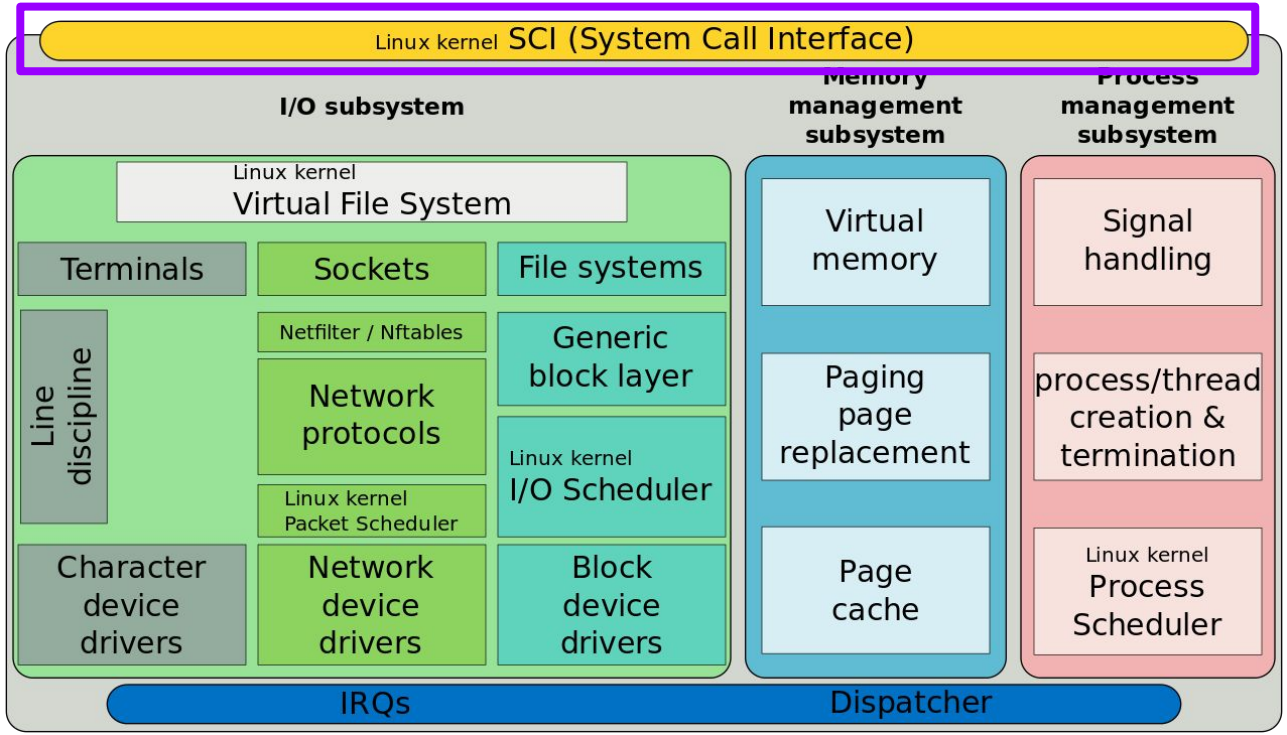
- **Circular Queue vs. O(1) scheduler**
  - Scheduling classes
- **Scheduler overhead**
  - O(log n)
  - Maybe a few μs for a context switch
  - Sources:
    - https://unix.stackexchange.com/questions/553120/linux-scheduling-overhead-in-orders-of-magnitude
    - https://eli.thegreenplace.net/2018/measuring-context-switching-and-memory-overheads-for-linux-threads/
    - https://events.static.linuxfound.org/sites/events/files/slides/lemoal-nvme-polling-vault-2017-final_0.pdf
- **Processes going from to sleep to awake**
  - One response indicates min_vruntime normalization does occur
    - https://stackoverflow.com/questions/11297285/how-can-the-linux-cfs-scheduler-prevent-a-task-with-a-very-small-vruntime-from-s

Goals of Today's Lecture

- Enable you to become a **kernel hacker**
- Learn the basics of **system calls**
  - How they're used
  - How they work
  - How to make your own

"Anonymous - Cool Hacker Guy". URL: https://i.kym-cdn.com/photos/images/original/001/109/423/cb4.jpg

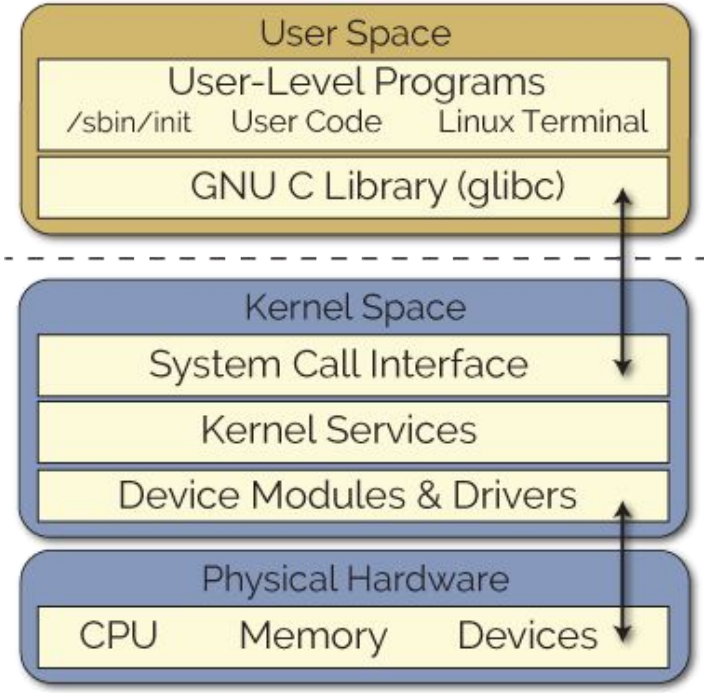
Where Are We In The Linux Kernel?



