Operating System Concepts

lynxbee.com

What is Operating System ?

lynxbee.com

An operating system acts as an intermediary between the user of a computer and the computer hardware.

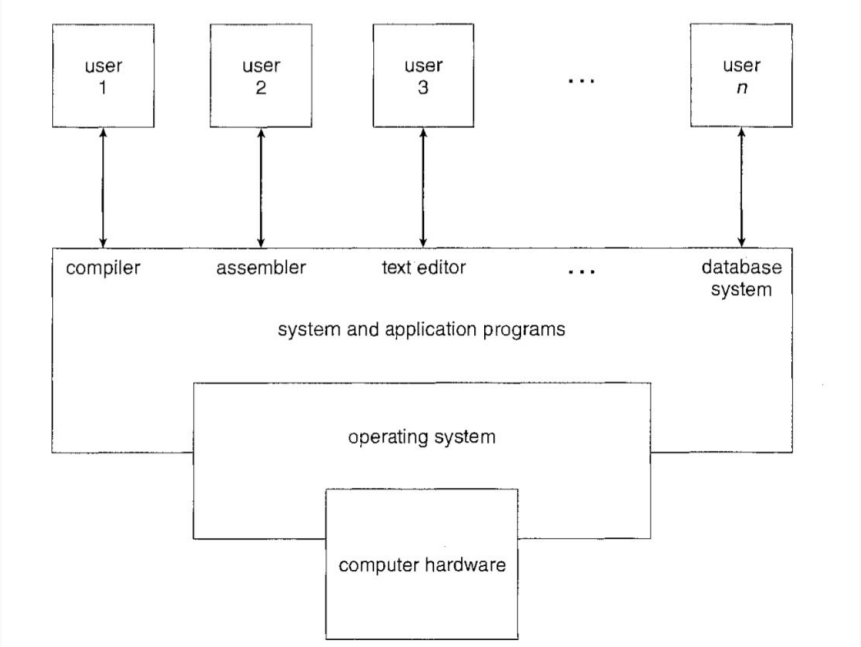
An operating system is software that manages the computer hardware.

The operating system acts as the manager of all resources.

Kernel, System Programs & Application Programs

Examples - Linux, Windows, Unix, Mac OS X,

Every computer needs to have initial program called “bootstrap program” in bootROM.



Storage Devices / Memory

Main Memory / RAM and Secondary Memory / Hard Disc

The main differences among the various storage systems lie in speed, cost, size, and volatility.

As speed increases, cost also increase but size reduces. Hence, RAM’s are costly than hard discs.

