Introduction to UNIX

What is an Operating System?

• OS as an extended machine

– hides differences in hardware and provides an *extended machine* or a *virtual machine* to users
– a program uses functions of the kernel through *system calls*

• OS as a resource manager

– allocates resources in time (e.g., CPU, printer, etc.)
– allocates resources in space (e.g., memory, disk space, etc.)

• OS in the broad sense
– applications are also included
• OS in the narrow sense
– only the basic part, called the *kernel*

Programs and Processes

- *Program* is a sequence of instructions
  - It is saved in the disk as a command file
  - e.g., “/bin/ls”

- *Process* is a program that is currently executed
  - A program is loaded in memory and is being executed.
  - A process has its *context*.
  - There are more than one processes if the same program is executed simultaneously.

– Resources are allocated to each process
• e.g., CPU, memory, I/O devices, etc.

Execution Modes

• The execution of user processes is divided into two levels:
user and *kernel*

– Processes in user mode can access their own instructions and data but not kernel instructions and data

– Processes in kernel mode can access kernel and user addresses

– Some machine instructions are privileged and result in an error when executed in user mod

Brief History of UNIX

• In 1965, the project to develop MULTICS began.
– MULTiplexed Information and Computing Service
– GE, MIT, and AT&T Bell Lab.
– goal: to provide simultaneous computer access to a large community of users similar to the electricity distribution System

• In 1969, primitive version of MULTICS was running on GE 645.
– but it didn't provide general service computing.
– Bell Lab. withdrew from the project; GE quit computer business.

• MULTICS introduced many seminal ideas into the computer literature.

Brief History of UNIX

• Ken Thompson and Dennis Ritchie of Bell Lab. implemented a new system on PDP-7, and it is named UNIX.

• UNIX was moved to PDP-11 in 1971 and rewritten in C in 1973.

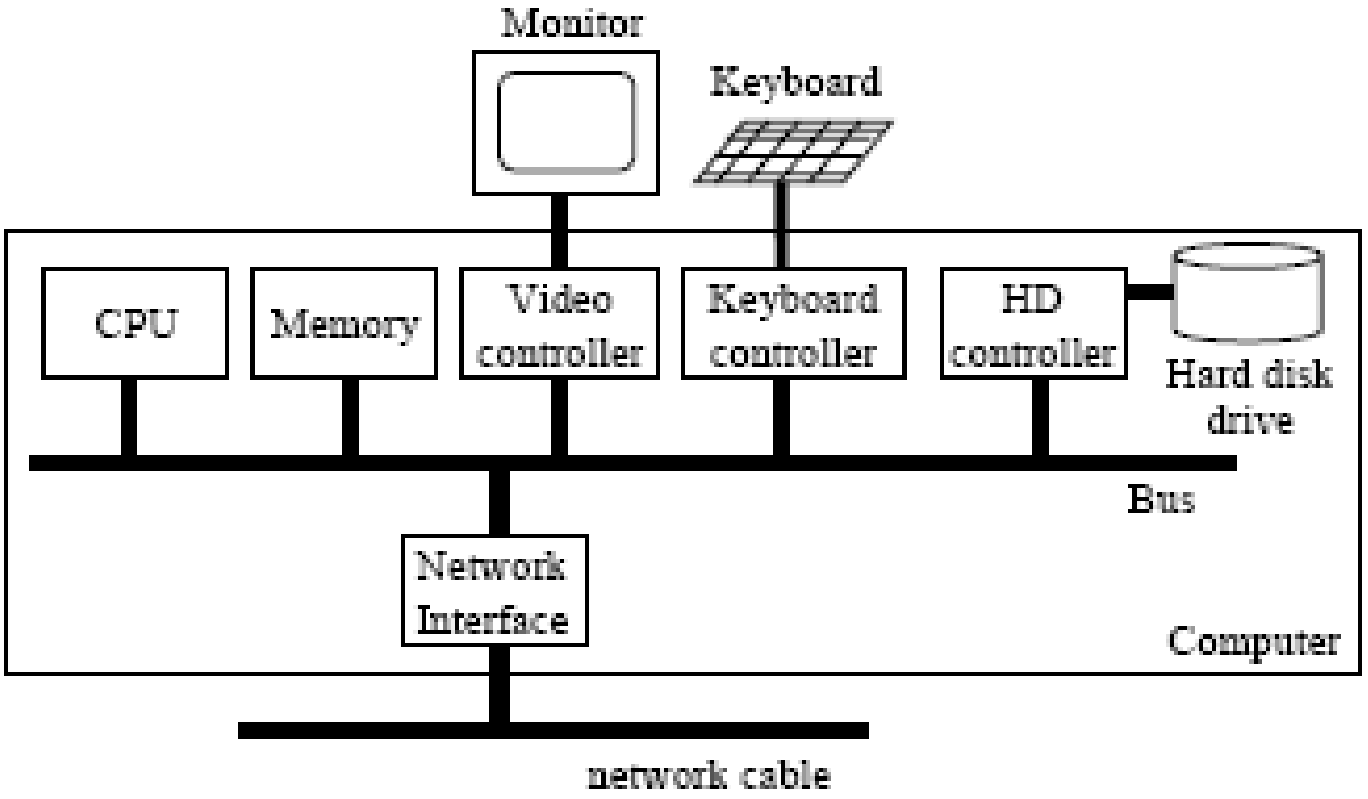
• Through development System III, in 1983, AT&T announced official support for System V.

