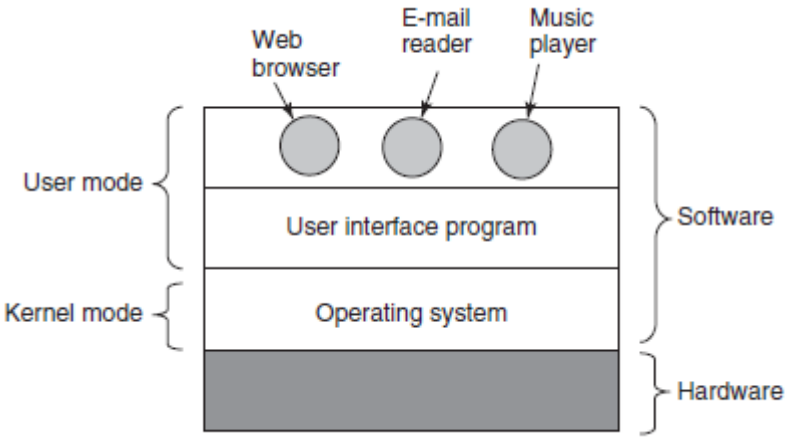
Modern Operating Systems

Introduction

On top of the hardware is the software. Most computers have two modes of operation: kernel mode and user mode. The operating system, the most fundamental piece of software, runs in **kernel mode** (also called **supervisor mode**). In this mode it has complete access to all the hardware and can execute any instruction the machine is capable of executing. The rest of the software runs in **user mode**, in which only a subset of the machine instructions is available. In particular, those instructions that affect control of the machine or do **I/O**)**Input**/Output" are forbidden to user-mode programs.



The user interface program, shell or GUI.

The concept of an operating system as primarily providing abstractions to application programs is a top-down view. An alternative, bottom-up, view holds that the operating system is there to manage all the pieces of a complex system. Modern computers consist of processors, memories, timers, disks, mice, network interfaces, printers, and a wide variety of other devices. In the bottom-up view, the job of the operating system is to provide for an orderly and controlled allocation of the processors, memories, and I/O devices among the various programs wanting them.

Os resouce management inclueds multiplexing (sharing) resources in two different ways: time and space. Time (whic app to run and how long) Space (access to memeory)

History of OS

First Generation: Vacuum Tubes (1945 - 1955)
Primitive and took seconds to perform simple calculation
No programming languages
Punched cards

Second Generation: Transisistors (1955 – 1965)
Intro into mainframes for large organizations (batch jobs)
Fortran and assembler
First os: Fortan Monitor System and IBSYS (IBM's os)

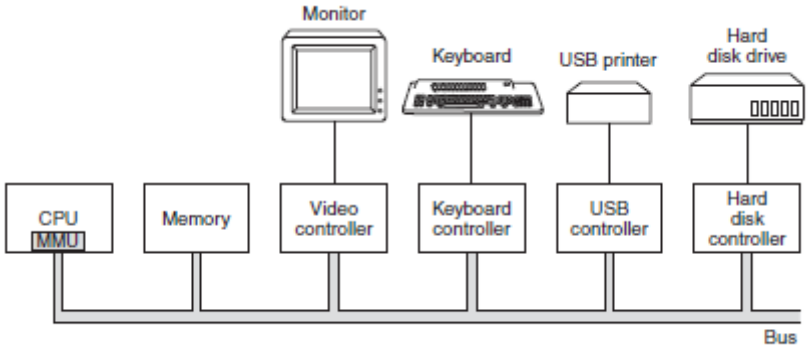
The Third Generation (1965 – 1980) IC and Multiprogramming
IBM uses IC . Intergraed Circuits and OS/360 (introductions several key techniques as multiprogramming, partition memory, timesharing
MULTICS cmputers come back in the form cloud compuutin, influence on UNIX os. On Unix is created MINIX and then Linux Glyn Moody’s (2001) book history of Unix

The Fourth Generation (1980 . now) Persona Computers
Large Scale Integration circuits

Intel 8080 and CP/M (Control Program for Microcomputers) operating system.
IBM design IBM pc and nedd software ro run on it. People from IBM contacted Bill Gates to license his BASIC interpreter. Gates realized that a local computer manufacturer, Seattle Computer Products, had a suitable operating system, **DOS** (**Disk Operating System**). He approached them and asked to buy it (allegedly for $75,000), Gates then offered IBM a DOS/BASIC package
1960 Doug Engelbar inventd Graphical User Interface

The Fifth Generation (1990 -) Moblie Computers
At the time of writing, Google's Android is the dominant operating system with Apple's iOS a clear second

Computer hardware review



The ''brain'' of the computer is the CPU. It fetches instructions from memory and executes them. The basic cycle of every CPU is to fetch the first instruction from memory, decode it to determine its type and operands, execute it.
The **program counter**, which contains the memory address of the next instruction to be fetched.
Stack pointer, which points to the top of the current stack in memory.
PSW (**Program Status Word**). This register contains the condition code bits, which are set by comparison instructions (the CPU priority, the mode (user or kernel), and various other control bits).