Single and Multi Processor Systems

Single Processor Systems & Multi Processor / Multi Core systems

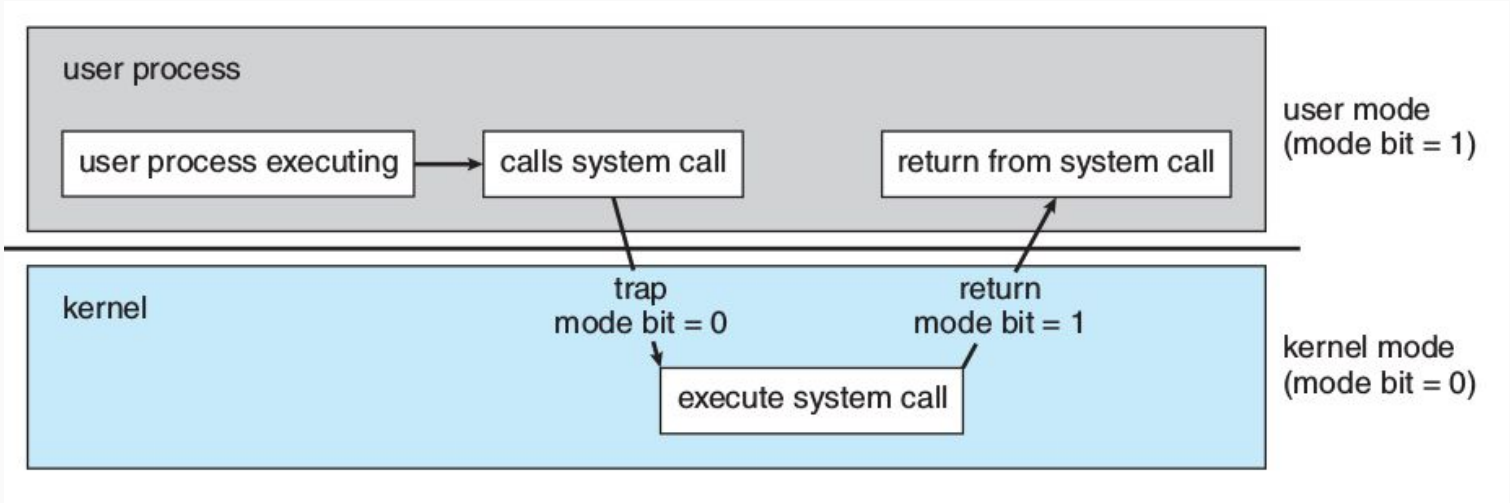
Advantages of Multiprocessor Systems

- Increased throughput
- Multiprocessor systems can cost less than equivalent multiple single-processor systems.
- failure of one processor will not halt the system, only slow it down
- Multicore - multiple computing cores on a single chip

Transition from user to Kernel Mode

A program loaded into memory and executing is called a process.

In order to ensure the proper execution of the operating system, we must be able to distinguish between the execution of operating-system code and user defined code.



System Calls

System calls provide the means for a user program to ask the operating system to perform tasks reserved for the operating system on the user program's behalf.

specific "syscall" instruction to invoke a system call.

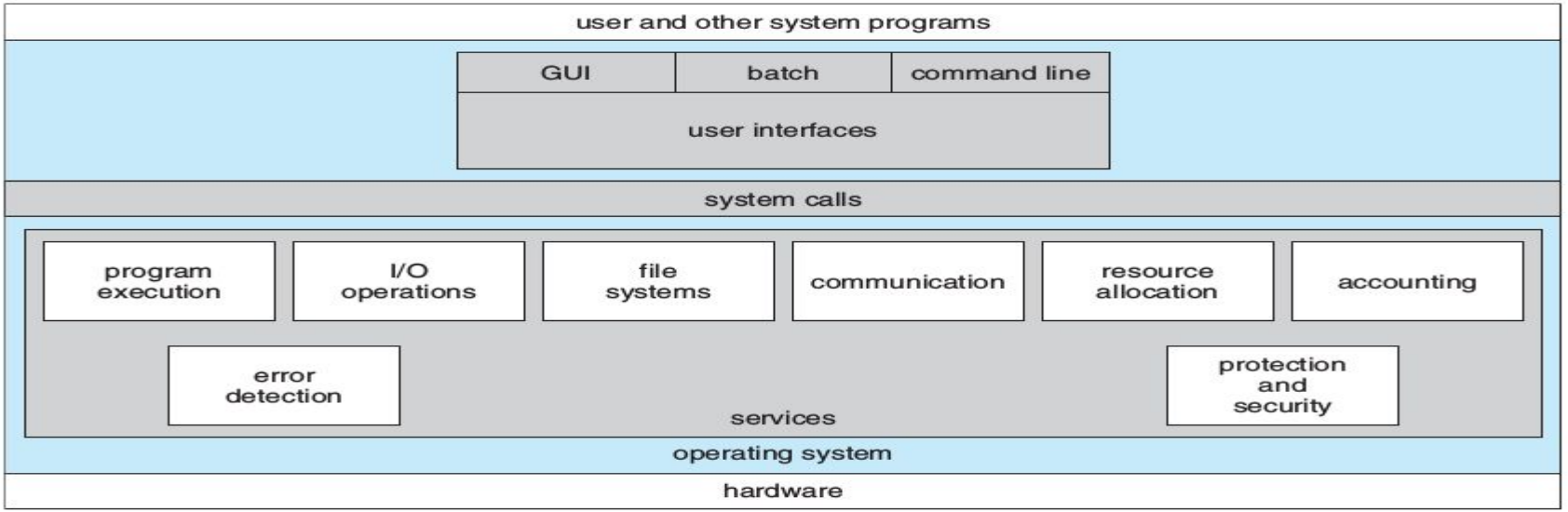
Process Management
Memory Management
Storage Management
Protection and Security - (user Id & group Id)
Kernel Data Structures

- Linked List (<linux/list.h>)
- Stack
- Queue (kfifo.c)
- Trees (<linux/rbtree.h>)
- Hash functions & maps
- bitmaps

Operating System Structures

An operating system provides an environment for the execution of programs.