• On the other hand, UCB developed 4.1BSD in 1980s.

Reasons behind success of UNIX

- The system is written in a high-level language, making it easy to read, understand, change, and move to other machines. Ritchie estimates that the first system in C was 20 to 40 percent larger and slower because it was not written in assembly language, but the advantages of using a higher-level language far outweigh the disadvantages (see page 1965 of [Ritchie 78b]).
- It has a simple user interface that has the power to provide the services that users want.
- It provides primitives that permit complex programs to be built from simpler programs.
- It uses a hierarchical file system that allows easy maintenance and efficient implementation.
- It uses a consistent format for files, the byte stream, making application programs easier to write.
- It provides a simple, consistent interface to peripheral devices.
- It is a multi-user, multiprocess system; each user can execute several processes simultaneously.
- It hides the machine architecture from the user, making it easier to write programs that run on different hardware implementations.

Overview of hardware



CPU (Processor)

• General purpose registers and special registers

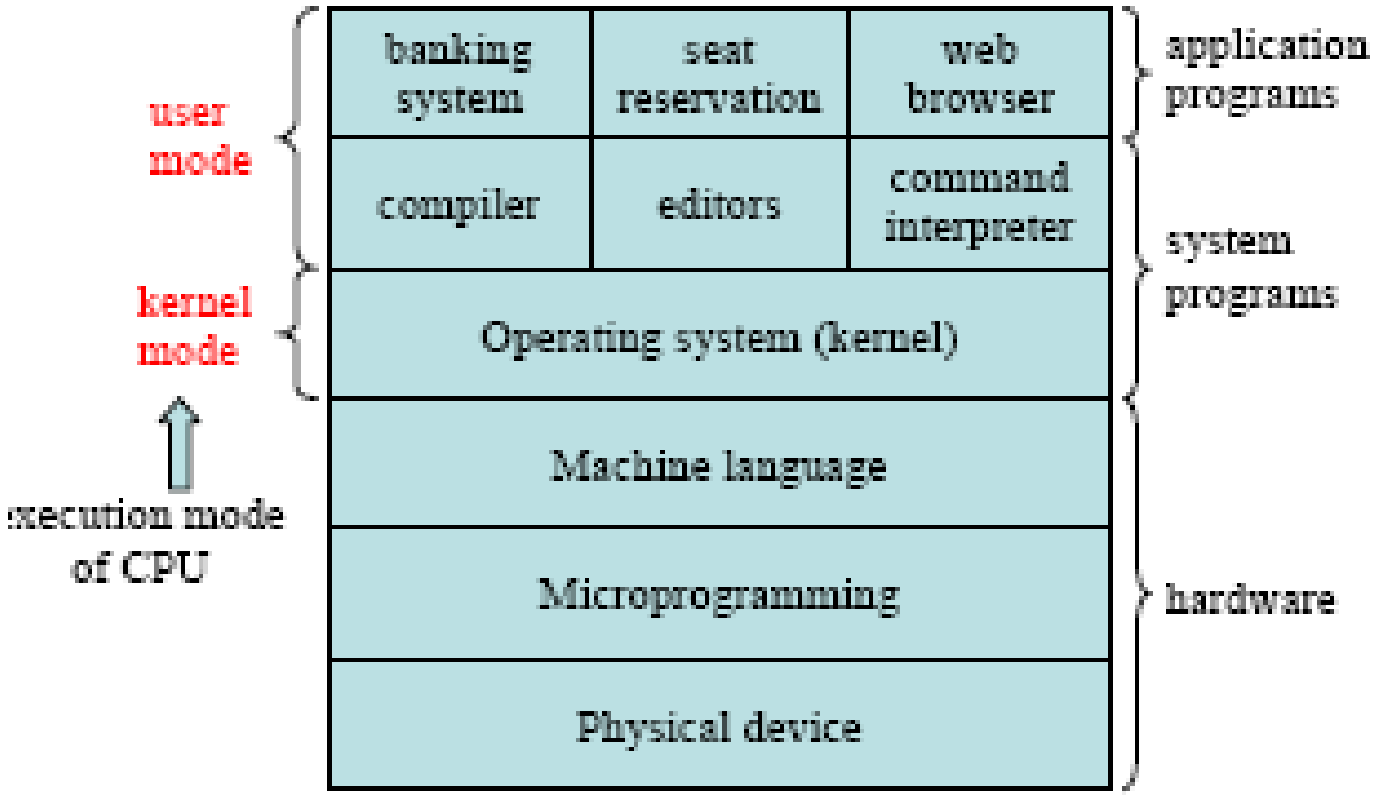
– general purpose registers are used for several types of execution
– program counter specifies the address of the operation code to be executed
– stack pointer specifies the address of the top of the stack
– PWS (Processor Status Word) specifies the status of the CPU

