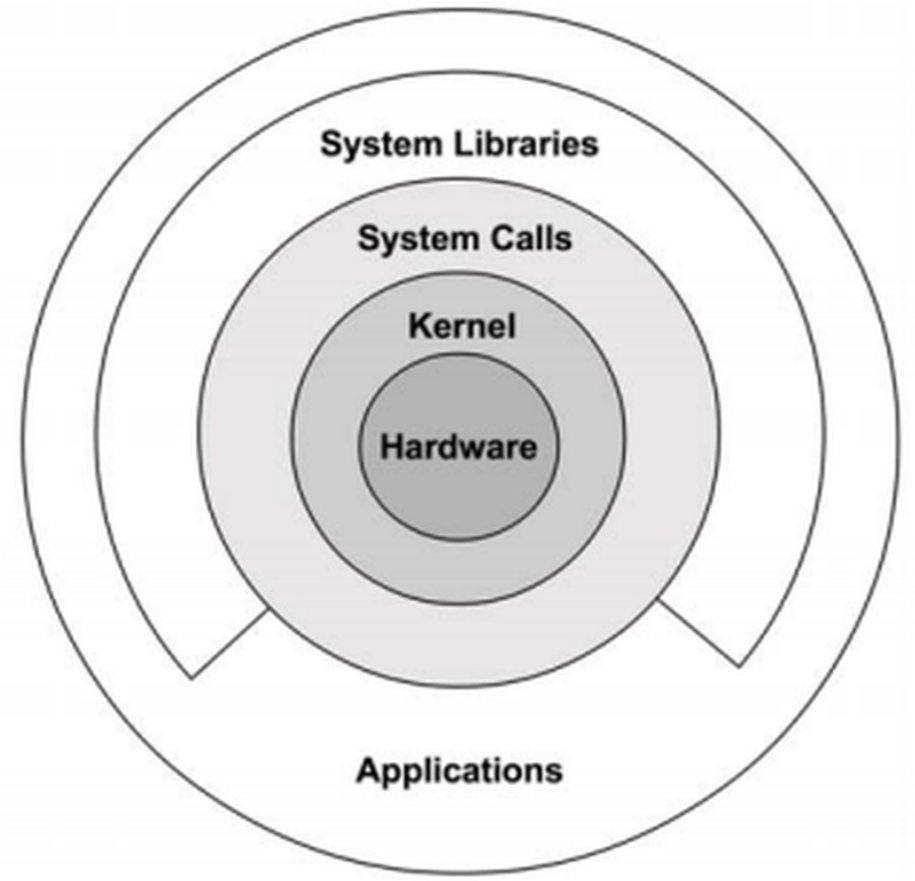
Lecture 2 - System Programming Concepts

EE3233 Systems Programming for Engrs Reference: M. Kerrisk, The Linux Programming Interface

UTSA® The University of Texas at San Antonio™

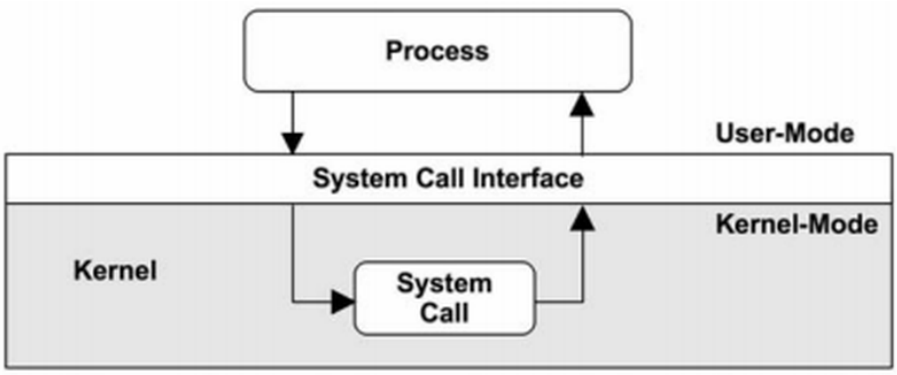
System Calls

- Entry point into kernel
- With System Calls, a process requests to kernel to perform some action on the process's behalf
  - Kernel makes a range of services (creating new process, performing I/O, creating a pipe for IPC) accessible to programs via the system call API



