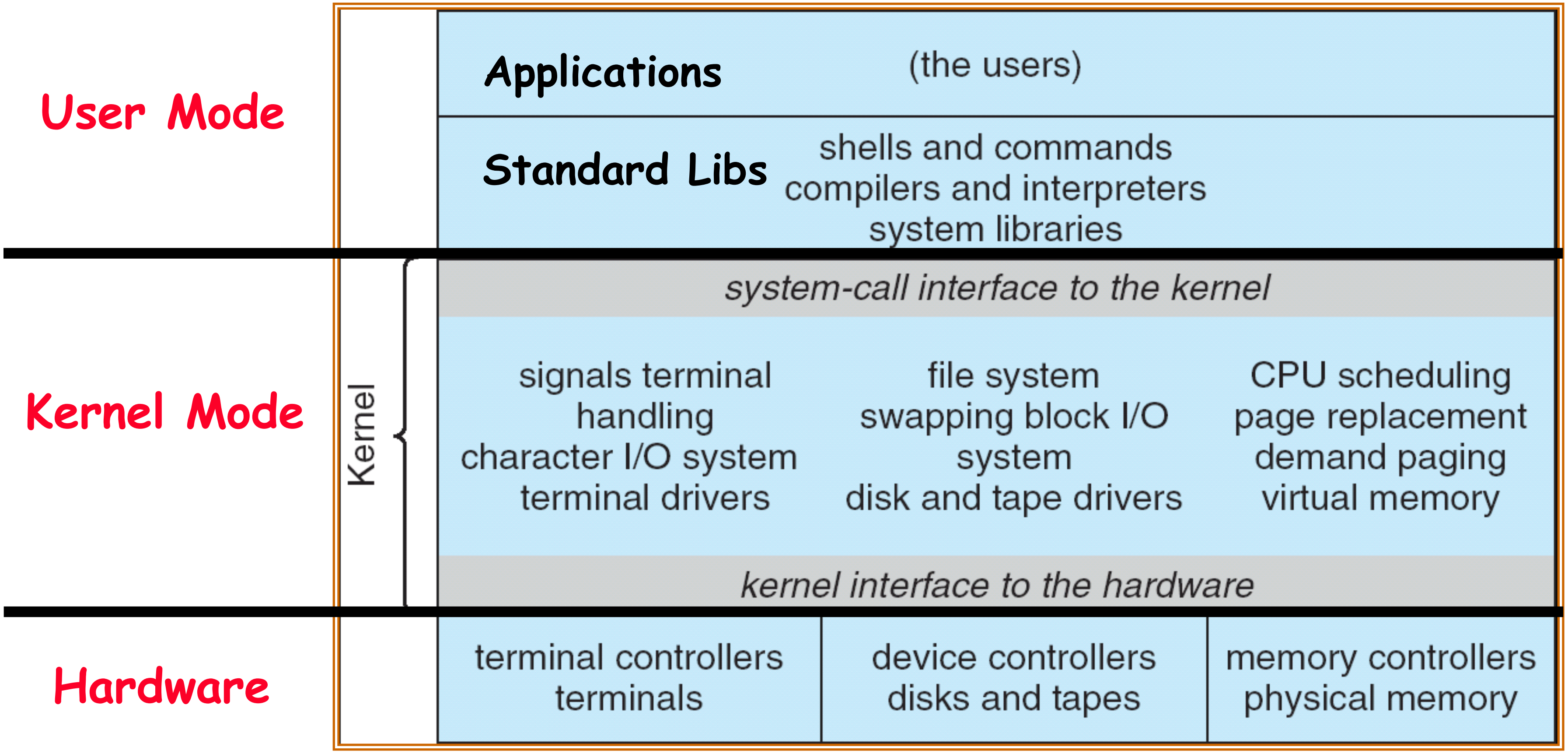
CS3510
Operating Systems

System Calls Interface

Bheemarjuna Reddy Tamma
IIT HYD

UNIX System Structure



User Mode

Kernel Mode

Hardware

Applications (the users)

