System Calls, fork(), exec()

Abhijit A. M.
abhijit.comp@coep.ac.in

Credits: Slides of “OS Book” ed10.

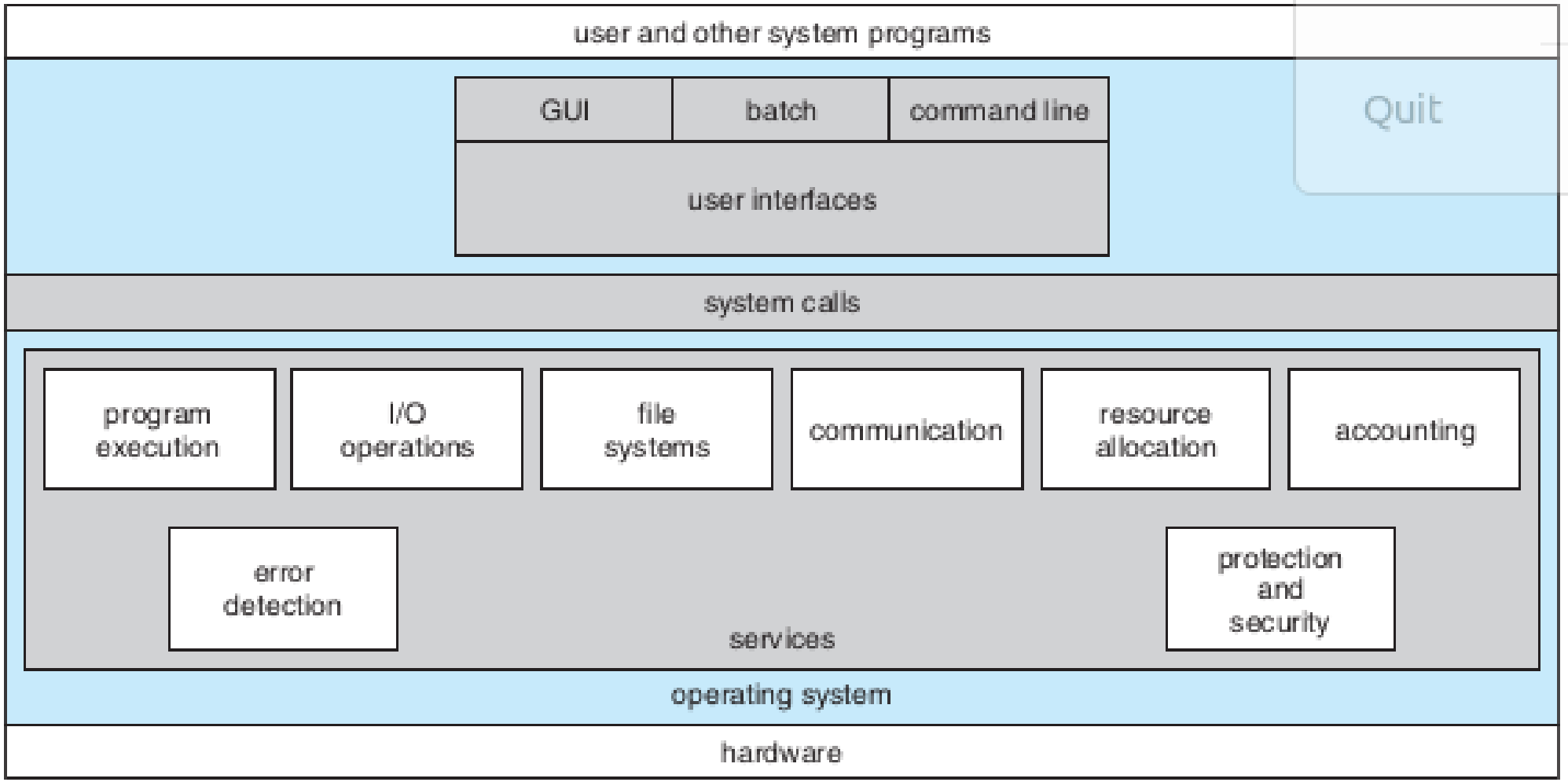


Figure 2.1 A view of operating system services.