• Execution modes of the CPU
  – *user mode* and *kernel mode*
  – in kernel mode, all instructions including *privileged instructions* can be executed.

– OS is running in kernel mode

– user program is running in user mode

Position of operating System



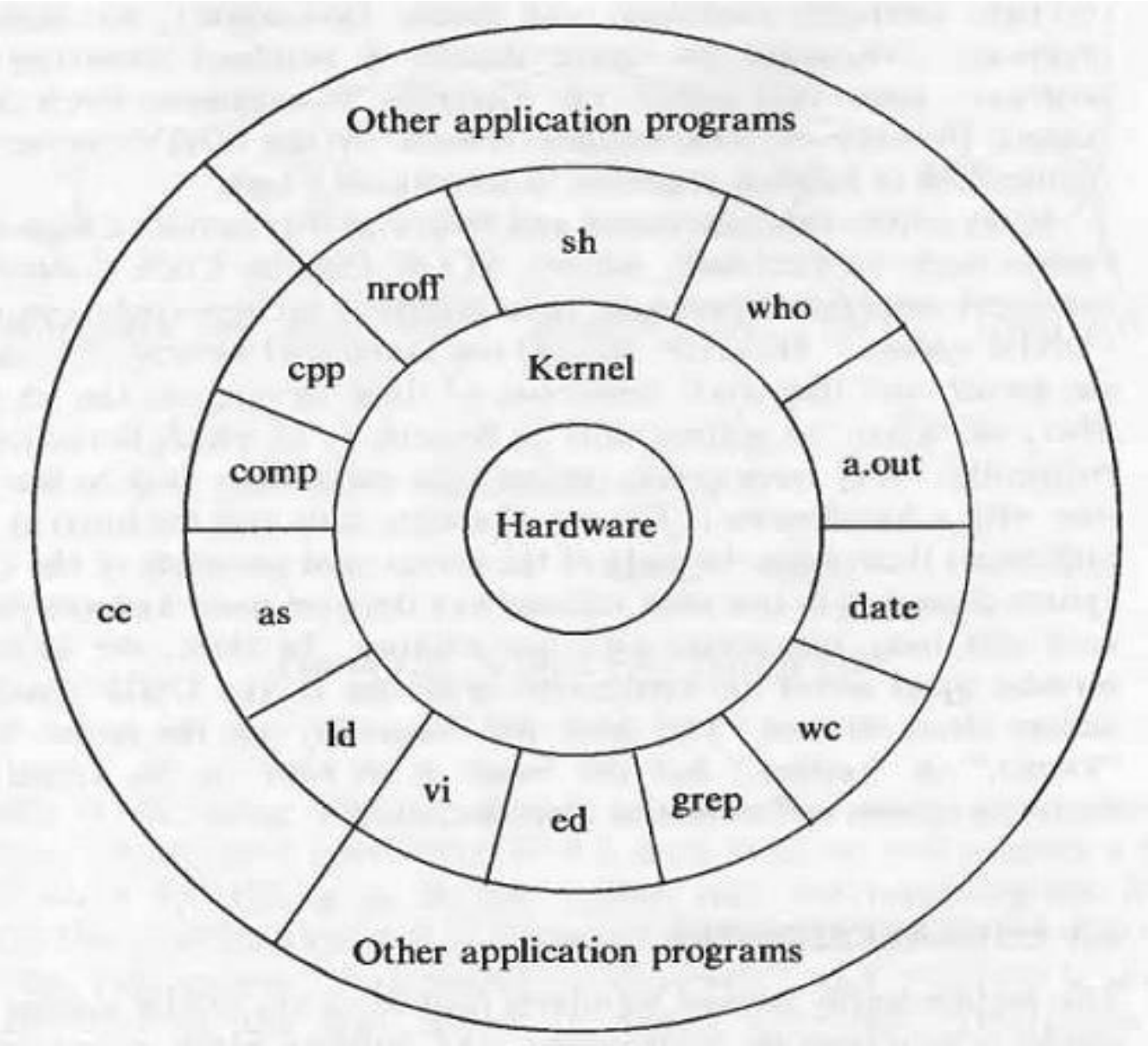
Architecture of UNIX

- 2 central concepts: *file* and *process*

– 2 major components of the kernel:
file subsystem and *process control subsystem*

- 3 levels: *user*, *kernel*, and *hardware*
  – user/kernel border: system call and library interface
  – system calls look like ordinary function calls.
  – libraries map these function calls to the primitives needed to enter the operating system.
