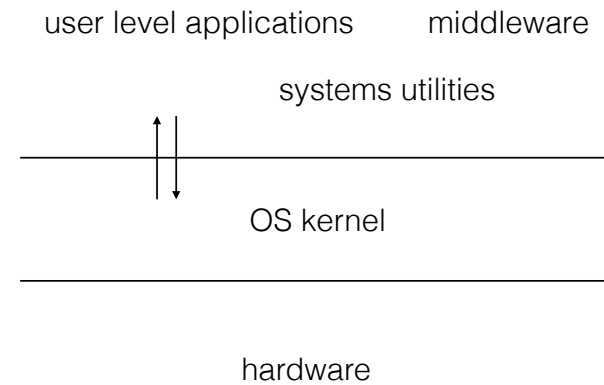


# Operating systems

Structure and services

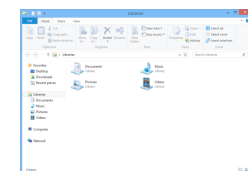
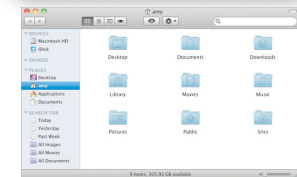
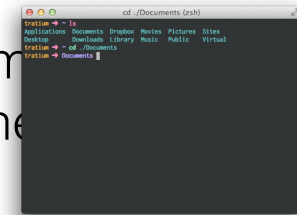


## Ex: writing from one file to another

- get input from user
- open files
- read from file
- write to file
- close files

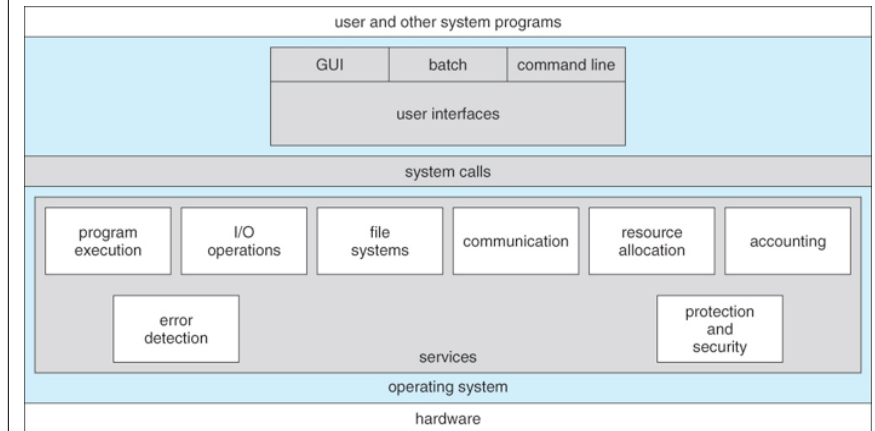
## Ex: writing from one file to another

- get input from user
- open files
- read from file
- write to file
- close files



## Ex: writing from one file to another

- get input from user                      I/O service
- open files                                      file system
- read from file
- write to file                                      errors
- close files



	Windows	Unix
Process Control	CreateProcess() ExitProcess() WaitForSingleObject()	fork() exit() wait()
File Manipulation	CreateFile() ReadFile() WriteFile() CloseHandle()	open() read() write() close()
Device Manipulation	SetConsoleMode() ReadConsole() WriteConsole()	ioctl() read() write()
Information Maintenance	GetCurrentProcessID() SetTimer() Sleep()	getpid() alarm() sleep()
Communication	CreatePipe() CreateFileMapping() MapViewOfFile()	pipe() shmget() mmap()
Protection	SetFileSecurity() InitializeSecurityDescriptor() SetSecurityDescriptorGroup()	chmod() umask() chown()

BUGS | SEE ALSO | COLOPHON

Linux Programmer's Manual

# OPEN(2)

NAME [top](#)

open, openat, creat - open and possibly create a file

SYNOPSIS [top](#)

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

int open(const char *pathname, int flags);
int open(const char *pathname, int flags, mode_t mode);

int creat(const char *pathname, mode_t mode);

int openat(int dirfd, const char *pathname, int flags);
int openat(int dirfd, const char *pathname, int flags, mode_t mode);
```

Feature Test Macro Requirements for glibc (see [feature\\_test\\_macros\(7\)](#)):