Standard Libs
shells and commands
compilers and interpreters
system libraries

Kernel

system-call interface to the kernel

signals terminal handling
character I/O system
terminal drivers

file system
swapping block I/O system
disk and tape drivers

CPU scheduling
page replacement
demand paging
virtual memory

kernel interface to the hardware

terminal controllers
terminals

device controllers
disks and tapes

memory controllers
physical memory

System Calls

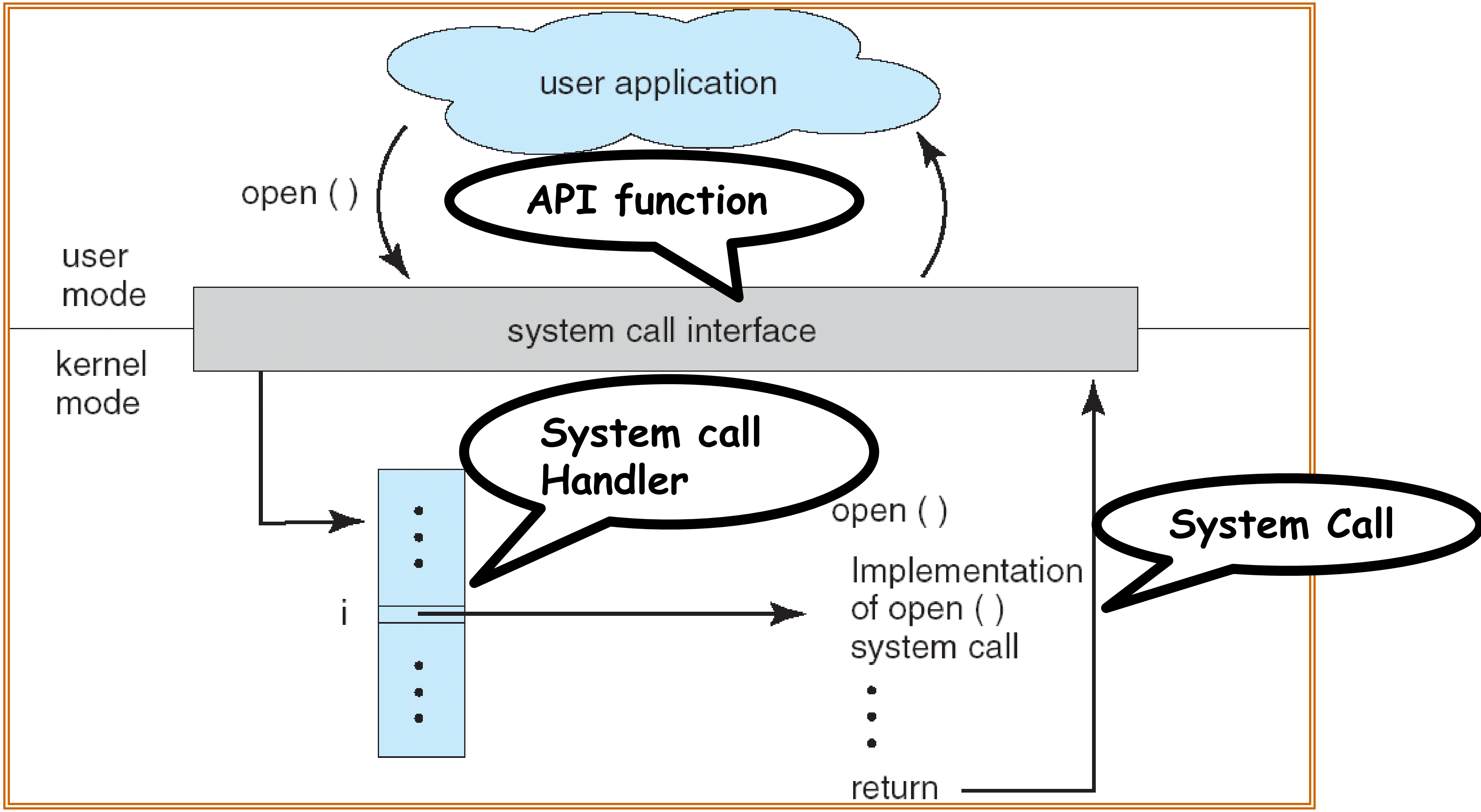
- Programming interface to the services provided by the OS to system/app programs
- Typically written in a high-level language (C/C++)
- Mostly accessed by application/system programs using procedure calls in APIs
- Programmer/job → procedure in API → System call
  - API is a function definition that specifies how to obtain a given service
  - System call is an explicit request to kernel made via a trap i.e., software interrupt
- Three most common APIs:
  - Win32 API for Windows
  - POSIX API for POSIX-based systems (UNIX, Linux, Mac OS X)
  - Java API for the Java virtual machine (JVM)
- A programmer accesses an API via a library of code (eg., libc for C programs in Linux) provided by OS

System Calls

- POSIX.1-2017 (IEEE 1003.1, ISO/IEC 9945)
  - Very widely used standard based on (and including) *C*-language
  - POSIX std refers to the API, not actual system calls provided by the kernel
  - Defines both
    - *API* and
    - compulsory *system programs/common utilities* together with their functionality and command-line format
      - E.g. `ls -w dir` prints the list of files in a directory in a 'wide' format
  - Complete specification is at http://www.opengroup.org/onlinepubs/9699919799/nframe.html