Kernel Space Must Be Protected At All Costs

- User applications have **no direct access** to kernel
- Indirect access via **system calls**
- System calls provide applications with access to **kernel services**
- Kernel acts **on behalf of application**

Rachelle Hampton, "There's Nothing Wrong With Server-Speak. In Fact, It's Polite!". Image source: SeventyFour / Thinkstock. URL: https://slate.com/human-interest/2018/02/food-service-grammar-is-perfectly-polite.html



Derek Molloy, "Writing a Linux Kernel Module — Part 1: Introduction". URL: http://derekmolloy.ie/writing-a-linux-kernel-module-part-1-introduction/#prettyPhoto

pchdtvr - A Case Study of System Calls In a Program

Home / Personal Media Networks frequal.com

pchdtvr -- record digital television on your PC

pchdtvr is a great program for recording digital television on your PC. It lets you:

1. Set up schedules
2. Record programs instantly
3. Check station signal strength
4. And much more!

http://frequal.com/pmn/pchdtvr.html

pchdtvr - A Case Study in Terrible Naming

`pchdtvr` - a good reason why techs should *never* be allowed to name their products :-)
– user10776 Dec 18 '16 at 8:04

"Can GPL licensed code be close sourced later by the author? [duplicate]". URL: https://softwareengineering.stackexchange.com/questions/98774/can-gpl-licensed-code-be-close-sourced-later-by-the-author/98778

pchdtvr - A Case Study of License Legal Issues Too

Home / Personal Media Networks frequal.com

pchdtvr -- record digital television on your PC

pchdtvr is a great program for recording digital television on your PC. It lets you:

1. Set up schedules
2. Record programs instantly
3. Check station signal strength
4. And much more!

Best of all, since pchdtvr is licensed under the GNU GPL, nobody can ever take away your right to use it, change it, or give it away!

Download

There are several versions available:

1. pchdtvr-1.0-frequal, the latest and greatest version available.
    1. New % file suffix to attach date to recordings
    2. Append instead of rename for restarted recording
2. pchdtvr-1.0
3. pchdtvr-0.9s

Last modified on 26 Jan 2008 by AO

Copyright © 2020 Andrew Oliver

http://frequal.com/pmn/pchdtvr.html

I have revoked the licensing under the GNU General Public License (herein after referred to as "the GPL") for the atscap version 1.1 codebase, all prior versions of the atscap codebase and all the various release candidates of the atscap codebase.

[...]

As sole author of both the atscap and the pchdtvr codebases, the licenses under the GPL were granted at my sole discretion and the licenses under the GPL are now hereby revoked at my sole discretion.

[...]

If you are currently using the atscap or pchdtvr packages, or any part thereof, it is in your best interest to remove the software from your system(s) and destroy all copies in your possession.

[...]

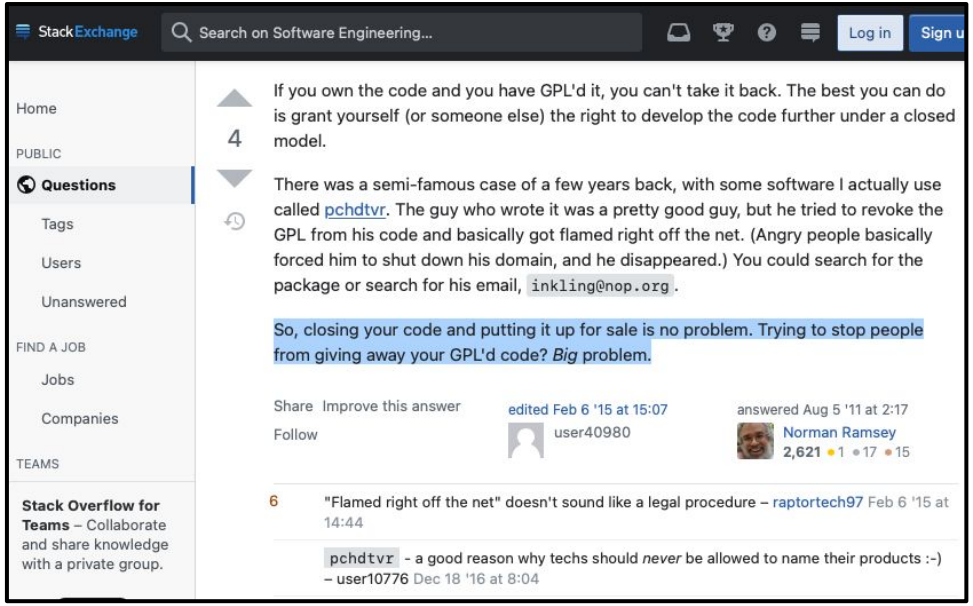
If you have incorporated the atscap or pchdtvr codebase, or any part thereof, into any of your projects, it is in your best interest to remove any and all of my code from your project(s).