  – The libraries are linked with the program at compile time and are thus part of the user program

Architecture of UNIX System



User Perspective of UNIX

High Level Features of UNIX:

- File subsystem
- Processing Environment
- Building Block Primitives

File Subsystem

• The file subsystem manages files, allocating file space, administering free space, controlling access to files, retrieving data for users.

• Processes interact with the file subsystem via system calls such as *open, close, read, write, stat*, and *chmod*.

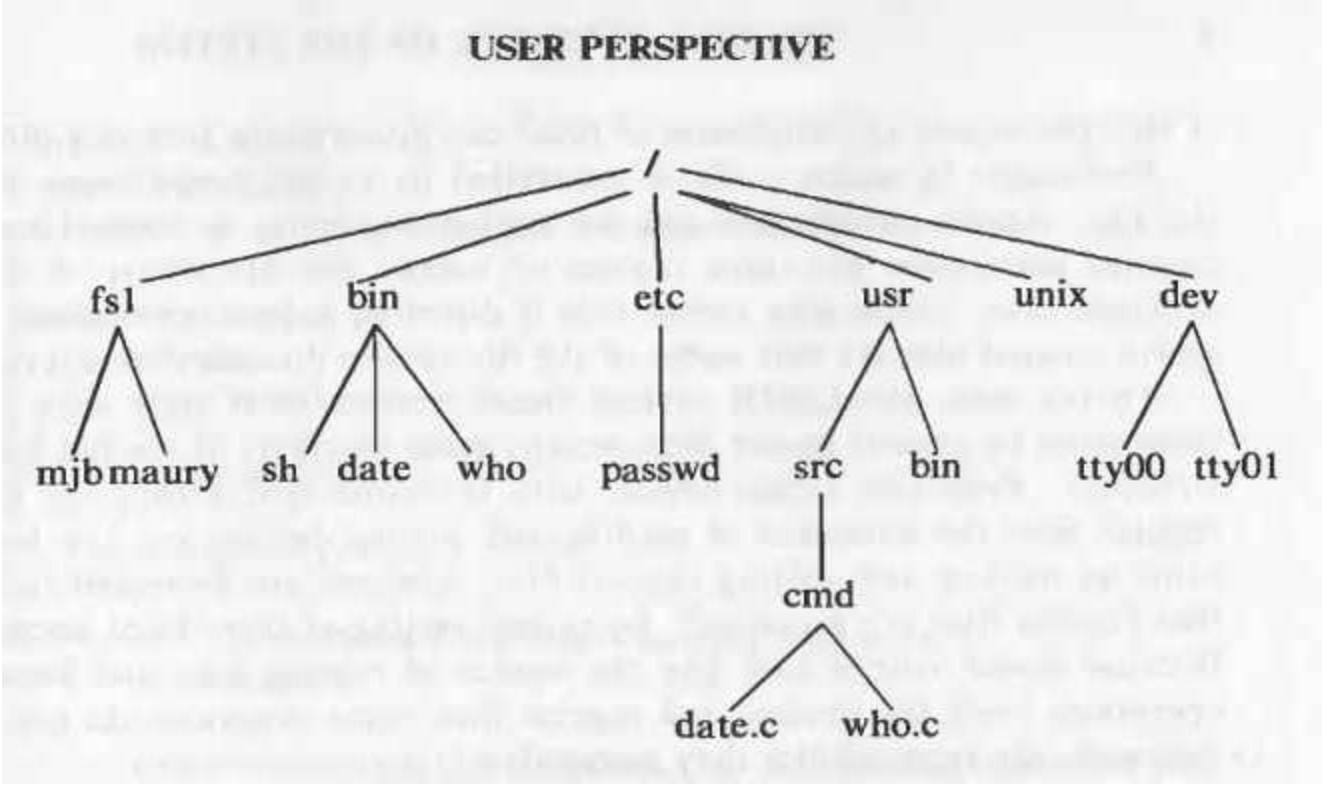
• The file subsystem access file data using a buffering mechanism that regulates data flow between the kernel and secondary storage.