The operating system will often stop the running program to (re)start another one. Every time it stops a running program, the operating system must save all the registers so they can be restored when the program runs later.

Many modern CPUs have facilities for executing more than one instruction at the same time. For example, a CPU might have separate fetch, decode, and execute units, so that while it is executing instruction *n*, it could also be decoding instruction $n + 1$ and fetching instruction $n + 2$. Such an organization is called a **pipeline**

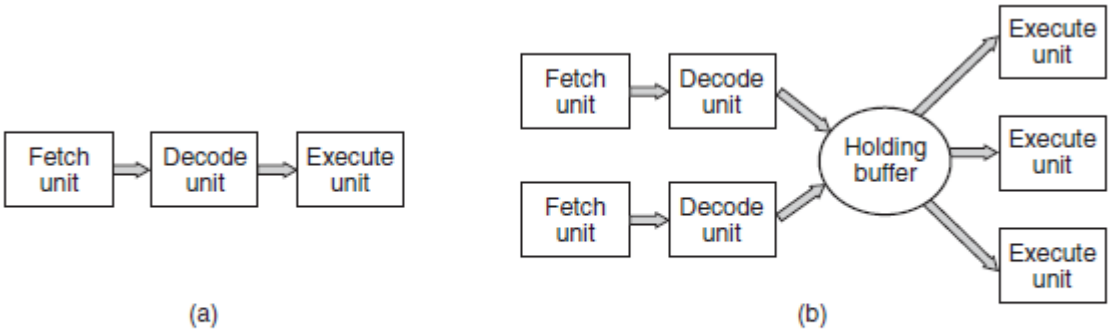


Figure 1-7. (a) A three-stage pipeline. (b) A superscalar CPU.

Even more advanced than a pipeline design is a **superscalar CPU.** multiple execution units are present, for example, one for integer arithmetic, one for floating-point arithmetic, and one for Boolean operations.
Two or more instructions are fetched at once, decoded, and dumped into a holding buffer until they can be executed.

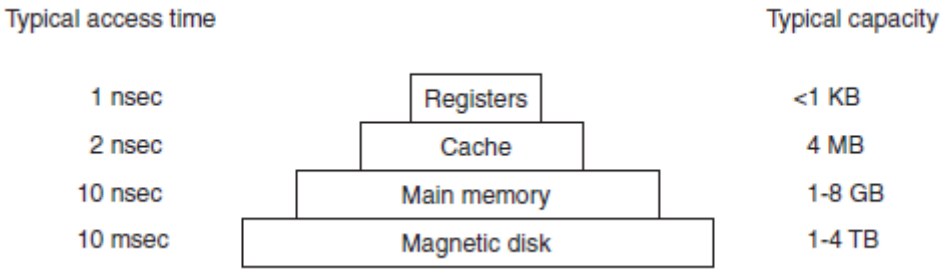
To obtain services from the operating system, a user program must make a **system call**, which traps into the kernel and invokes the operating system. The TRAP instruction switches from user mode to kernel mode and starts the operating system

Multithreading or **hyperthreading s allow the** CPU to hold the state of two different threads and then switch back and forth on a nanosecond time scale. Multithreading has implications for the operating system because each thread appears to the operating system as a separate CPU. Consider a system with two actual CPUs, each with two threads. The operating system will see this as four CPUs.

Many CPU chips now hav e four, eight, or more complete processors or **cores** on them.

GPU (**Graphics Processing Unit**). A GPU is a processor with, literally, thousands of tiny cores. They are very good for many small computations done in parallel, like rendering polygons in graphics applications. They are not so good at serial tasks

Memory system is constructed as a hierarchy of layers. Top = higer speed smaller capacity and greter cost per bit thath the lower ones.