https://blog.jasonantman.com/2008/01/im-sorry-but-i-dont-like-this-contract-anymore/

pchdtvr - Not a Totally Satisfying Conclusion in Regards to Legal Issues

I've never used this package (apparently its used for HDTV scheduling/recording on Linux), but this link on Slashdot caught my eye: http://sourceforge.net/developer/diary.php?diary_id=26407&diary_user=147583. Apparently, the developer of this software package is seeking to revoke the GPL license not just for his current code, but his past code/package as well. I have a difficult time believing that this is possible, but I'm sure we will soon find out. My guess is that this guy is productizing his software and has a good idea who is currently using, selling and distributing his source so there will likely be some kind of legal challenge to the GPL as well. It's always interesting to see how these kind of things play out in the U.S. courts which can sometimes be a little schizophrenic, though I'd have a difficult time believing that this type of retroactive license change is actually possible.

User: reeset, "atscap and pchdtvr GPL revoked or can it be". URL: https://blog.reeset.net/archives/493



"Can GPL licensed code be close sourced later by the author? [duplicate]". URL: https://softwareengineering.stackexchange.com/questions/98774/can-gpl-licensed-code-be-close-sourced-later-by-the-author/98778

pchdtvr - GPL Means We Still Have the Source

pchdtvr.c

```
/*****************************************************************************
 *
 * pchdtvr.c    (c) Copyright 2004-2006 by inkling@nop.org
 *    ATSC Transport Stream Capture Application Program
 *  for the pcHDTV Inc., HD-2000 and HD-3000 Terrestrial DTV adapters
 *
 *  Watch your DTV shows WHEN, WHERE and HOW you want to watch them.
 *
 *  ****** pchdtvr is ONLY for Digital Terrestrial (Over The Air) ******
 *
 * pchdtvr is free software; you may only redistribute it and/or modify
 * it under the terms of the GNU General Public License Version 2, as
 * published by the Free Software Foundation.
 *
 * pchdtvr is distributed to you in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY OR SUPPORT; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  Please see the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License Version 2
 * along with this program; if not, write me or the Free Software Foundation,
 * Inc., at 51 Franklin Street, Fifth Floor, Boston, MA 02110-1301 USA
 *
 *****************************************************************************/
```

pchdtvr source available: http://frequal.com/pmn/pchdtvr.html [pchdtvr-1.0-frequal shown]

pchdtvr - We Can Find Our Old Friend `fork()`

pchdtvr.c

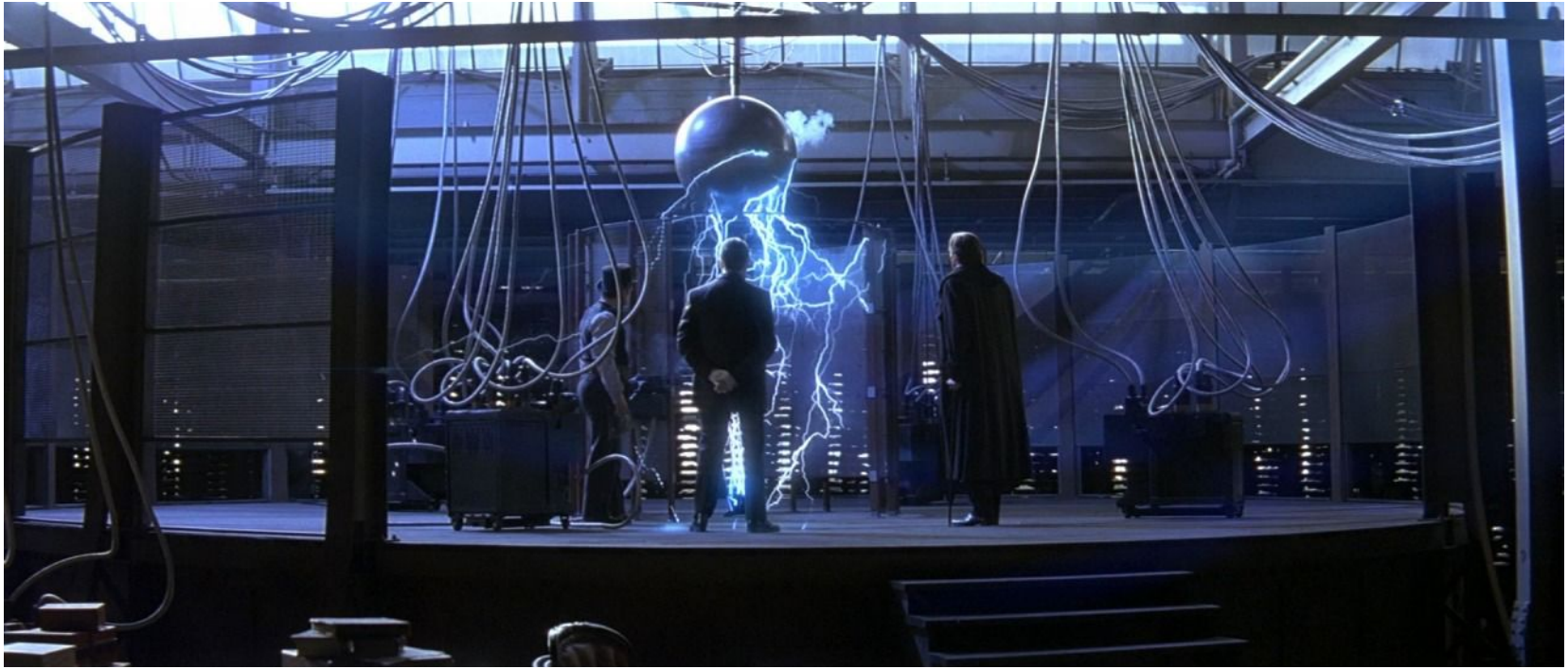
```
22250     if (0 != arg_detach)
22251     {
22252   pid_t fp;
22253   fp = fork();
22254   if (0 == fp)
22255   {
22256       /* new forked path, will fall thru */
22257       pid_m = getpid();
22258       asyslog( LOG_INFO,
22259         "daemon forked, main() is now pid %d", pid_m );
22260       /* fall thru to below */
22261   } else {
22262       /* old forker must die right here */
22263       console_exit(0);
22264   }
```