Os Services



Use and Operating System Interface

Use and Operating System Interface

- command-line interface
  Sh & Bash
- Graphical user interface (GUI)

System Calls - provide an interface to the services made available by an operating System to the user applications.

API - application developers design programs according to an application programming interface (API). The API specifies a set of functions that are available to an application programmer, including the parameters that are passed to each function and the return values the programmer can expect.

A programmer accesses an API via a library of code provided by the operating system. In the case of UNIX and Linux for programs written in the C language, the library is called libc.

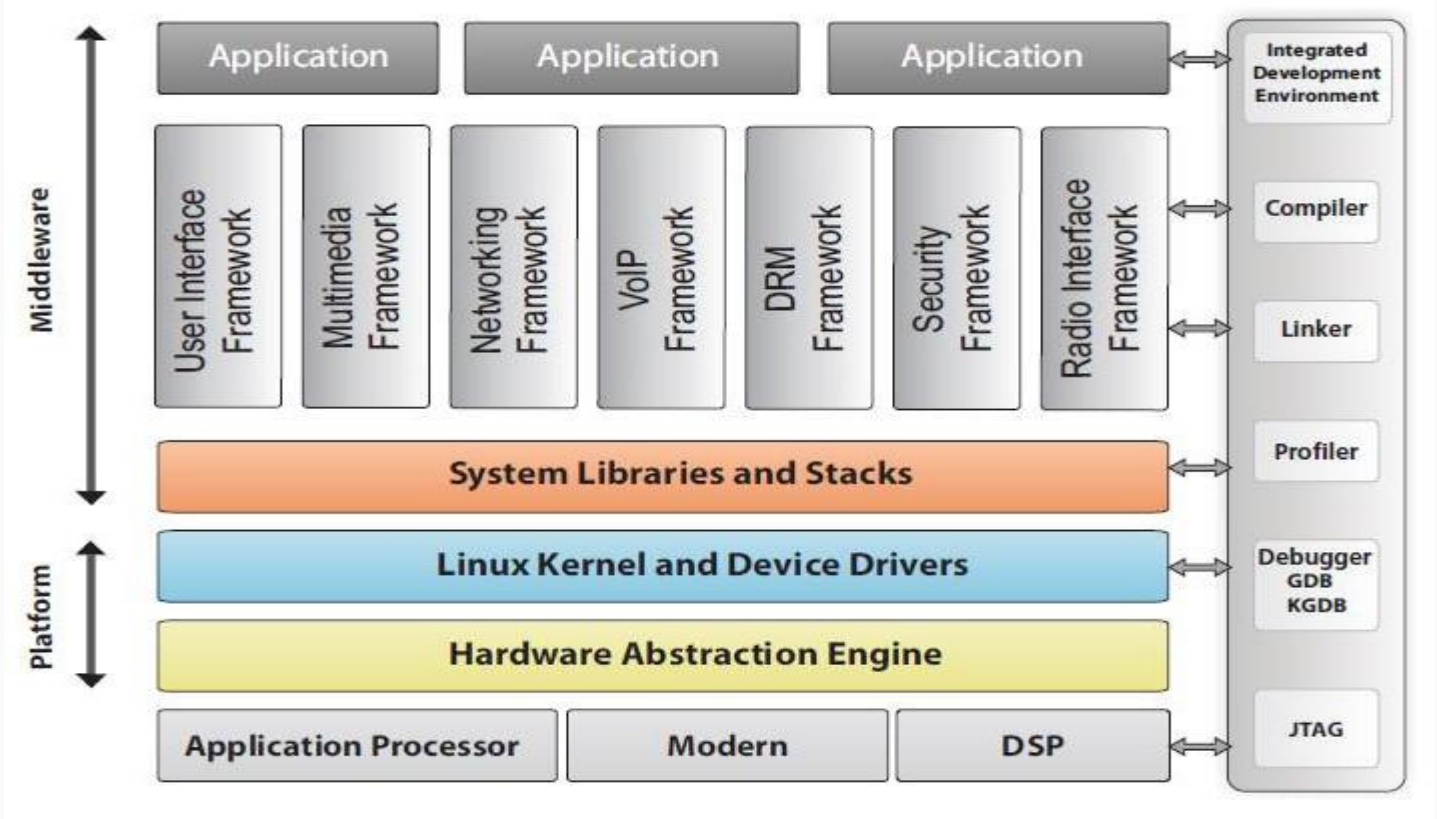
Behind the scenes, the functions that make up an API typically invoke the actual system calls on behalf of the application programmer.