- Strong correlation b/w a procedure/function in API and its associated system call within kernel
  - Typically One-to-one (e.g., read, open, write, exit)
  - many-to-one (e.g., exec,brk) and one-to-many

System Calls



Instructions used to transition to kernel mode in diff archs

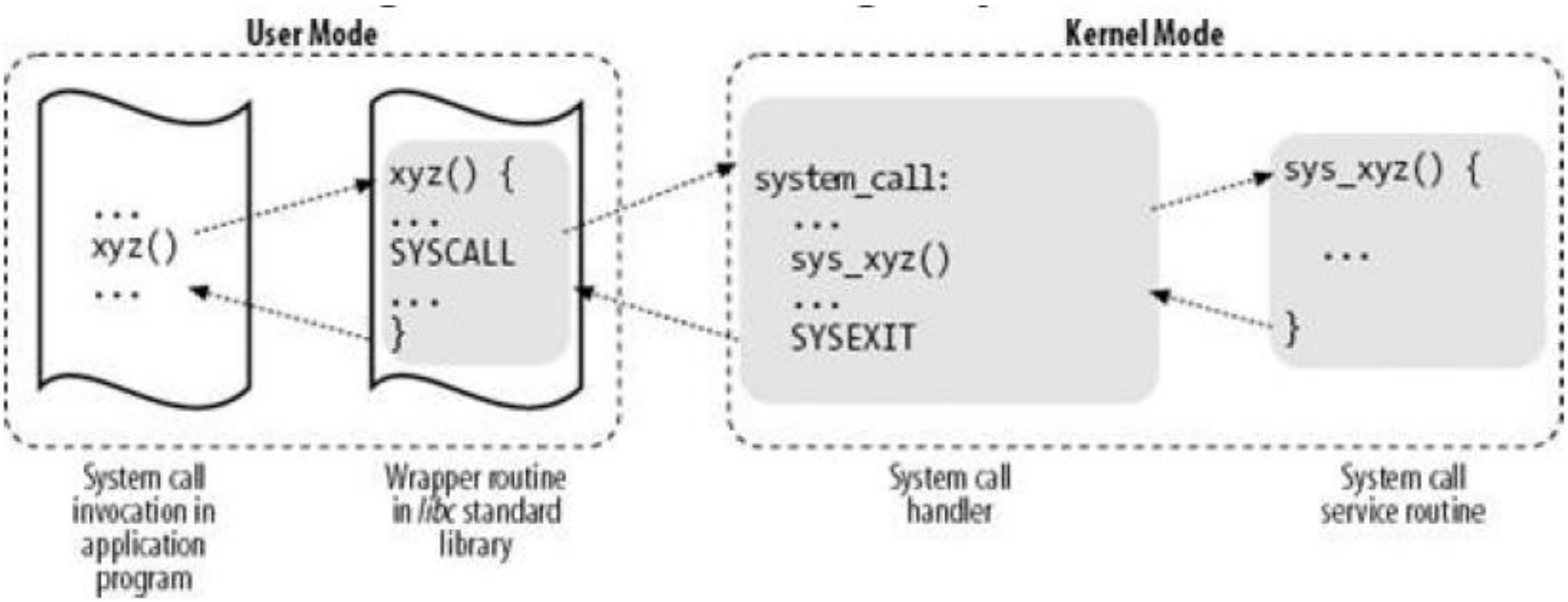
- i386 (int 0x80), eax register is used to indicate syscall number
- x86_64 (syscall), rax register is used similarly

API-System Call Implementation

- The interface to the services provided by the OS has two parts:

1. Higher language interface – a part of system library
   - Executes in user mode
   - Implemented to accept standard procedure calls
   - Traps to the Part 2
2. Kernel part
   - Executes in kernel mode
   - Implements the required system service
   - May cause blocking the caller (forcing it to wait)
   - After completion returns back to Part 1 (may report the success or failure of the call)

System Calls


User Mode
Kernel Mode
System call invocation in application program
Wrapper routine in *libc* standard library
System call handler
System call service routine

```
...
xyz()
...
```

```
xyz() {
...
SYSCALL
...
}
```