Remember: `pid_t` is an opaque type (not known by application)

`fork()` called

pchdtvr source available: http://frequal.com/pmn/pchdtvr.html [pchdtvr-1.0-frequal shown]

Christopher Nolan's Explanation of `fork()` Return Value



Scene from *The Prestige* (2006). URL: https://raysofcinema.tumblr.com/post/124342045723/the-prestige-2006-directed-by-christopher-nolan

pchdtvr - We Can Find Our Old Friend `fork()`

pchdtvr.c

```
22250     if (0 != arg_detach)
22251     {
22252   pid_t fp;
22253   fp = fork();
22254   if (0 == fp)
22255   {
22256       /* new forked path, will fall thru */
22257       pid_m = getpid();
22258       asyslog( LOG_INFO,
22259         "daemon forked, main() is now pid %d", pid_m );
22260       /* fall thru to below */
22261   } else {
22262       /* old forker must die right here */
22263       console_exit(0);
22264   }
```

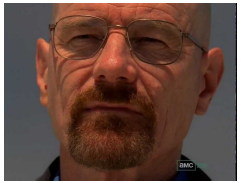
Remember: `pid_t` is an opaque type

`fork()` called

Am I the **child process**?

What is my **PID**?

"say my name"

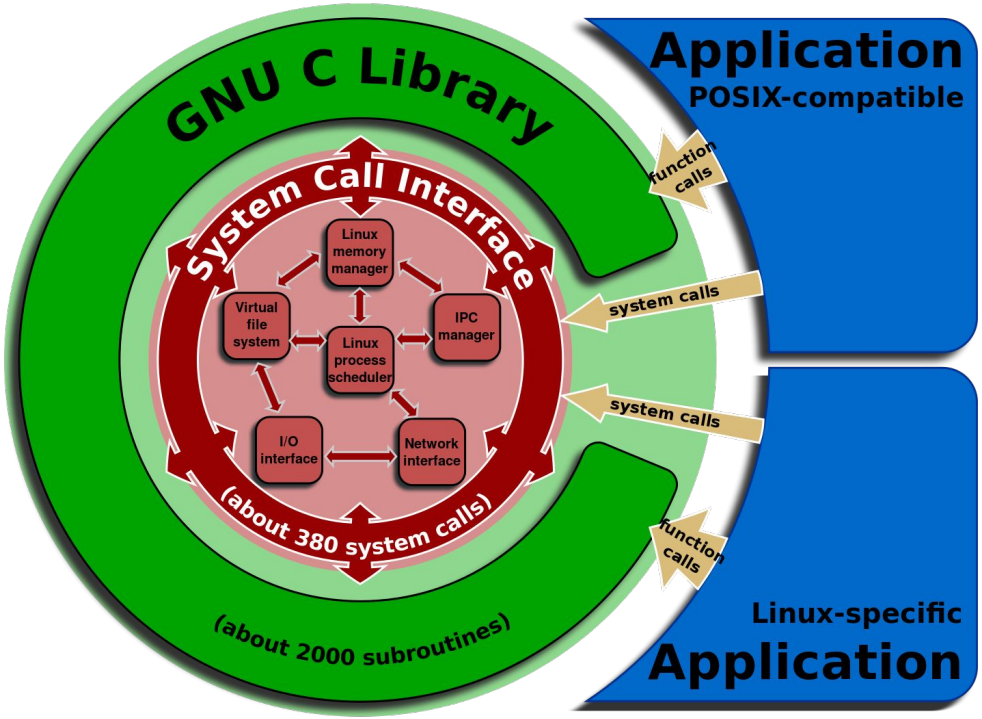


I'm the **parent process**
Death imminent

pchdtvr source available: http://frequal.com/pmn/pchdtvr.html [pchdtvr-1.0-frequal shown]