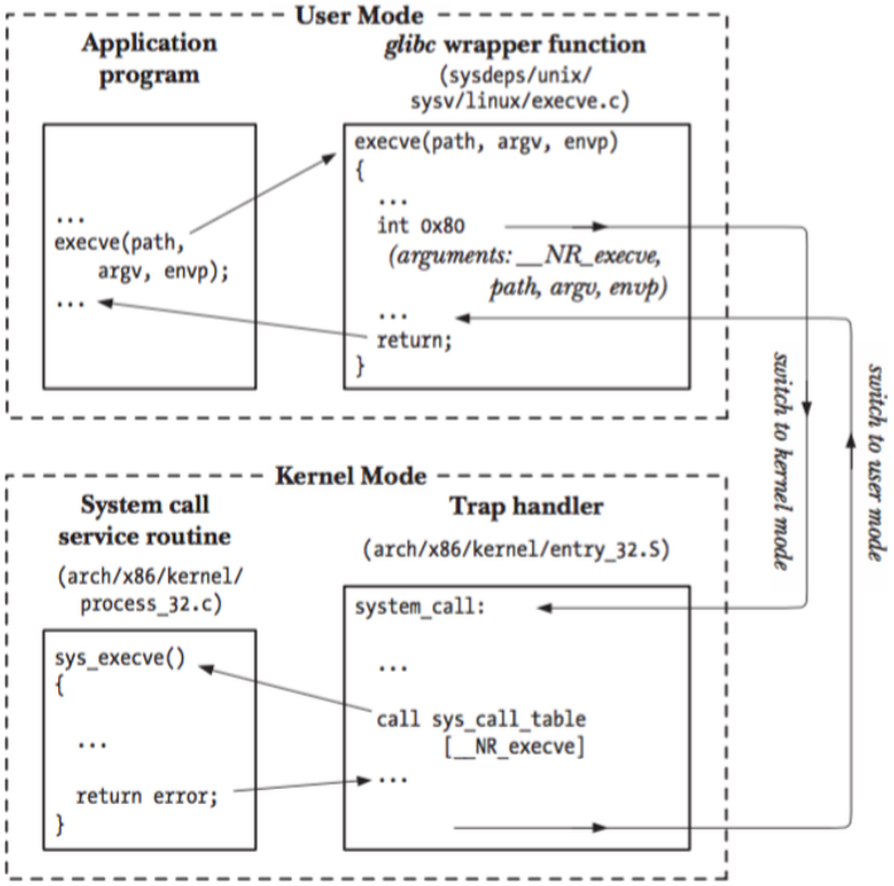
System Calls - Cont'd

- A system call changes processor state from user mode to kernel mode
  - so that CPU can access protected kernel memory
- The set of system calls is fixed
- Arguments of system call are transferred from user space to kernel space (vice versa)