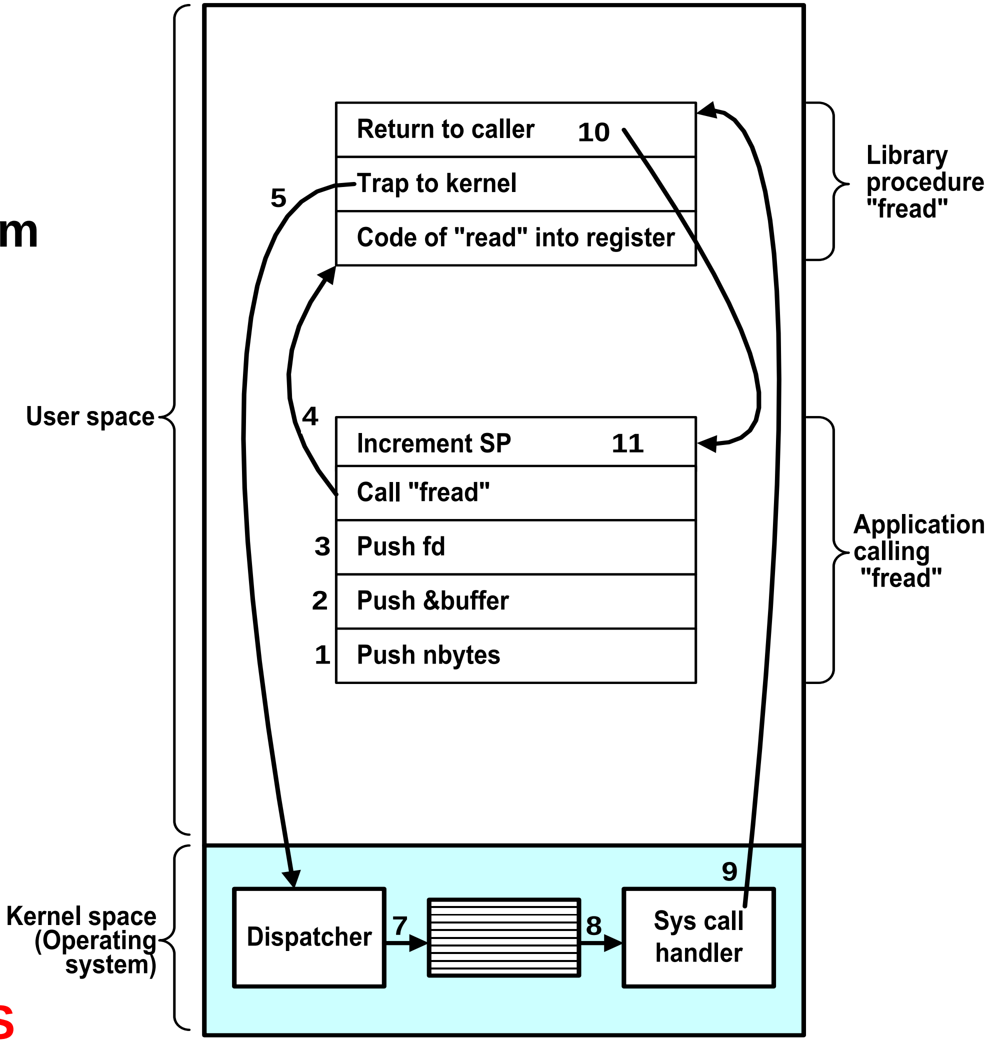
```
system_call:
  ...
  sys_xyz()
  ...
  SYSEXIT
```

```
sys_xyz() {
   ...
}
```

- **Why use APIs rather than system calls directly?**
  - Program portability
  - Easier to use