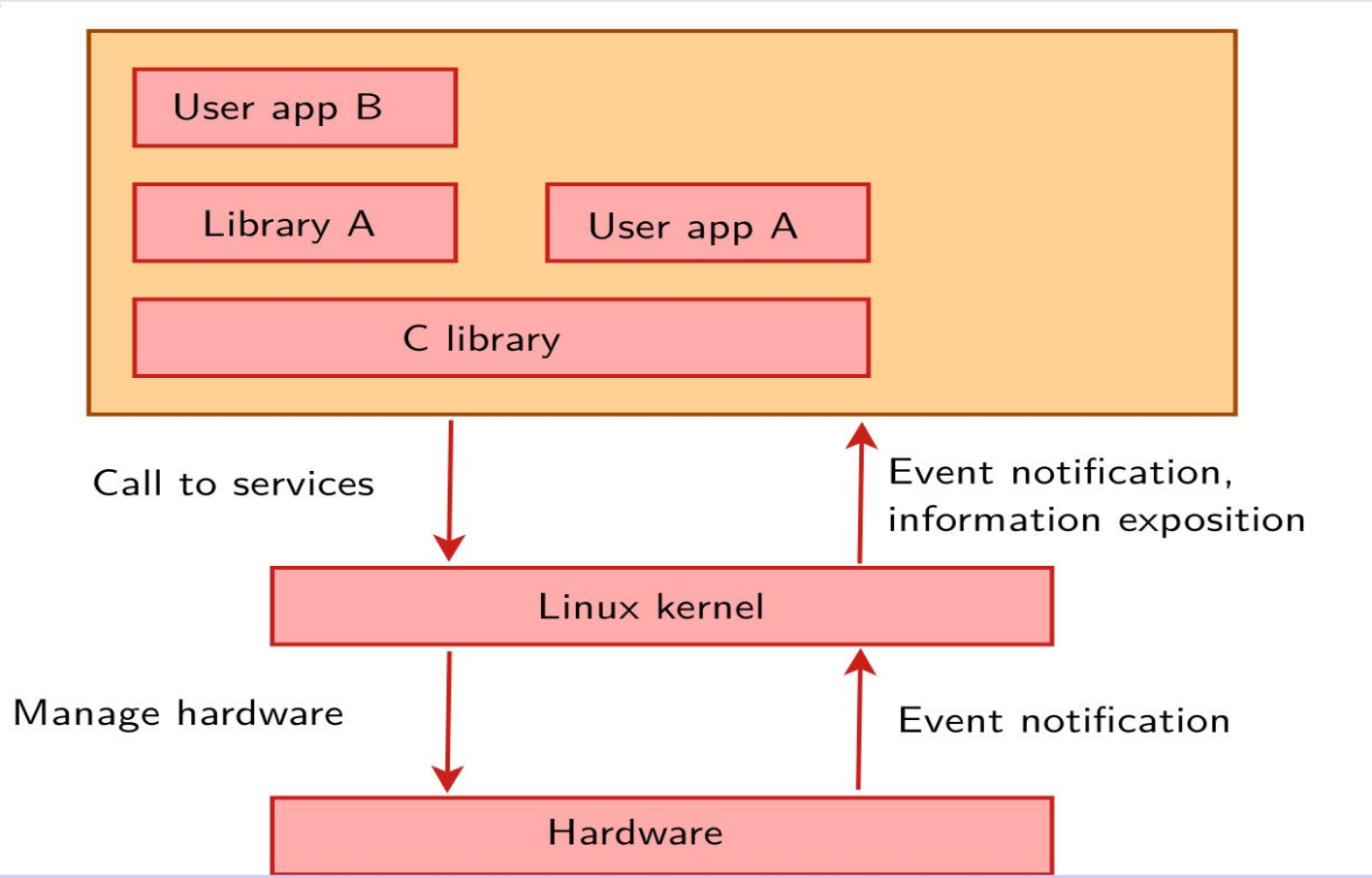
Types of System Calls

- Process control,
  - fork()
  - exit()
  - wait()
- file manipulation
  - open()
  - read()
  - write()
  - close()
- device manipulation
  - ioctl()
  - read()
  - write()
- Information Maintenance
  - getpid()
  - alarm()
  - sleep()
- Communications
  - pipe()
  - shm open()
  - mmap()
- Protection
  - chmod()
  - umask()
  - chown()

Embedded Linux Stack Diagram

lynxbee.com

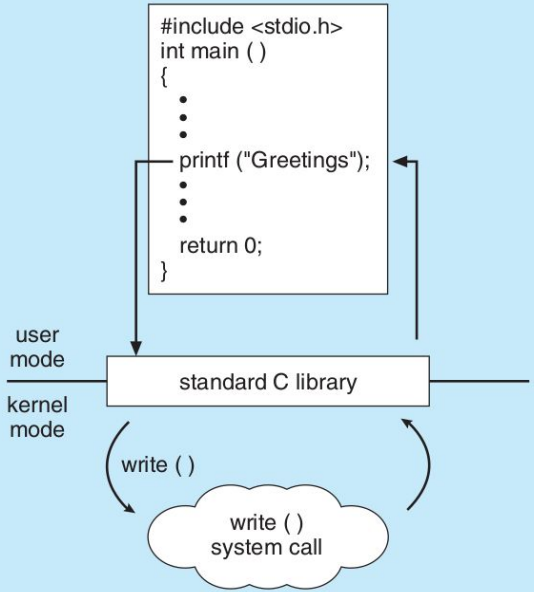



User app B
Library A
User app A
C library
Call to services
Event notification,
information exposition
Linux kernel
Manage hardware