Caching plays a major role in many areas of computer science. Whenever a resource can be divided into pieces, some of which are used much more heavily than others, caching is often used to improve performance. Caches are such a good idea that modern CPUs have two of them. The first level or **L1 cache** is always inside the CPU and usually feeds decoded instructions into the CPU's execution engine. **L2 cache**, that holds several megabytes of recently used memory words. The difference between the L1 and L2 caches lies in the timing. Access to the L1 cache is done without any delay, whereas access to the L2 cache involves a delay of one or two clock cycles.

Virtual memory makes it possible to run programs larger than physical memory by placing them on the disk and using main memory as a kind of cache for the most heavily executed parts. This scheme requires remapping memory addresses on the fly to convert the address the program generated to the physical address in RAM where the word is located. This mapping is done by a part of the CPU called the **MMU** (**Memory Management Unit**).

In a multiprogramming system, when switching from one program to another, sometimes called a **context switch**, it may be necessary to flush all modified blocks from the cache and change the mapping registers in the MMU. Both of these are expensive operations, and programmers try hard to avoid them.

I/O devices generally consist of two parts: a controller and the device itself. The controller is a chip or a set of chips that physically controls the device. It accepts commands from the operating system, for example, to read dana from the device, and carries them out. In many cases, the actual control of the device is complicated and detailed, so it is the job of the controller to present a simpler (but still very complex) interface to the operating system. The software that talks to a controller, giving it commands and accepting responses, is called a **device driver**. Each controller manufacturer has to supply a driver for each operating system it supports.
There are three ways the driver can be put into the kernel. The first way is to relink the kernel with the new driver and then reboot the system. Many older UNIX systems work like this. The second way is to make an entry in an operating system file telling it that it needs

the driver and then reboot the system. At boot time, the operating system goes and finds the drivers it needs and loads them. Windows works this way. The third way is for the operating system to be able to accept new drivers while running and install them on the fly without the need to reboot.

Input and output can be done in three different ways. In the simplest method, auser program issues a system call, which the kernel then translates into a procedure call to the appropriate driver. The driver then starts the I/O and sits in a tight loop continuously polling the device to see if it is done (usually there is some bit that indicates that the device is still busy). When the I/O has completed, the driver puts the data (if any) where they are needed and returns. The operating system then returns control to the caller. This method is called **busy waiting** and has the disadvantage of tying up the CPU polling the device until it is finished.
The second method is for the driver to start the device and ask it to give an interrupt when it is finished. At that point the driver returns. The operating system then blocks the caller if need be and looks for other work to do. When the controller detects the end of the transfer, it generates an **interrupt** to signal completion.

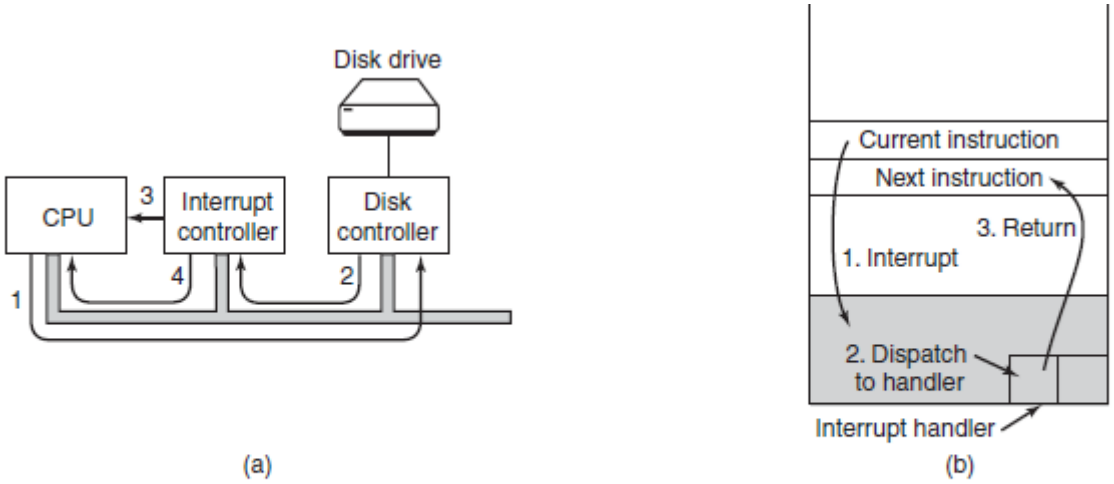


Figure 1-11. (a) The steps in starting an I/O device and getting an interrupt. (b) Interrupt processing involves taking the interrupt, running the interrupt handler, and returning to the user program.