System Call Interface: Implementation

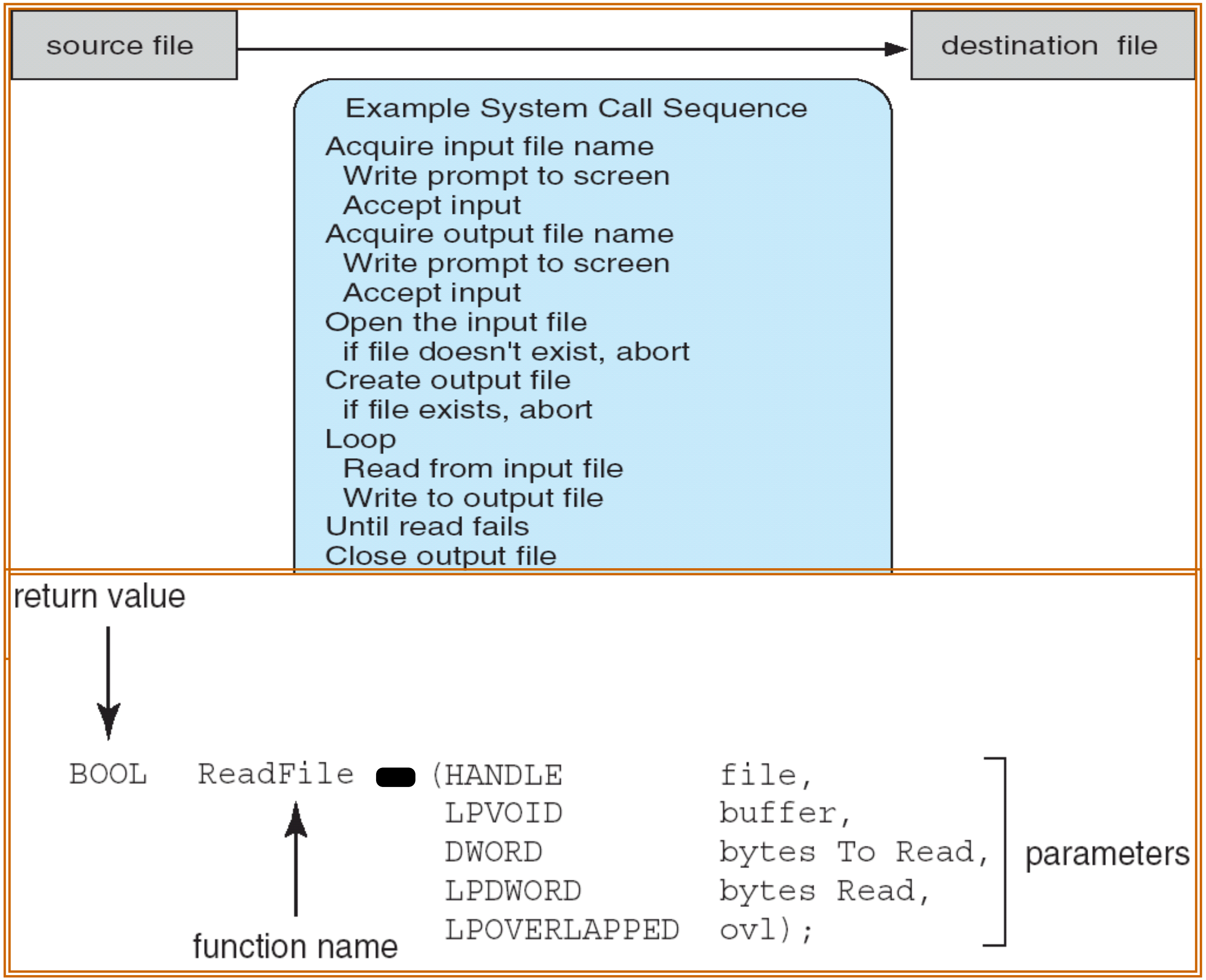
- An application program wants to make use of a System Call:
  - A system library routine is called first
  - It transforms the call to the system standard (*native API*) and traps to the kernel
  - Control is taken by the kernel running in the kernel mode
  - According to the service "code", the *Call dispatcher* invokes the responsible part of the Kernel
  - Depending on the nature of the required service, the kernel may block the calling process
- After the call is finished, the calling process execution resumes obtaining the result (success/failure) as if an ordinary function was called
- Three ways to pass parameters to OS
  - Registers
  - Stack
  - Memory block



11 steps to execute the service *fread (fd, buffer, nbytes)*

System Calls

- System call sequence to copy contents of one file to another



Example System Call Sequence

```
Acquire input file name
  Write prompt to screen
  Accept input
Acquire output file name
  Write prompt to screen
  Accept input
Open the input file
  if file doesn't exist, abort
Create output file
  if file exists, abort
Loop
  Read from input file
  Write to output file
Until read fails
Close output file
```

```
BOOL   ReadFile [redacted] (HANDLE        file,
                      LPVOID        buffer,
                      DWORD         bytes To Read,
                      LPDWORD       bytes Read,
                      LPOVERLAPPED  ovl);
```

glibc: GNU C Library

- Any Unix-like OS needs a C library
- C lang has no built-in facilities for doing I/O, memory management, string manipulation, etc
- A std C library (ISO C std) provides these facilities
- The GNU C Library (glibc) implements all of the functions specified in
  - ISO C library (malloc, printf, fopen, exit, etc)
  - POSIX.1 (system calls)
  - And extensions specific to GNU systems
- glibc is used as *the* C library in GNU systems and most systems with Linux kernel
  - Current version 2.32 (link)

glibc: GNU C Library

- glibc has procedures (wrapper functions) which in turn call system calls
  - getpid(), getppid(), chmod() are defined in glibc
- glibc provides syscall which helps you to call system calls explicitly (directly) from user/app program
  - syscall is also a library function!, but very simple one
  - long syscall (*long sysno, ...*)
  - sysno is system call number, refer <sys/syscall.h> for Macros
  - Return val is the return value of syscall pointed to by sysno
    - » -1 when system call is failed
  - Employing syscall() is useful when invoking a system call that has no wrapper function defined in the C library