Walter White (language warning): https://www.youtube.com/watch?v=dy_DASt7hDs

Using System Calls Directly & Indirectly



Indirectly Using System Call Via `glibc`

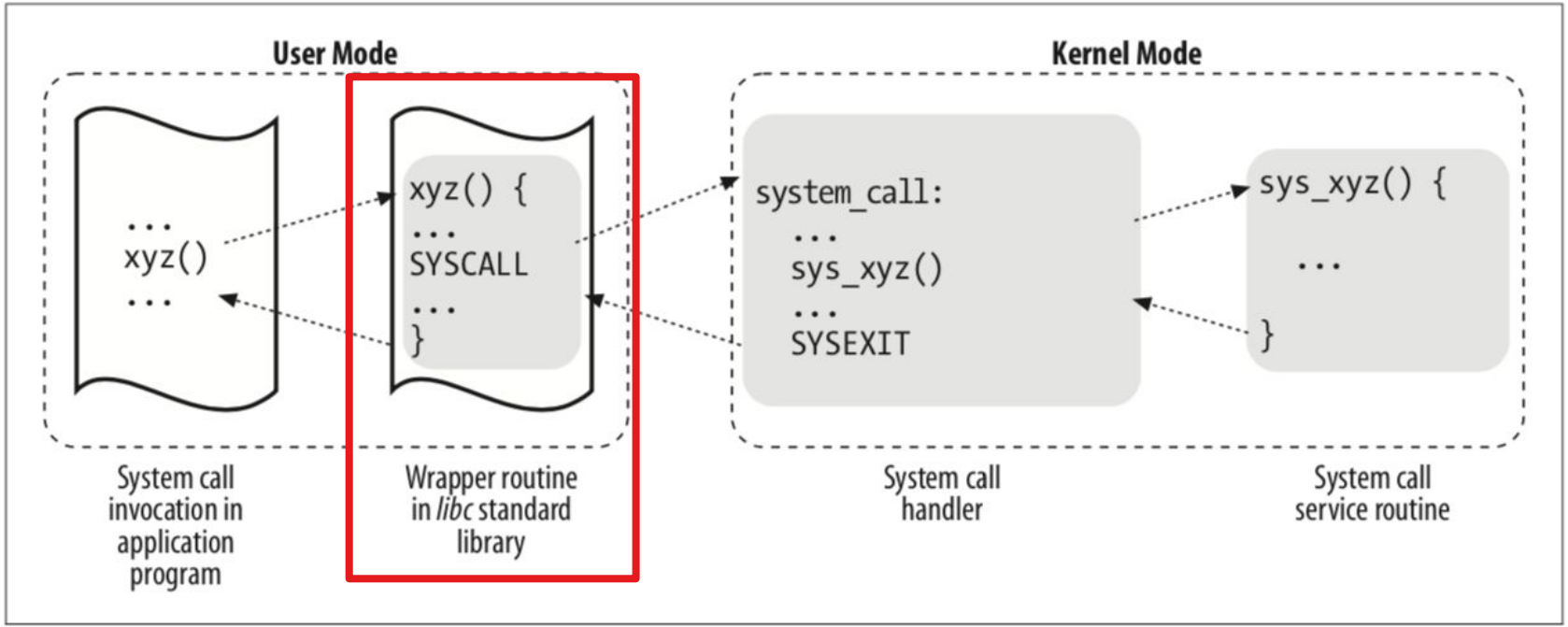
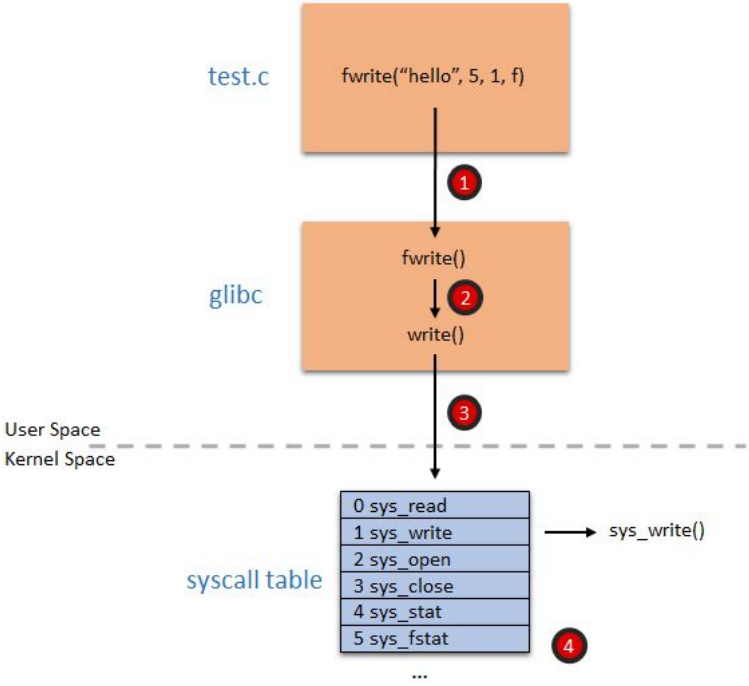