System Calls

- **Services provided by operating system to applications**
  - **Essentially available to applications by calling the particular software interrupt application**
    - **All system calls essentially involve the “INT 0x80” on x86 processors + Linux**
    - **Different arguments specified in EAX register inform the kernel about different system calls**
- **The C library has wrapper functions for each of the system calls**
  - **E.g. open(), read(), write(), fork(), mmap(), etc.**

Types of System Calls

- **File System Related**
  - **Open(), read(), write(), close(), etc.**
- **Processes Related**
  - **Fork(), exec(), ...**
- **Memory management related**
  - **Mmap(), shm_open(), ...**
- **Device Management**
- **Information maintainance – time,date**
- **Communication between processes (IPC)**
- **Read man syscalls**

https://linuxhint.com/list_of_linux_syscalls/

EXAMPLES OF WINDOWS AND UNIX SYSTEM CALLS

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

```
int main() {
    int a = 2;
    printf("hi\n");
}
-----
C Library
-----
int printf("void *a, ...) {
    ...
    write(1, a, ...);
}
int write(int fd, char *, int len) {
    int ret;
    ...
    mov $5, %eax,
    mov ... %ebx,
    mov ..., %ecx
    int $0x80
    __asm__("movl  %eax, -4(%ebp)");
# -4ebp is ret
    return ret;
}
```

Code schematic

-------user-kernel-mode-boundary----

```
//OS code
int sys_write(int fd, char *, int len) {
    figure out location on disk
    where to do the write and
    carry out the operation,
    etc.
}
```