  - http://man7.org/linux/man-pages/man2/syscall.2.html
  - http://man7.org/linux/man-pages/man2/syscalls.2.html
  - http://man7.org/linux/man-pages/man7/vdso.7.html
  - http://man7.org/linux/man-pages/man7/libc.7.html

Example 1

```
#define _GNU_SOURCE
 #include <unistd.h> //wrapper for syscalls
 #include <sys/syscall.h> // loc: /usr/src/include/i386-linux-gnu/bits/syscall.h, defines syscall numbers/Macros
 #include <sys/types.h>
 #include <stdio.h>

int main(int argc, char *argv[]) {
  pid_t tid;

  tid = syscall(SYS_gettid); //SYS_gettid does not have glibc wrapper function, so calling syscall directly using "syscall" func; refer man
syscall, man gettid

  printf("TID=%d\n", tid);

  tid = getpid(); //getpid is wrapper function given in glibc

  printf("PID=%d\n", tid);

  tid = getppid(); //getppid is wrapper in glibc

  printf("PPID=%d\n", tid);

  tid = syscall(__NR_getpid); //calling SYSCALL directly

  printf("PID=%d\n", tid);

  tid = syscall(SYS_getpid); //calling SYSCALL directly

  printf("PID=%d\n", tid);

  tid = syscall(__NR_getppid); //calling SYSCALL directly

  printf("PPID=%d\n", tid);

  return 0; }
```

Example 2: (kind of) direct system call

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

      ...
      int rc;

      rc = syscall(SYS_chmod, "/etc/passwd", 0444);

   if (rc == -1)
   fprintf(stderr, "chmod failed, errno = %d\n", errno);
       …
```

Example 2': glibc wrapper call

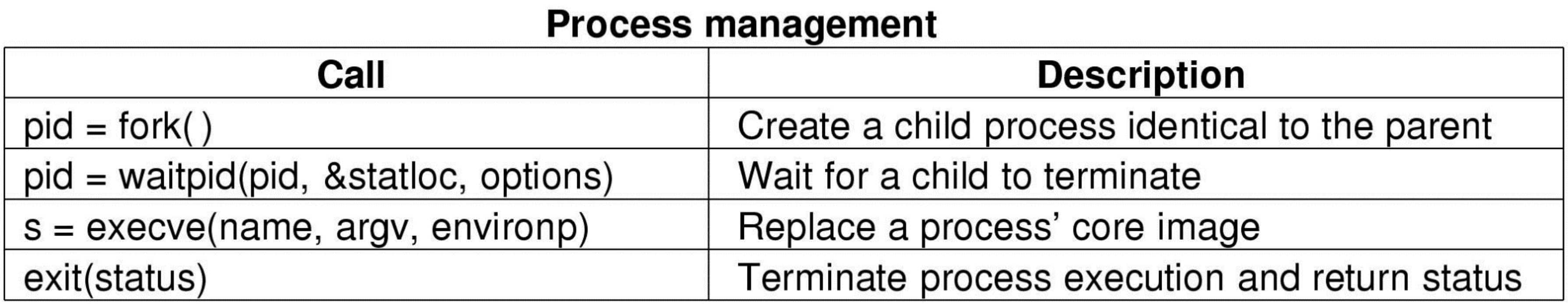
```
#include <sys/types.h>
#include <sys/stat.h>
#include <errno.h>

          ...

          int rc;