The third method for doing I/O makes use of special hardware: a **DMA** (**Direct Memory Access**) chip that can control the flow of bits between memory and some controller without constant CPU intervention. The CPU sets up the DMA chip, telling it how many bytes to transfer, the device and memory addresses involved, and the direction, and lets it go. When the DMA chip is done, it causes an interrupt, which is handled as described above.

Interrupts can (and often do) happen at highly inconvenient moments, for example, while another interrupt handler is running. For this reason, the CPU has a way to disable interrupts and then reenable them later.

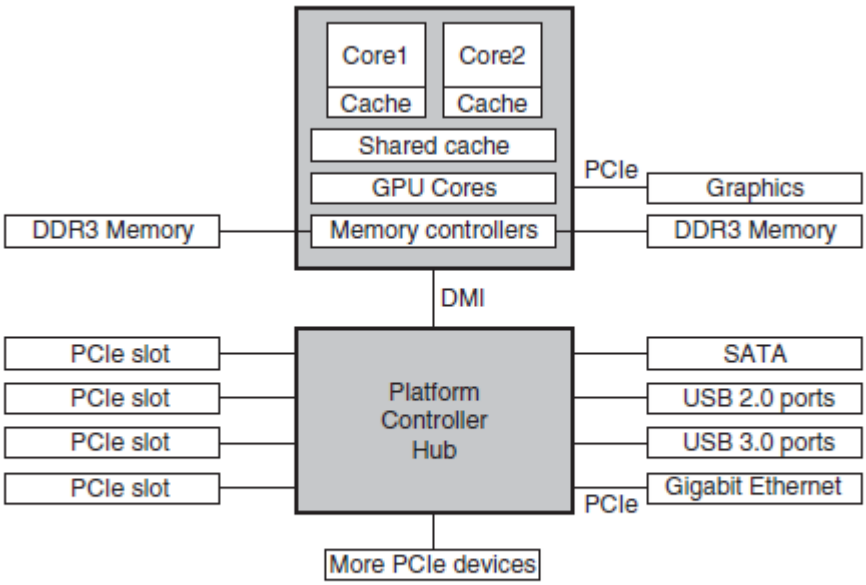


Figure 1-12. The structure of a large x86 system.

This system has many buses (e.g., cache, memory, PCIe, PCI, USB, SATA, and DMI), each with a different transfer rate and function. The operating system must be aware of all of them for configuration and management. The main bus is the **PCIe** (**Peripheral Component Interconnect Express**) bus.

This system has many buses (e.g., cache, memory, PCIe, PCI, USB, SATA, and DMI), each with a different transfer rate and function. The operating system must be aware of all of them for configuration and management. The main bus is the **PCIe** (**Peripheral Component Interconnect Express**) bus.

In regular PCI buses, a single 32-bit number is sent over 32 parallel wires. In contrast to this, PCIe uses a **serial bus architecture** and sends all bits in a message through a single connection, known as a lane, much like a network packet. Parallelism is still used, because you can have multiple lanes in parallel. For instance, we may use 32 lanes to carry 32 messages in parallel.

The CPU talks to memory over a fast DDR3 bus, to an external graphics device over PCIe and to all other devices via a hub over a **DMI** (**Direct Media Interface**) bus. Each of the cores has a dedicated cache and a much larger cache that is shared between them. Each of these caches introduces another bus.

USB uses a small connector with four to eleven wires (depending on the version), some of which supply electrical power to the USB devices or connect to ground. USB is a centralized bus in which a root device polls all the I/O devices every 1 msec to see if they hav e any traffic

Booting the Computer

Every PC contains a parentboard (formerly called a motherboard before political correctness hit the computer industry). On the parentboard is a program called the system **BIOS** (**Basic Input Output System**). The BIOS contains low-level I/O software, including procedures to read the keyboard, write to the screen, and do disk I/O, among other things
When the computer is booted, the BIOS is started. It first checks to see how much RAM is installed and whether the keyboard and other basic devices are installed and responding correctly. It starts out by scanning the PCIe and PCI buses to detect all the devices attached to them. If the devices present are different from when the system was last booted, the new devices are configured.
The BIOS then determines the boot device by trying a list of devices stored in the CMOS memory.

The first sector from the boot device is read into memory and executed. This sector contains a program that

normally examines the partition table at the end of the boot sector to determine which partition is active. Then a secondary boot loader is read in from that partition. This loader reads in the operating system from the active partition and starts it.