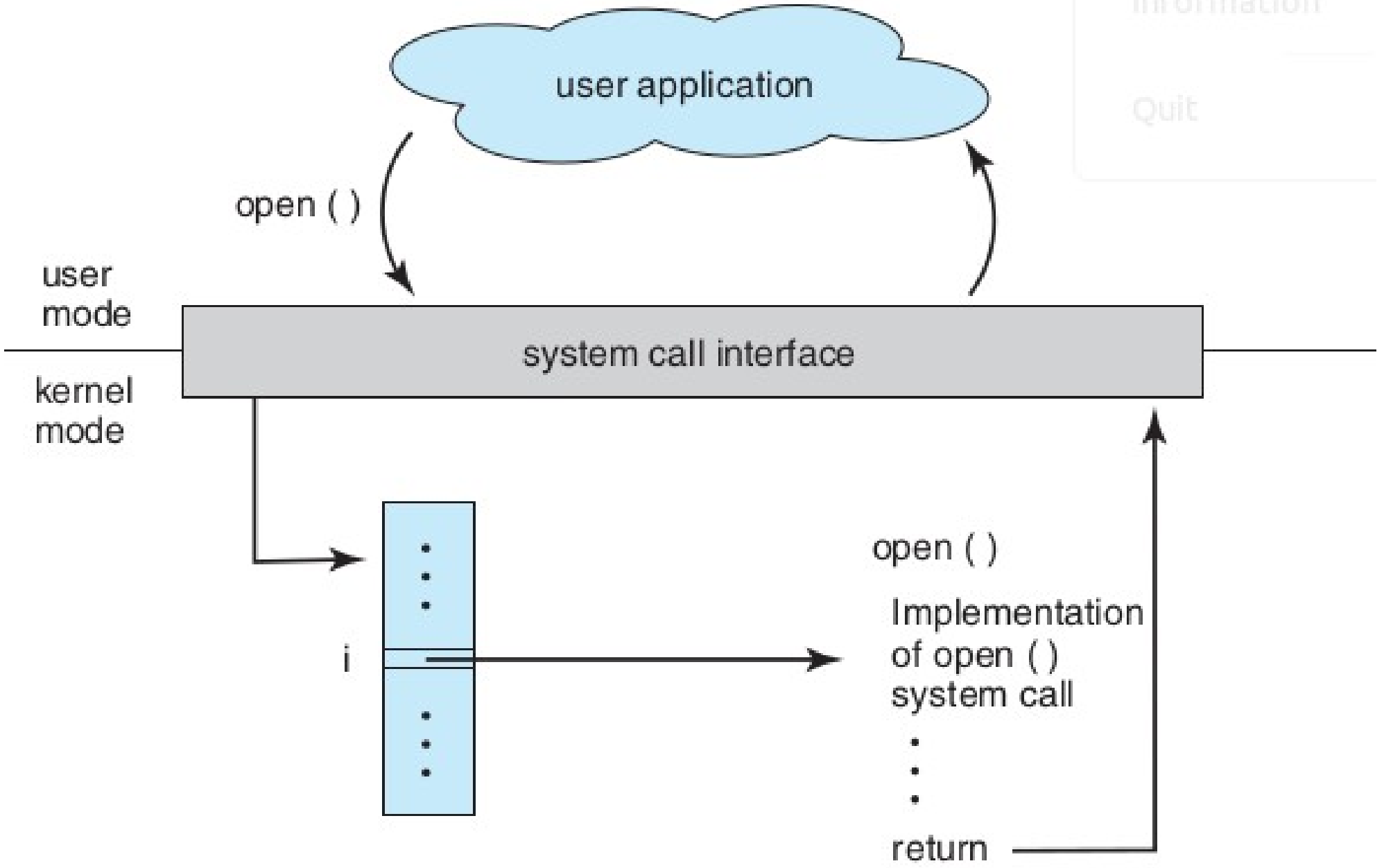


Figure 2.6 The handling of a user application invoking the `open()` system call.

Two important system calls
Related to processes

fork() and exec()

Process

- A program in execution
- Exists in RAM
- Scheduled by OS
  - In a timesharing system, intermittantly scheduled by allocating a time quantum, e.g. 20 microseconds
- The “ps” command on Linux

Process in RAM

- **Memory is required to store the following components of a process**
  - **Code**
  - **Global variables (data)**
  - **Stack (stores local variables of functions)**
  - **Heap (stores malloced memory)**
  - **Shared libraries (e.g. code of printf, etc)**
  - **Few other things, may be**