Event notification
Hardware

Example of standard C Library

EXAMPLE OF STANDARD C LIBRARY

The standard C library provides a portion of the system-call interface for many versions of UNIX and Linux. As an example, let's assume a C program invokes the `printf()` statement. The C library intercepts this call and invokes the necessary system call (or calls) in the operating system—in this instance, the `write()` system call. The C library takes the value returned by `write()` and passes it back to the user program. This is shown below:

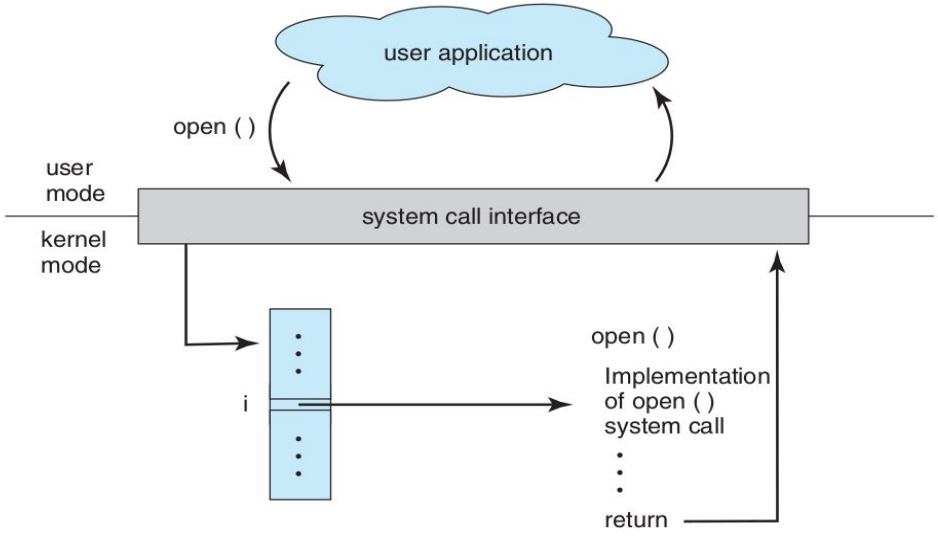


System call interface

Why to use API & not System calls ?