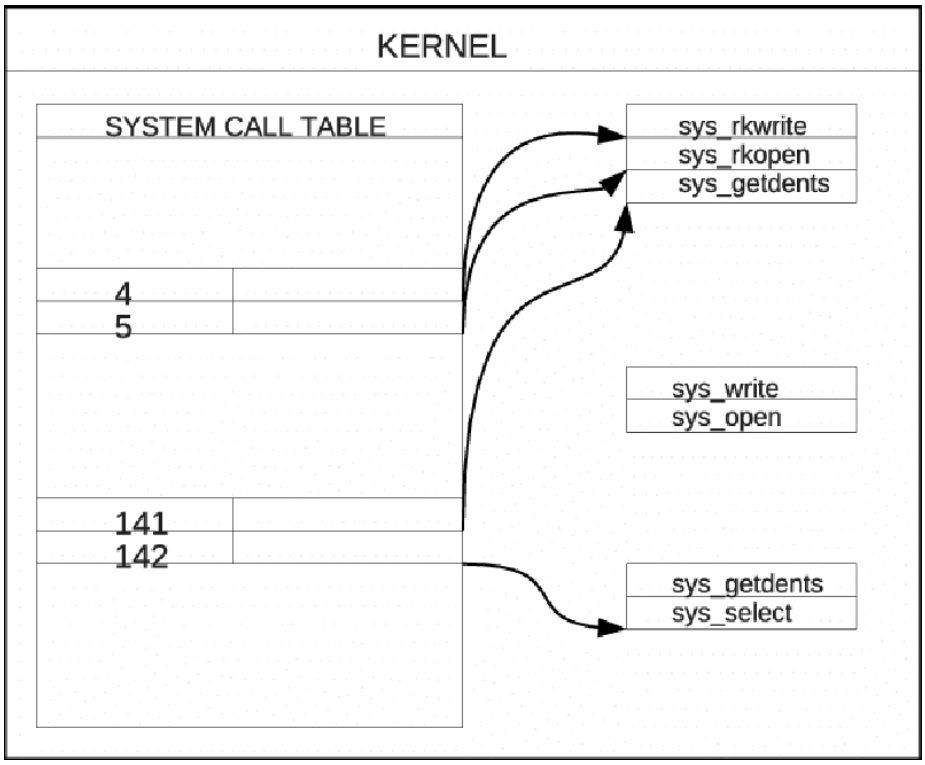
System Calls - Cont'd

1. Application makes a system call by invoking a wrapper function in the C library
2. Wrapper function makes all arguments available to trap-handling routine
   - These arguments are passed to wrapper via the stack
   - Wrapper copies arguments to register (kernel requires them in registers)
3. Wrapper copies system call number into a specific CPU register (%eax)
4. Wrapper executes a trap machine instruction (int 0x80) : switch user mode $\rightarrow$ kernel mode
5. Kernel invokes its system_call() routine located at "arch/x86/kernel/entry_32.S"
   - Saves register values onto the kernel stack
   - Checks the validity of the system call number
   - Invokes appropriate system call service routine
   - If the system call service routine has any arguments, it first checks their validity (addresses of pointers are valid)
6. Trap handler places the system call return value on the stack
   - If the system call service routine returned the error value, the wrapper function sets the global variable *errno* using this value
   - Simultaneously returning to user mode



System Calls - Cont'd

`execve()` : system call number 11 (_NR_execve) * in system_call_table vector, entry 11 contains the address of sys_execve(), service routine for this system call (sys_xxx() is a typical name in Linux) * "invoking the system call xyz()" * means "calling the wrapper function that invokes the system call xyz()"



Library Functions

- Functions that constitute standard C library
- Many library functions don't use system calls
  - e.g., string manipulation functions
- Some library functions are layered on top of system calls
  - e.g., `fopen()` library function uses the `open()` system call
  - designed to provide caller friendly interface
    - e.g., `printf()` provides output formatting while `write()` just outputs a block of bytes
    - e.g., `malloc()` and `free()` performs various bookkeeping tasks while `brk()` system call does not

---

Standard C Library (glibc)

- The most commonly used implementation on Linux is GNU C library (glibc)
- Version of glibc

```
$ /lib/x86_64-linux_gnu/libc.so.6
```

```
root@90dca184ddd2:/notebooks/temp# /lib/x86_64-linux-gnu/libc.so.6
GNU C Library (Ubuntu GLIBC 2.23-0ubuntu10) stable release version 2.23, by Roland McGrath e
Copyright (C) 2016 Free Software Foundation, Inc.
This is free software; see the source for copying conditions.
There is NO warranty; not even for MERCHANTABILITY or FITNESS FOR A
PARTICULAR PURPOSE.
Compiled by GNU CC version 5.4.0 20160609.
Available extensions:
crypt add-on version 2.1 by Michael Glad and others
GNU Libidn by Simon Josefsson
Native POSIX Threads Library by Ulrich Drepper et al
BIND-8.2.3-T5B
libc ABIs: UNIQUE IFUNC
For bug reporting instructions, please see:
<https://bugs.launchpad.net/ubuntu/+source/glibc/+bugs>.
```

Handling Errors from System Calls and Library Functions

- They return status value indicating success or fail
  - This status value should always be checked
  - Excluding this check to save time is a false economy

Handling system call errors

Usually, an error is indicated by a return of -1

```
fd = open(pathname, flags, mode);   /* system call to open a file */
if ( fd == -1 )
{
    /* Code to handle the error */
}
***
if ( close(fd) == -1 )
{
    /* Code to handle the error */
}
```

Handling system call errors

- When a system call fails, it sets the global variable *errno* to a positive value (specific error)