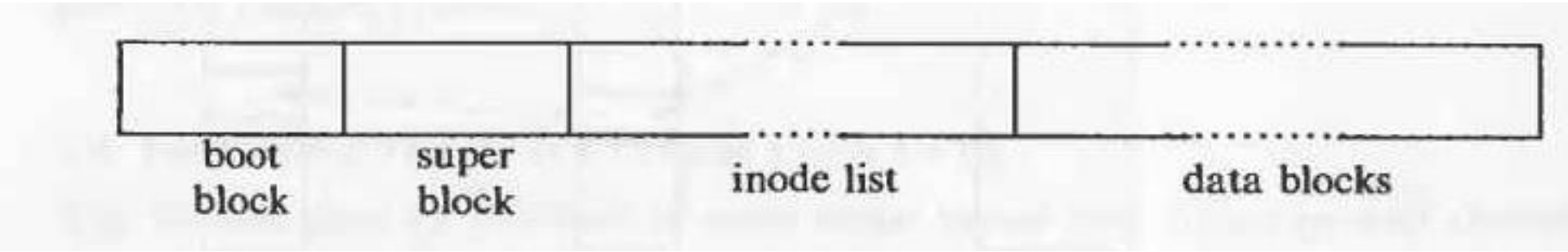
• The buffering mechanism interacts with block I/O devices drivers.

• The file subsystem also interacts with “raw” I/O device drivers without buffering mechanism.

The UNIX file system is characterized by

- a hierarchical structure,
- consistent treatment of file data,
- the ability to create and delete files,
- dynamic growth of files,
- the protection of file data,
- the treatment of peripheral devices (such as terminals and tape units) as files.