- Portability

System Call Interface



The handling of a user application invoking the `open()` system call.

Program to Open File & Read


```
int main(void) {
	FILE *fp;
	char ch;

	fp = fopen(FILENAME, "r");
	if (fp == NULL) {
		printf("file %s is not present, please
check\n", FILENAME);
		return -1;
	}

	while((ch = fgetc(fp)) != EOF) {
		printf("%c", ch);
	}

	return 0;
}
```

Understanding program execution

lynxbee.com

```
$ strace ./a.out sample_text_file.txt
execve("./a.out", ["./a.out", "sample_text_file.txt"], [/* 69 vars */]) = 0
brk(NULL)                               = 0x995c000
access("/etc/ld.so.nohwcap", F_OK)      = -1 ENOENT (No such file
or directory)
mmap2(NULL, 8192, PROT_READ|PROT_WRITE,
MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0xb77d7000
access("/etc/ld.so.preload", R_OK)      = -1 ENOENT (No such file or
directory)
open("/etc/ld.so.cache", O_RDONLY|O_CLOEXEC) = 3
fstat64(3, {st_mode=S_IFREG|0644, st_size=169579, ...}) = 0
mmap2(NULL, 169579, PROT_READ, MAP_PRIVATE, 3, 0) =
0xb77ad000
close(3)                                = 0
access("/etc/ld.so.nohwcap", F_OK)      = -1 ENOENT (No such file
or directory)
open("/lib/i386-linux-gnu/libc.so.6", O_RDONLY|O_CLOEXEC) = 3
read(3,
"\177ELF\1\1\1\3\0\0\0\0\0\0\0\0\3\0\3\0\1\0\0\0\320\207\1\0004\0\0\
0"..., 512) = 512
fstat64(3, {st_mode=S_IFREG|0755, st_size=1786484, ...}) = 0
mmap2(NULL, 1792540, PROT_READ|PROT_EXEC,
MAP_PRIVATE|MAP_DENYWRITE, 3, 0) = 0xb75f7000
mmap2(0xb77a7000, 12288, PROT_READ|PROT_WRITE,
MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x1af000) =
0xb77a7000
mmap2(0xb77aa000, 10780, PROT_READ|PROT_WRITE,
MAP_PRIVATE|MAP_FIXED|MAP_ANONYMOUS, -1, 0) =
0xb77aa000
close(3)                                = 0
mmap2(NULL, 4096, PROT_READ|PROT_WRITE,
MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0xb77fe000
set_thread_area({entry_number:-1, base_addr:0xb77fe940,
limit:1048575, seg_32bit:1, contents:0, read_exec_only:0,
limit_in_pages:1, seg_not_present:0, useable:1}) = 0 (entry_number:6)
mprotect(0xb77a7000, 8192, PROT_READ)   = 0
mprotect(0x8049000, 4096, PROT_READ)    = 0
mprotect(0xb77ff000, 4096, PROT_READ)   = 0
munmap(0xb77ad000, 169579)              = 0
brk(NULL)                               = 0x995c000
brk(0x997d000)                          = 0x997d000
open("sample_text_file.txt", O_RDONLY)  = 3
fstat64(3, {st_mode=S_IFREG|0664, st_size=92, ...}) = 0
read(3, "This is a contents from the samp"..., 4096) = 92
fstat64(1, {st_mode=S_IFCHR|0620, st_rdev=makedev(136, 1), ...}) =
0
write(1, "This is a contents from the samp"..., 46This is a contents from
the sample text file.
) = 46
write(1, "We will use this to read from mu"..., 46We will use this to read
from multiline file.
) = 46
read(3, "", 4096)                       = 0
exit_group(0)                           = ?
```

Visit

lynxbee.com