OS queries the BIOS to get the configuration information. For each device, it checks to see if it has the device driver. Once it has all the device drivers, the operating system loads them into the kernel. Then it initializes its tables, creates whatever background processes are needed, and starts up a login program or GUI

OS concepts

A process is basically a program in execution. Associated with each process is its **address space**, a list ofmemory locations from 0 to some maximum, which the process can read and write. The address space contains the executable program, the program's data, and its stack. Also associated with each process is a set of resources, commonly including registers (including the program counter and stack pointer), a list of open files, outstanding
alarms, lists of related processes, and all the other information needed to run the program. A process is fundamentally a container that holds all the information needed to run a program.

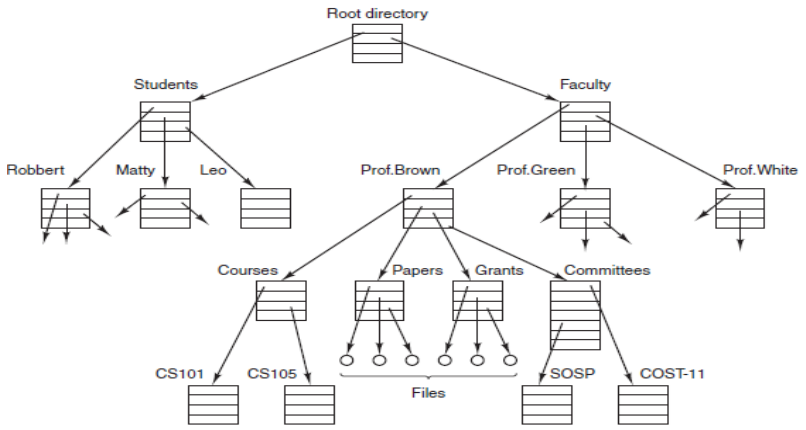
In many operating systems, all the information about each process, other than the contents of its own address space, is stored in an operating system table called the **process table**, which is an array of structures, one for each process currently in existence. Thus, a (suspended) process consists of its address space, usually called the **core image** and its process table entry, which contains the contents of its registers and many other items needed to restart the process later.

If a process can create one or more other processes (referred to as **child processes**) and these processes in turn can create child processes, we quickly arrive at the process tree structure.

Each person authorized to use a system is assigned a **UID** (**User Identification**) by the system administrator. Every process started has the UID of the person who started it. A child process has the same UID as its parent. Users can be members of groups, each of which has a **GID** (**Group Identification**).

Technique called virtual memory exists, as mentioned earlier, in which the operating system keeps part of the address space in main memory and part on disk and shuttles pieces back and forth between them as needed. In essence, the operating system creates the abstraction of an address space as the set of addresses a process may reference. The address space is decoupled from the machine's physical memory and may be either larger or smaller than the physical memory.

To provide a place to keep files, most PC operating systems have the concept of a **directory** as a way of grouping files together. System calls are then needed to create and remove directories. Calls are also provided to put an existing file in a directory and to remove a file from a directory. Directory entries may be either files or other directories. This model also gives rise to a hierarchy—the file system. Every file within the directory hierarchy can be specified by giving its **path name** from the top of the directory hierarchy, the **root directory**. In Windows, the backslash (\) character is used as the separator instead of the slash (/) on unix.



At every instant, each process has a current **working directory**, in which path names not beginning with a slash are looked for. For example, in Fig. 1-14, if */Faculty/Prof.Brown* were the working directory, use of the path *Courses/CS101*

Mount system call allows the file system on the CD-ROM to be attached to the root file system wherever the program wants it to be.