Figure 10-1. Invoking a system call

"Understanding the Linux Kernel 读书笔记-System Calls". URL: http://liujunming.top/2018/12/29/Understanding-the-Linux-Kernel-读书笔记-System-Calls/

Example of `write()` System Call Being Used



Loris Degioanni, "The Fascinating World of Linux System Calls". URL: https://sysdig.com/blog/fascinating-world-linux-system-calls/

`fork()` Does Not Call fork()

fork(2) — Linux manual page

NAME | SYNOPSIS | DESCRIPTION | RETURN VALUE | ERRORS | CONFORMING TO | NOTES | EXAMPLES | SEE ALSO | COLOPHON

Search online pages

FORK(2) Linux Programmer's Manual FORK(2)

NAME top

fork - create a child process

C library/kernel differences

Since version 2.3.3, rather than invoking the kernel's **fork**() system call, the glibc **fork**() wrapper that is provided as part of the NPTL threading implementation invokes clone(2) with flags that provide the same effect as the traditional system call. (A call to **fork**() is equivalent to a call to clone(2) specifying *flags* as just **SIGCHLD**.) The glibc wrapper invokes any fork handlers that have been established using pthread_atfork(3).

"fork(2) — Linux manual page". https://man7.org/linux/man-pages/man2/fork.2.html

Remember when?...

- 1983: UNIX System V released
  - **Richard Stallman also started GNU Project**
    - Started developing free programs functionally equivalent to UNIX programs
- The UNIX wars (late 80s to mid 90s)
  - What is UNIX? Many versions were floating around
- Efforts for peace were made
  - 1985: AT&T introduced the System V Interface Definition (SVID)
  - **1988: IEEE POSIX (Portable Operating System Interface)**
  - AT&T worked with others to merge other UNIX versions into System V
  - The Open Software Foundation countered with OSF/1
  - All the while, more commercial distributions of UNIX appeared that expanded beyond AT&T's standard

Richard Stallman

From the cover of: *Free as in Freedom*, Sam Williams.
https://www.amazon.com/Free-Freedom-Richard-Stallmans-Software-dp-0596002874/dp/0596002874/ref=mt_other?_encoding=UTF8&me=&qid=1630343742

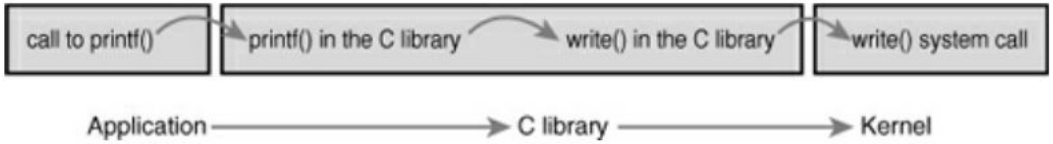
https://frontpagelinux.com/articles/guide-through-history-of-unix-linux-everything-you-need-to-know/

POSIX - Portable Operating System Interface

- **POSIX** encourages compatibility among OS's for applications via **set of standards**
- UNIX Wars - Guarantee application worked across UNIX flavors
- Standardizes:
  - Shell
  - **API (application programming interface) to kernel**
- Many things are not standardized
  - GUIs
- This does not talk about **binary compatibility**

POSIX - API Specifics

- Common API enables applications to access kernel services across UNIX and UNIX-like systems
- POSIX talks about **System Interfaces**:
  - “the functions, macros, and external variables to support applications portability at the C-language source level” [IEEE & The Open Group: link]
- Implementation unspecified - can include:
  - System calls via the C library
  - Staying *entirely* within the C library (user space)
- `glibc` - The GNU C Library
  - C standard library realization by GNU Project