```
openat():
    Since glibc 2.10:
        _POSIX_C_SOURCE >= 200809L
    Before glibc 2.10:
        _ATFILE_SOURCE
```

DESCRIPTION [top](#)

Given a [pathname](#) for a file, [open\(\)](#) returns a file descriptor, a small, nonnegative integer for use in subsequent system calls ([read\(2\)](#), [write\(2\)](#), [lseek\(2\)](#), [fcntl\(2\)](#), etc.). The file descriptor returned by a successful call will be the lowest-numbered file descriptor not currently open for the process.

By default, the new file descriptor is set to remain open across an [execve\(2\)](#) (i.e., the [FD\\_CLOEXEC](#) file descriptor flag described in [fcntl\(2\)](#) is initially disabled); the [O\\_CLOEXEC](#) flag, described below, can be used to change this default. The file offset is set to the beginning of the file (see [lseek\(2\)](#)).

A call to [open\(\)](#) creates a new [open file description](#), an entry in the system-wide table of open files. The open file description records

Microsoft Developer resources

Windows Dev Center Explore Docs Downloads Samples Support Dashboard

File Management Functions > CreateFile

## CreateFile function

Creates or opens a file or I/O device. The most commonly used I/O devices are as follows: file, file stream, directory, physical disk, volume, console buffer, tape drive, communications resource, mailslot, and pipe. The function returns a handle that can be used to access the file or device for various types of I/O depending on the file or device and the flags and attributes specified.

To perform this operation as a transacted operation, which results in a handle that can be used for transacted I/O, use the [CreateFileTransacted](#) function.

### Syntax

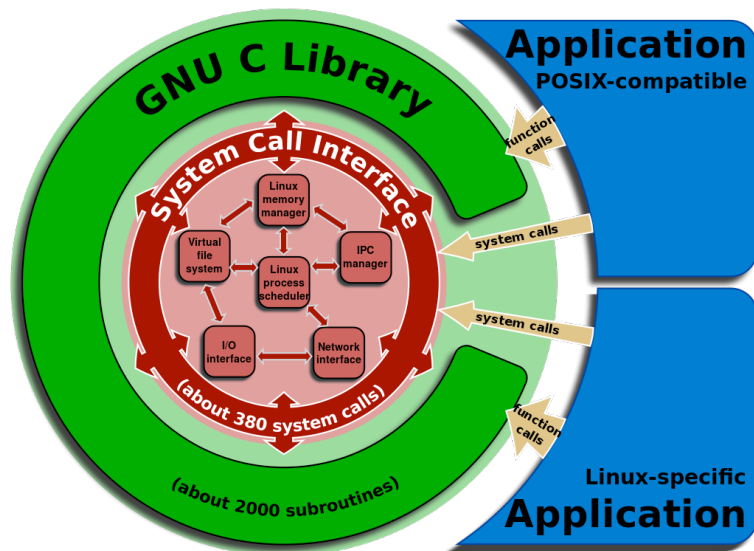
```
C++
HANDLE WINAPI CreateFile(
    _In_ LPCTSTR          lpFileName,
    _In_ DWORD            dwDesiredAccess,
    _In_ DWORD            dwShareMode,