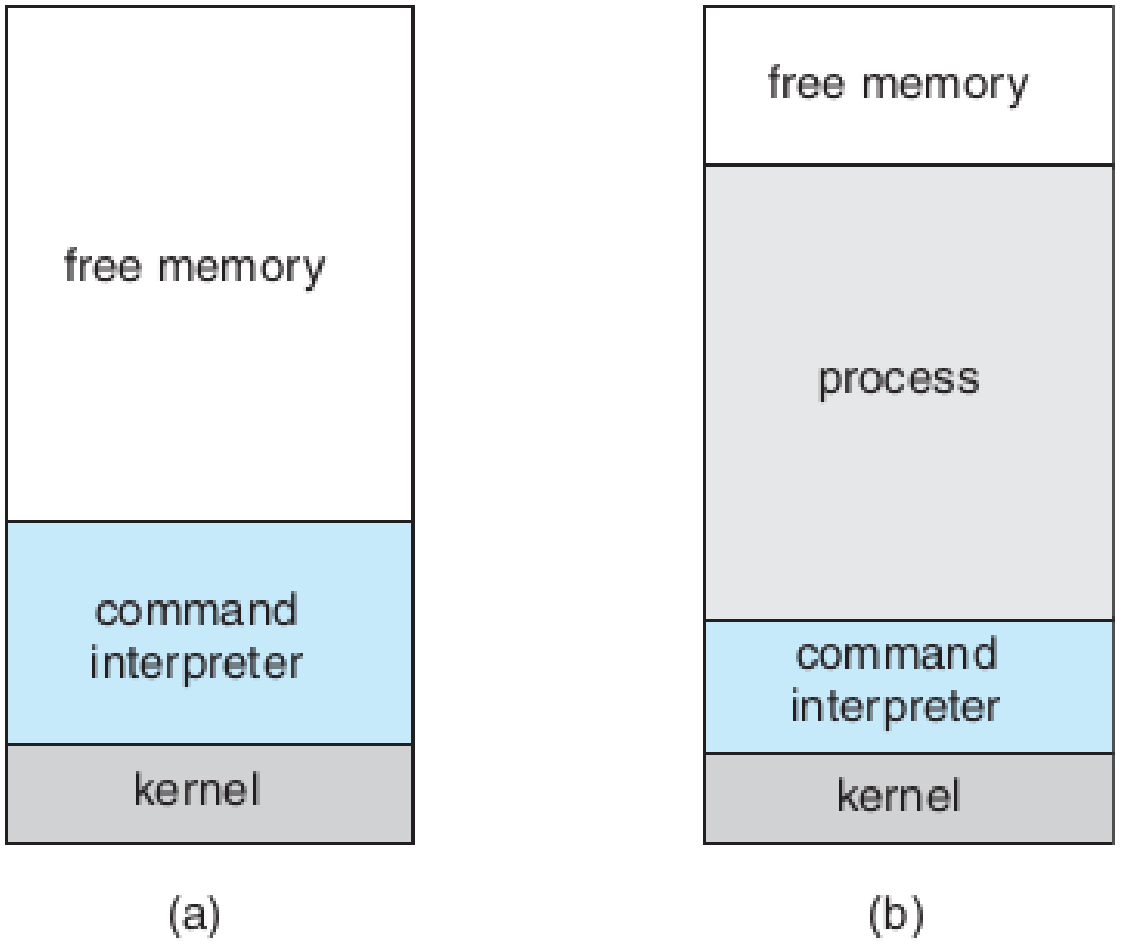
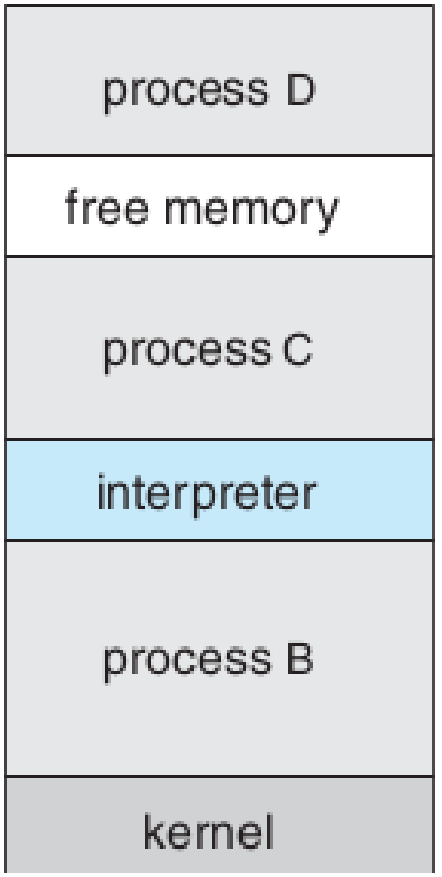


Figure 2.9 MS-DOS execution. (a) At system startup. (b) Running a program.

MS-DOS: a single tasking operating system
Only one program in RAM at a time, and only one program can run at a time



A multi tasking system
With multiple programs loaded in memory
Along with kernel
(A very simplified conceptual diagram. Things are more complex in reality)

fork()

- A running process creates it’s duplicate!
- After call to fork() is over
  - Two processes are running
  - Identical
  - The calling function returns in two places!
  - Caller is called parent, and the new process is called child
  - PID is returned to parent and 0 to child

exec()

- Variants: execvp(), execl(), etc.
- Takes the path name of an executable as an argument
- Overwrites the existing process using the code provided in the executable
- The original process is OVER ! Vanished!
- The new program starts running overwritting the existing process!

Shell using fork and exec

- **Demo**
- **The only way a process can be created on Unix/Linux is using fork() + exec()**
- **All processes that you see were started by some other process using fork() + exec() , except the initial *"init" process***
- ***When you click on "firefox" icon, the user-interface program does a fork() + exec() to start firefox; same with a command line shell program***
- **The "bash" shell you have been using is nothing but an advanced version of the shell code shown during the demo**
- **See the process tree starting from "init"**
- **Your next assignment**

The boot process, once again

- BIOS
- Boot loader
- OS – kernel
- Init created by kernel by Hand(kernel mode)
- Kernel schedules init (the only process)
- Init fork-execs some programs (user mode)
  - Now these programs will be scheduled by OS
- Init -> GUI -> terminal -> shell
  - One of the typical parent-child relationships