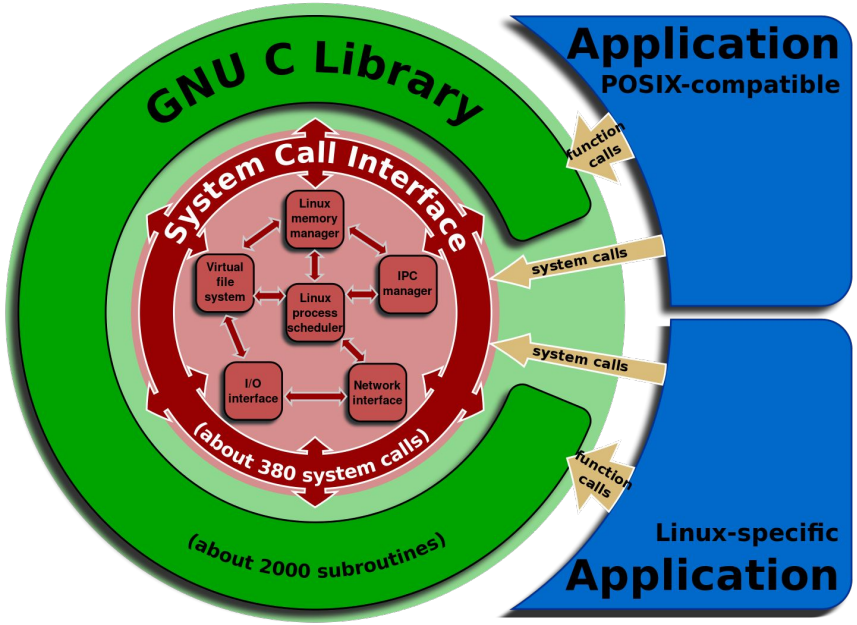
Robert Love, *Linux Kernel Development*, 3rd Edition. Figure from page 70.

POSIX - System Interface Examples

- List of POSIX system interfaces available on The Open Group's site here
  - Again, not the same, but we have a list of Linux system calls here
- System interface: `fork()`
  - System call: `fork()`
  - But we know glibc will not use the `fork()` system call, it will use the `clone()` system call
    - Note: no `clone()` system interface
- System interface: `fopen()`
  - System call: `open()`
- System interface: `sqrt()`
  - System call: Assumably none

Using System Call Directly or Indirectly

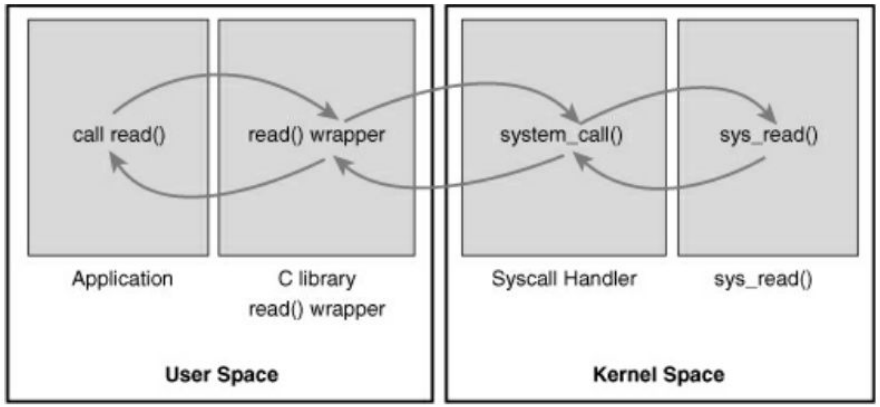
```
#include <syscall.h>
#include <unistd.h>
#include <stdio.h>
#include <sys/types.h>

int main(void) {

      long ID1, ID2;
      /*-----------------------------*/
      /* direct system call */
      /* SYS_getpid (func no. is 20) */
      /*-----------------------------*/
      ID1 = syscall(SYS_getpid);
      printf ("syscall(SYS_getpid)=%ld\n", ID1);

      /*-----------------------------*/
      /* "libc" wrapped system call */
      /* SYS_getpid (Func No. is 20) */
      /*-----------------------------*/
      ID2 = getpid();
      printf ("getpid()=%ld\n", ID2);
      return(0);
}
```



Code from: Jialong He, “LINUX System Call Quick Reference”. URL: http://www.cheat-sheets.org/saved-copy/Linux_Syscall_quickref.pdf

How Does This Really Work? (x86 Edition)

- Cause a **software interrupt**: exception/trap/fault
- **Kernel mode** activated
- Exception handler (**system call handler**) runs
- Which system call?
  - Use **eax register** to give system call # to kernel
  - **eax** read by handler



Robert Love, *Linux Kernel Development*, 3rd Edition. Top right fig. from p. 73

System Call Numbers (x86-64)

/ arch / x86 / entry / syscalls / syscall_64.tbl

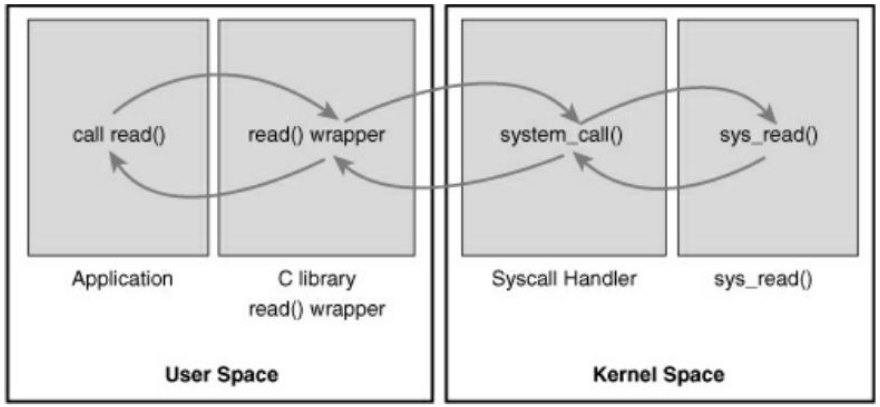
All sy Search Identifier

```
#
# 64-bit system call numbers and entry vectors
#
# The format is:
# <number> <abi> <name> <entry point>
#
# The __x64_sys_*() stubs are created on-the-fly for sys_*() system calls
#
# The abi is "common", "64" or "x32" for this file.
#
0       common  read                    sys_read
1       common  write                   sys_write
2       common  open                    sys_open
3       common  close                   sys_close
4       common  stat                    sys_newstat
5       common  fstat                   sys_newfstat
6       common  lstat                   sys_newlstat
7       common  poll                    sys_poll
```