          rc = chmod("/etc/passwd", 0444);
          if (rc == -1)
fprintf(stderr, "chmod failed, errno = %d\n", errno);
           …
```

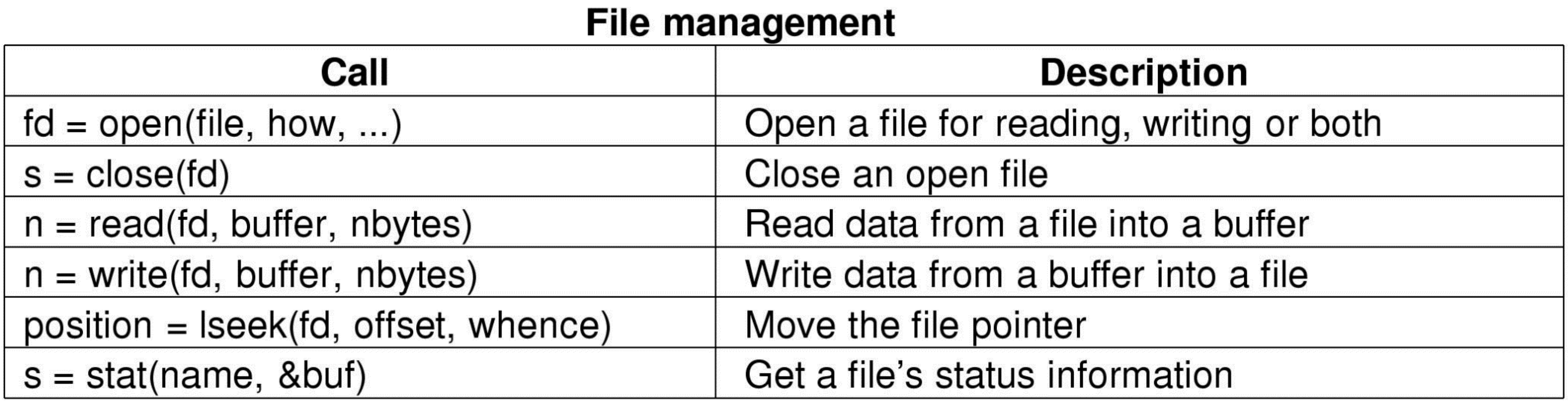
Some API Calls For Process Management



Process management

Call	Description
pid = fork( )	Create a child process identical to the parent
pid = waitpid(pid, &statloc, options)	Wait for a child to terminate
s = execve(name, argv, environp)	Replace a process’ core image
exit(status)	Terminate process execution and return status

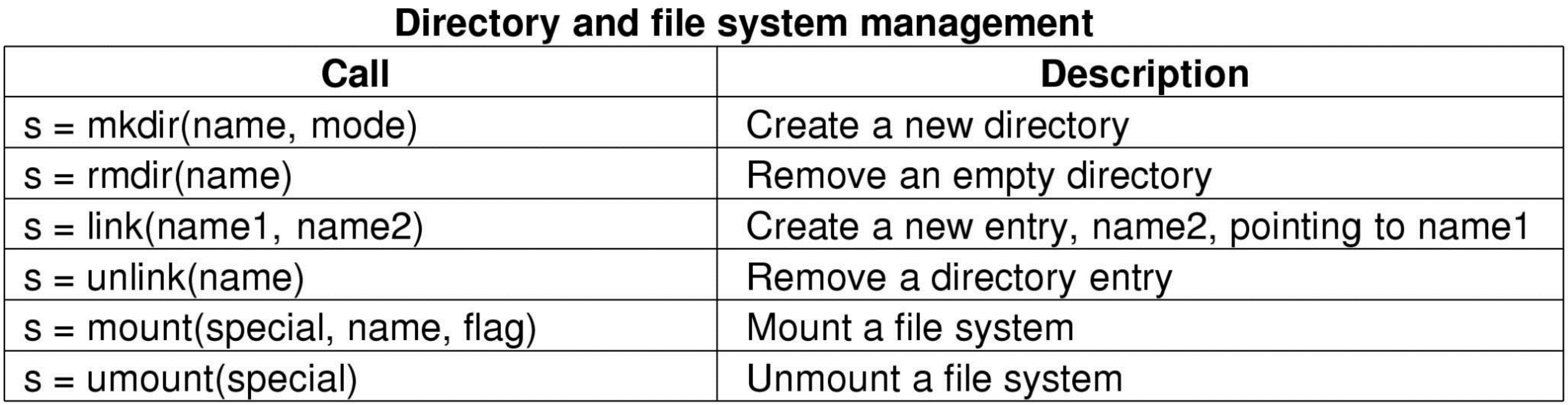
Some API Calls For File Management



File management

Call	Description
fd = open(file, how, ...)	Open a file for reading, writing or both
s = close(fd)	Close an open file
n = read(fd, buffer, nbytes)	Read data from a file into a buffer
n = write(fd, buffer, nbytes)	Write data from a buffer into a file
position = lseek(fd, offset, whence)	Move the file pointer
s = stat(name, &buf)	Get a file's status information

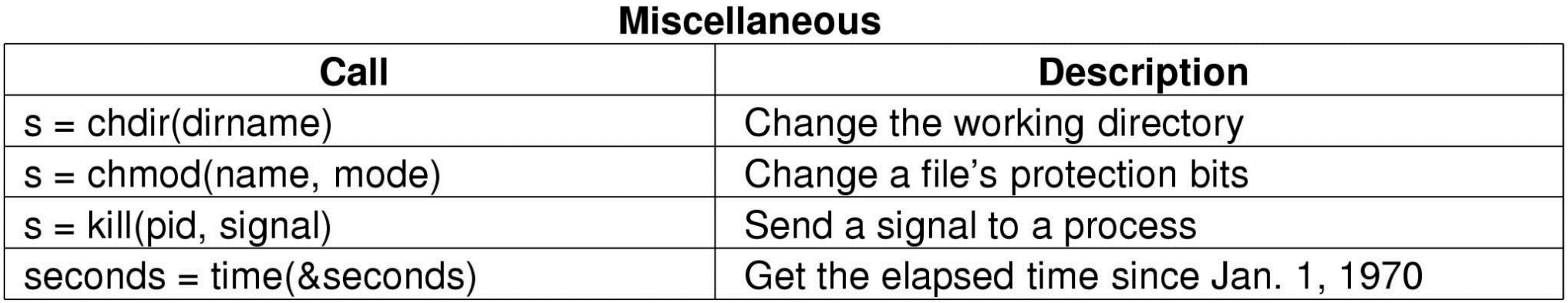
Some API Calls For Directory Management



Directory and file system management

Call	Description
s = mkdir(name, mode)	Create a new directory
s = rmdir(name)	Remove an empty directory
s = link(name1, name2)	Create a new entry, name2, pointing to name1
s = unlink(name)	Remove a directory entry
s = mount(special, name, flag)	Mount a file system
s = umount(special)	Unmount a file system

Some API Calls For Other Tasks



Miscellaneous

Call	Description
s = chdir(dirname)	Change the working directory
s = chmod(name, mode)	Change a file’s protection bits
s = kill(pid, signal)	Send a signal to a process
seconds = time(&seconds)	Get the elapsed time since Jan. 1, 1970

POSIX and Win32 Calls Comparison

- **Only some important calls are shown**

POSIX	Win32	Description
fork	CreateProcess	Create a new process
wait	WaitForSingleObject	The parent process may wait for the child to finish
execve	--	CreateProcess = fork + execve
exit	ExitProcess	Terminate process
open	CreateFile	Create a new file or open an existing file
close	CloseHandle	Close a file
read	ReadFile	Read data from an open file
write	WriteFile	Write data into an open file
lseek	SetFilePointer	Move read/write offset in a file (file pointer)
stat	GetFileAttributesExt	Get information on a file
mkdir	CreateDirectory	Create a file directory
rmdir	RemoveDirectory	Remove a file directory
link	--	Win32 does not support “links” in the file system
unlink	DeleteFile	Delete an existing file
chdir	SetCurrentDirectory	Change working directory