– you should include <errno.h> header (declaration of *errno*)

```
cnt = read(fd, buf, numbytes);
if (cnt == -1)
{
    if (errno == EINTR) {
        fprintf(stderr, "read was interrupted by a signal\n");
    }
    else {
        /* Some other error occurred */
    }
}
```

/usr/include/asm-generic/errno-base.h

Handling system call errors

- Common action after a failed system call is to print an error message based on the *errno* value : use perror()
- `perror()` prints the string pointed by msg and message corresponding to the current value of *errno*

Prototype:

```
#include <stdio.h>

void perror (const char *msg);
```

```
fd = open (pathname, flags, mode);
if  ( fd == -1 )
{
    perror ("open");
    exit (EXIT_FAILURE);
}
```

System Data Type

- Most of the standard system data types have names ending in **_t**

```
typedef  int  pid_t;        /* intended to represent process IDs */

pid_t mypid;
```

System Data Type

Data type	SUSv3 type requirement	Description
blkcnt_t	signed integer	File block count (Section 15.1)
blksize_t	signed integer	File block size (Section 15.1)
cc_t	unsigned integer	Terminal special character (Section 62.4)
clock_t	integer or real-floating	System time in clock ticks (Section 10.7)
clockid_t	an arithmetic type	Clock identifier for POSIX.1b clock and timer functions (Section 23.6)
comp_t	not in SUSv3	Compressed clock ticks (Section 28.1)
dev_t	an arithmetic type	Device number, consisting of major and minor numbers (Section 15.1)
DIR	no type requirement	Directory stream (Section 18.8)
fd_set	structure type	File descriptor set for *select()* (Section 63.2.1)
fsblkcnt_t	unsigned integer	File-system block count (Section 14.11)
fsfilcnt_t	unsigned integer	File count (Section 14.11)
gid_t	integer	Numeric group identifier (Section 8.3)
id_t	integer	A generic type for holding identifiers; large enough to hold at least *pid_t*, *uid_t*, and *gid_t*
in_addr_t	32-bit unsigned integer	IPv4 address (Section 59.4)
in_port_t	16-bit unsigned integer	IP port number (Section 59.4)
ino_t	unsigned integer	File i-node number (Section 15.1)
key_t	an arithmetic type	System V IPC key (Section 45.2)
mode_t	integer	File permissions and type (Section 15.1)
mqd_t	no type requirement, but shall not be an array type	POSIX message queue descriptor
msglen_t	unsigned integer	Number of bytes allowed in System V message queue (Section 46.4)

Data type	SUSv3 type requirement	Description
msgqnum_t	unsigned integer	Counts of messages in System V message queue (Section 46.4)
nfds_t	unsigned integer	Number of file descriptors for *poll()* (Section 63.2.2)
nlink_t	integer	Count of (hard) links to a file (Section 15.1)
off_t	signed integer	File offset or size (Sections 4.7 and 15.1)
pid_t	signed integer	Process ID, process group ID, or session ID (Sections 6.2, 34.2, and 34.3)
ptrdiff_t	signed integer	Difference between two pointer values, as a signed integer
rlim_t	unsigned integer	Resource limit (Section 36.2)
sa_family_t	unsigned integer	Socket address family (Section 56.4)
shmatt_t	unsigned integer	Count of attached processes for a System V shared memory segment (Section 48.8)
sig_atomic_t	integer	Data type that can be atomically accessed (Section 21.1.3)
siginfo_t	structure type	Information about the origin of a signal (Section 21.4)
sigset_t	integer or structure type	Signal set (Section 20.9)
size_t	unsigned integer	Size of an object in bytes
socklen_t	integer type of at least 32 bits	Size of a socket address structure in bytes (Section 56.3)
speed_t	unsigned integer	Terminal line speed (Section 62.7)
ssize_t	signed integer	Byte count or (negative) error indication
stack_t	structure type	Description of an alternate signal stack (Section 21.3)
suseconds_t	signed integer allowing range [−1, 1000000]	Microsecond time interval (Section 10.1)
tcflag_t	unsigned integer	Terminal mode flag bit mask (Section 62.2)
time_t	integer or real-floating	Calendar time in seconds since the Epoch (Section 10.1)
timer_t	an arithmetic type	Timer identifier for POSIX.1b interval timer functions (Section 23.6)
uid_t	integer	Numeric user identifier (Section 8.1)