    _In_opt_ LPSECURITY_ATTRIBUTES lpSecurityAttributes,
    _In_ DWORD            dwCreationDisposition,
    _In_ DWORD            dwFlagsAndAttributes,
    _In_opt_ HANDLE       hTemplateFile
);
```

### Parameters

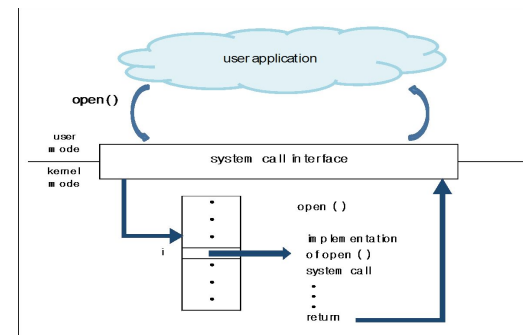
*(Parameters list is truncated in the image)*

## Common APIs

- POSIX API
  - UNIX, Linux, Mac OS X
- Win32 API
- Java API

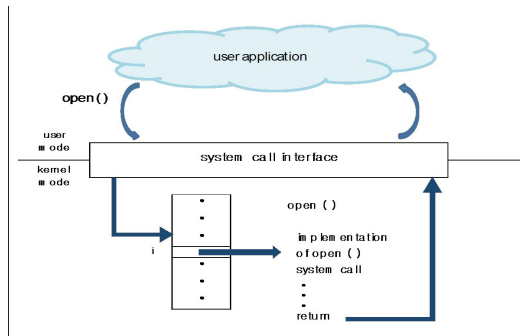


## Making a system call

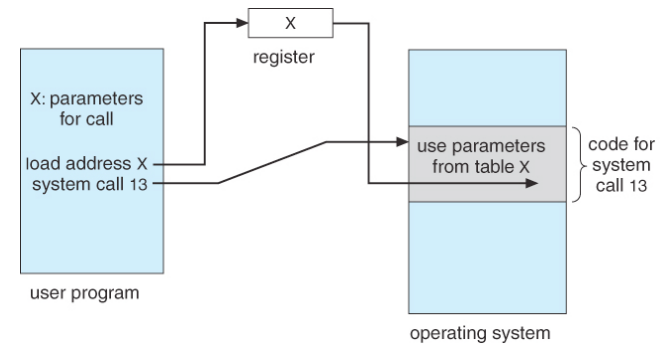


- typically in C or C++
- in assembly for low-level tasks

# Making a system call



- 3 ways to pass parameters:
- registers
- block or table
- program stack



Passing of parameters as a table

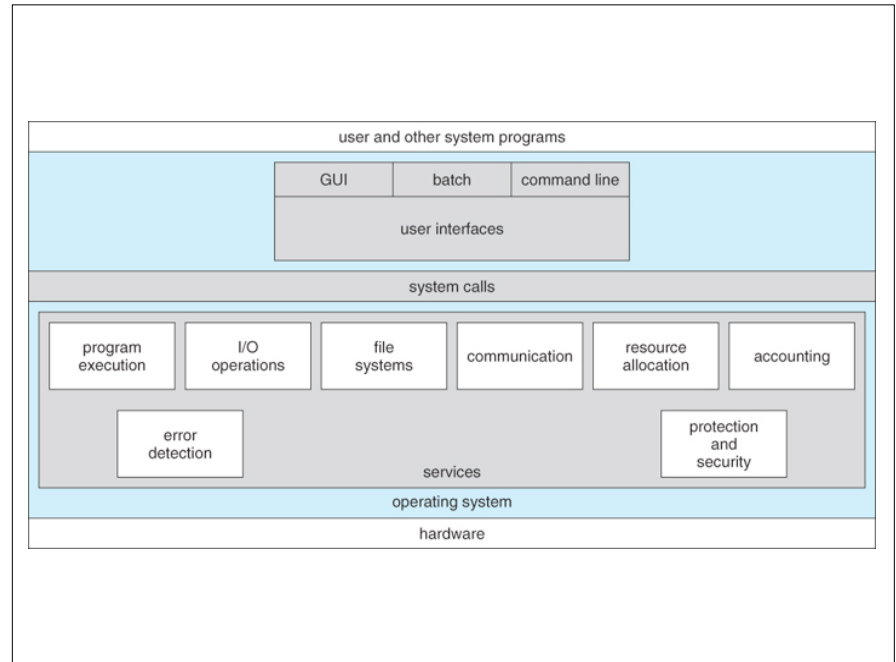
# Accessing OS services

- system calls
- libraries
- message passing
- system services

	Linux kernel-to-userspace	Linux kernel-internal
<b>API</b>	<p>✓ API stability <b>is</b> guaranteed, source code is portable!</p>	<p>✗ API stability <b>is not</b> guaranteed, source code portability is not a given</p>
<b>ABI</b>	<p>✓ compatible ABI <b>can be</b> guaranteed, binaries are portable</p> <p>compiled against LSB 5.0 for x86-64</p> <p>compiled against LSB 5.0 for x86-64</p>	<p>✗ <b>no</b> stable ABI over Linux kernel releases, binaries are not portable</p> <p>binary device driver compiled for Linux kernel 3.0</p> <p>in Linux kernel 3.14</p> <p>in Linux kernel 3.7</p> <p>in Linux kernel 3.0</p>

# Interoperability

- Can't test all applications on all platforms
- Focus on well-designed interfaces
- Test for compliance
- APIs need must be stable, backwards compatible



# User interface

- Batch processing
- Command-line
- GUI

# GUIs in Linux

- desktop environment (KDE, gnome, Unity, XFCE)
- WIMP: window, icon, menu, pointer
- modularity: ex choose your favorite lock-screen
- display server: responsible for input output
- window manager: tiling (i3, monad, open box)
- compositor (Compiz)