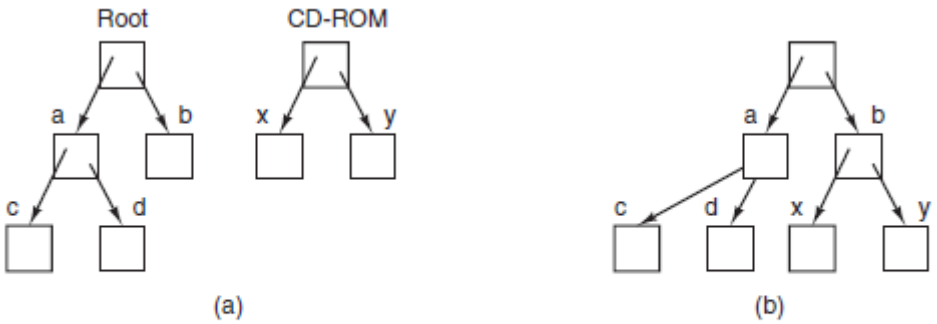


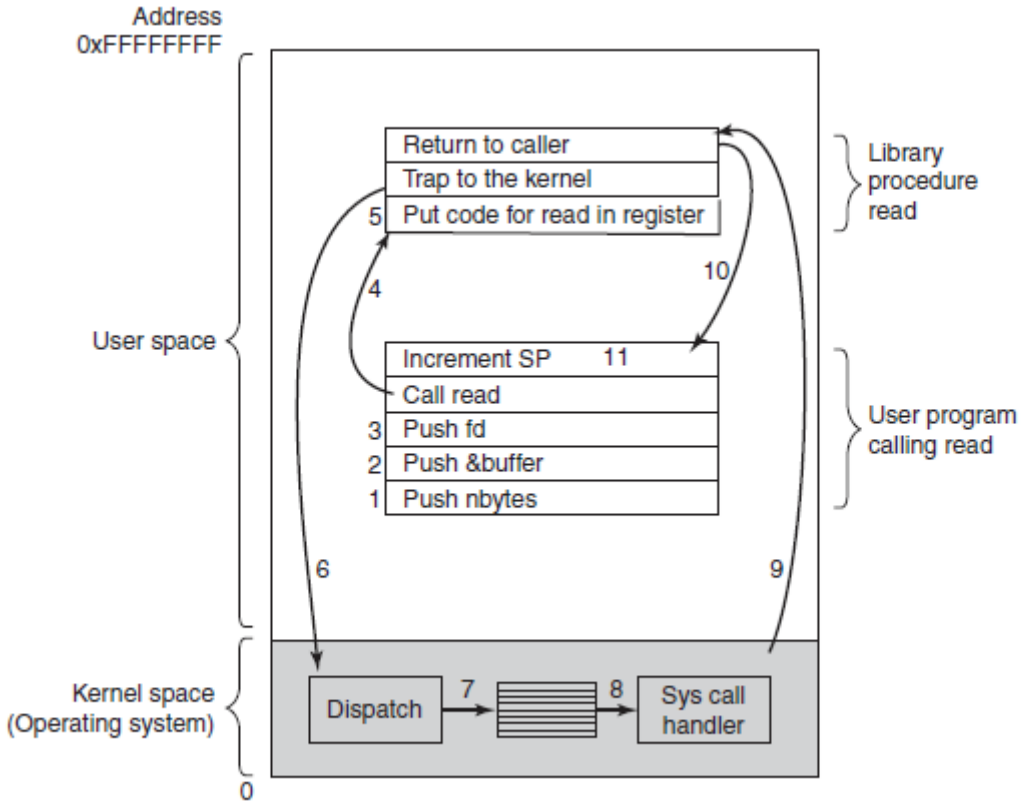
Figure 1-15. (a) Before mounting, the files on the CD-ROM are not accessible. (b) After mounting, they are part of the file hierarchy.

Block special files are used to model devices that consist of a collection of randomly addressable blocks, such as disks. By opening a block special file and reading, say, block 4, a program can directly access the fourth block on the device, without regard to the structure of the file system contained on it. special files are used to model printers, modems, and other devices that accept or output a character stream. By convention, the special files are kept in the */dev* directory.

A **pipe** is a sort of pseudofile that can be used to connect two processes. When process *A* wants to send data to process *B*, it writes on the pipe as though it were an output file. Process *B* can read the data by reading from the pipe as though it were an input file. Thus, communication between processes in UNIX looks very much like ordinary file reads and writes

Files in UNIX are protected by assigning each one a 9-bit binary protection code. The protection code consists of three 3-bit fields, one for the owner, one for other members of the owner's group (users are divided into groups by the system administrator), and one for everyone else. Each field has a bit for read access, a bit for write access, and a bit for execute access. These 3 bits are known as the **rwx bits**.

When any user logs in, a shell is started up. The shell has the terminal as standard input and standard output. GUI is just a program running on top of the operating system, like a shell


Address
0xFFFFFFFF
User space
Kernel space
(Operating system)
0
Return to caller
Trap to the kernel
5
Put code for read in register
4
10
Increment SP
11
Call read
3
Push fd
2
Push &buffer
1
Push nbytes
6
9
Dispatch
7
8
Sys call
handler
Library
procedure
read
User program
calling read