chmod	SeFileSecurity	Change file mode bits (rwx)

Hands-on on System calls

- helloWorld.c
  - Compile it and save as helloWorld.o
- ltrace ./helloWorld.o (options –c, -S, -t, -T)
- strace ./helloWorld.o (options –c, -C, -t, -w)
- time ./helloWorld.o
- /usr/bin/time –v ./helloWorld.o
- getpid: procedure call and system call
- syscall: to directly invoke system calls from user-space programs
- virtual system calls (e.g.vDSO gettimeofday())

Reducing System Call Overhead

- Problem: User-kernel mode distinction poses performance barrier
  - » Crossing this hardware barrier is costly
  - » System calls take 10x-1000x more time than a regular procedure call
- Solution: Perform some system functionality in user mode itself
  - » by caching results (getpid, gettimeofday)
  - » buffering I/O operations to minimize no. of system calls made (read/write vs. fread/fwrite in API)
  - » *Libraries (DLLs)* can reduce number of system calls
    - E.g.,"vDSO" (virtual dynamic shared object) is a small library of read-only type of system calls that the kernel automatically maps into the address space of all user-space applications
      - – vDSO on x84_64 offers virtual system call __vdso_gettimeofday() for the system call gettimeofday()

https://lwn.net/Articles/771441/ & https://lwn.net/Articles/615809/

http://arkanis.de/weblog/2017-01-05-measurements-of-system-call-performance-and-overhead

Example

- A stripped down shell:

```
while (TRUE) {                                      /* repeat forever */
    type_prompt( );                                 /* display prompt */
    read_command (command, parameters)              /* input from terminal */

if (fork() != 0) {                                  /* fork off child process */
    /* Parent code */
    waitpid( -1, &status, 0);                       /* wait for child to exit */
} else {
    /* Child code */
    execve (command, parameters, 0);                /* execute command */
 }
}
```

Reading and Viewing Assignments

- Appendix A from Understanding Linux Kernel by Bovet et al
- http://www.gnu.org/software/libc/
- http://www.gnu.org/software/libc/documentation.html
- Man syscall, syscalls, intro (man –a intro), libc, etc
- https://www.kernel.org/doc/man-pages/
- https://en.wikipedia.org/wiki/Unified_Extensible_Firmware_Interface

* Professor Messer's Linux+ Training:

http://www.youtube.com/playlist?list=PLCDA423AB5CEC8FDB

http://www.youtube.com/watch?v=6eTi2qu4Fb0&feature=c4-overview&list=UUkefXKtInZ9PLsoGRtml2FQ