Linux Source via Bootlin Elixir Cross Referencer. URL: https://elixir.bootlin.com/linux/latest/source/arch/x86/entry/syscalls/syscall_64.tbl

Parameter Passing & Return Value

- Use mechanism similar to syscall # transmission
- Place parameters in **registers**
- Shown registers are for 5 arguments
- If more needed, put user-space pointer in 1 register
  - No other registers used
- How do we get the return value?
  - `eax`



Robert Love, *Linux Kernel Development*, 3rd Edition. Top right fig. from p. 73

4 Simple Steps to Making Your Very Own System Call

1. What is the point of its existence?
2. Write the code & add to file
3. Add call to the system call table
4. Compile

Robert Love, *Linux Kernel Development*, 3rd Edition

1. What Is The Point of The System Call?

- Linux philosophy: syscall does one thing
- Determine:
  - Input arguments
  - Return value
  - Error codes
- Unix motto: “Provide mechanism, not policy”
- Portability, robustness
  - Fundamental syscalls have been helpful for 30 years!

Robert Love, *Linux Kernel Development*, 3rd Edition

2. Write The System Call Code: Parameter Verification

- We’re in **kernel space**, tread carefully
- **File I/O**: Is file descriptor legitimate?
- **Processes**: Is PID legitimate?
- **Permissions**: Is the process allowed to see this?
- **Pointers** 💀
  - Is this really pointing to user space?
  - Is this really pointing to process’s space?
  - Is the memory readable/writable/executable?

Robert Love, *Linux Kernel Development*, 3rd Edition

2. Example of System Call Code

```
SYSCALL_DEFINE1(stephen, char *, msg)
{
  char buf[256];
  long copied = strncpy_from_user(buf, msg, sizeof(buf));
  if (copied < 0 || copied == sizeof(buf))
    return -EFAULT;
  printk(KERN_INFO "stephen syscall called with \"%s\"\n", buf);
  return 0;
}
```

Stephen Brennan, “Tutorial - Write a System Call”. URL: https://brennan.io/2016/11/14/kernel-dev-ep3/

Include syscall code in kernel file

- E.g. `sys.c`

3. Let’s Make It Official: Add to Syscall Table

- Append call to end of syscall table
  - Number defined by ranking in table
  - Architecture-specific

Robert Love, *Linux Kernel Development*, 3rd Edition
Stephen Brennan, “Tutorial - Write a System Call”. URL: https://brennan.io/2016/11/14/kernel-dev-ep3/

Add to `syscall_64.tbl`

/ arch / x86 / entry / syscalls / syscall_64.tbl All syml Search

```
437     common  openat2                 sys_openat2
438     common  pidfd_getfd             sys_pidfd_getfd
439     common  faccessat2              sys_faccessat2
440     common  process_madvise         sys_process_madvise
441     common  epoll_pwait2            sys_epoll_pwait2
442     common  mount_setattr           sys_mount_setattr
443     common  quotactl_fd             sys_quotactl_fd
444     common  landlock_create_ruleset sys_landlock_create_ruleset
445     common  landlock_add_rule       sys_landlock_add_rule
446     common  landlock_restrict_self  sys_landlock_restrict_self
447     common  memfd_secret            sys_memfd_secret
```


Linux Source via Bootlin Elixir Cross Referencer. URL: https://elixir.bootlin.com/linux/latest/source/arch/x86/entry/syscalls/syscall_64.tbl

4. Final Step

Compile and cross your fingers

Test Ride Your Beautiful New Syscall

- `glibc` contains the accepted system calls
- Yours is brand new!
- Use `syscall()` function

```c
/*
 * Test the stephen syscall (#329)
 */
#define _GNU_SOURCE
#include <unistd.h>
#include <sys/syscall.h>
#include <stdio.h>

/*
 * Put your syscall number here.
 */
#define SYS_stephen 329

int main(int argc, char **argv)
{
  if (argc <= 1) {
    printf("Must provide a string to give to system call.\n");
    return -1;
  }
  char *arg = argv[1];
  printf("Making system call with \"%s\".\n", arg);
  long res = syscall(SYS_stephen, arg);
  printf("System call returned %ld.\n", res);
  return res;
}
```

Stephen Brennan, “Tutorial - Write a System Call”. URL: https://brennan.io/2016/11/14/kernel-dev-ep3/

Robert Love, *Linux Kernel Development*, 3rd Edition