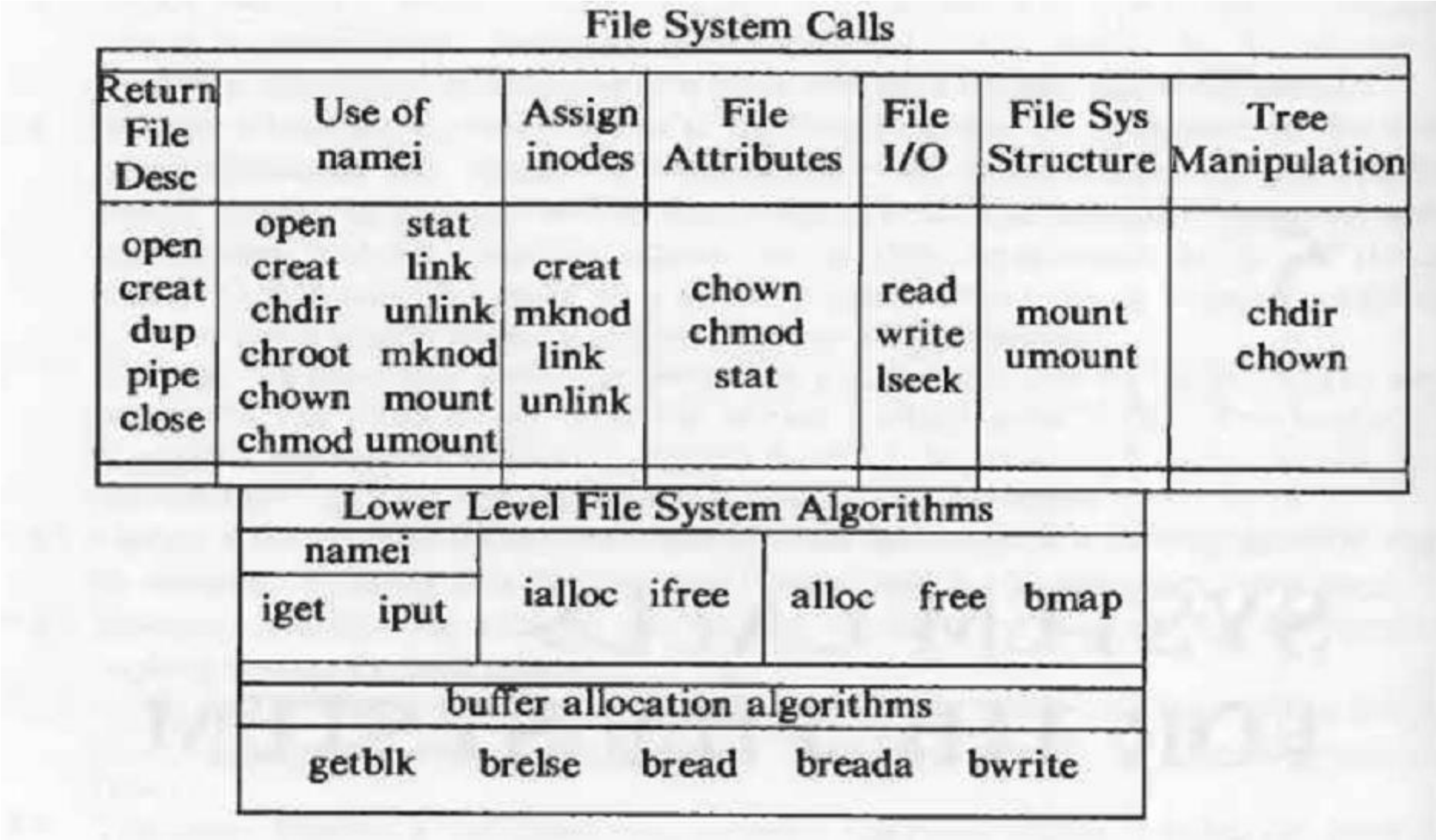
Storage structure on disk

- File identification :– path – relative and absolute
  example: /etc/password
  /bin/who
  / - root directory

- User is not aware of internal storage of files.

- File permissions

- Device Files.


File System Calls
Return File Desc
Use of namei
Assign inodes
File Attributes
File I/O
File Sys Structure
Tree Manipulation
open
creat
dup
pipe
close
open stat
creat link
chdir unlink
chroot mknod
chown mount
chmod umount
creat
mknod
link
unlink
chown
chmod
stat
read
write
lseek
mount
umount
chdir
chown
Lower Level File System Algorithms
namei
iget iput
ialloc ifree
alloc free bmap
buffer allocation algorithms
getblk brelse bread breada bwrite

An Example of System Calls and Library Functions

```
main( )
{
        int fd, c;
        char buf[512], buf2[512];
        if ((fd = open("test.txt", O_RDONLY)) < 0) {

                perror("open");
                exit(1);
        }
        if ((c = read(fd, buf, 512)) < 0) {
                perror("read");
                exit(1);
        }
        close(fd);
        bcopy(buf, buf2, 512);
}
```

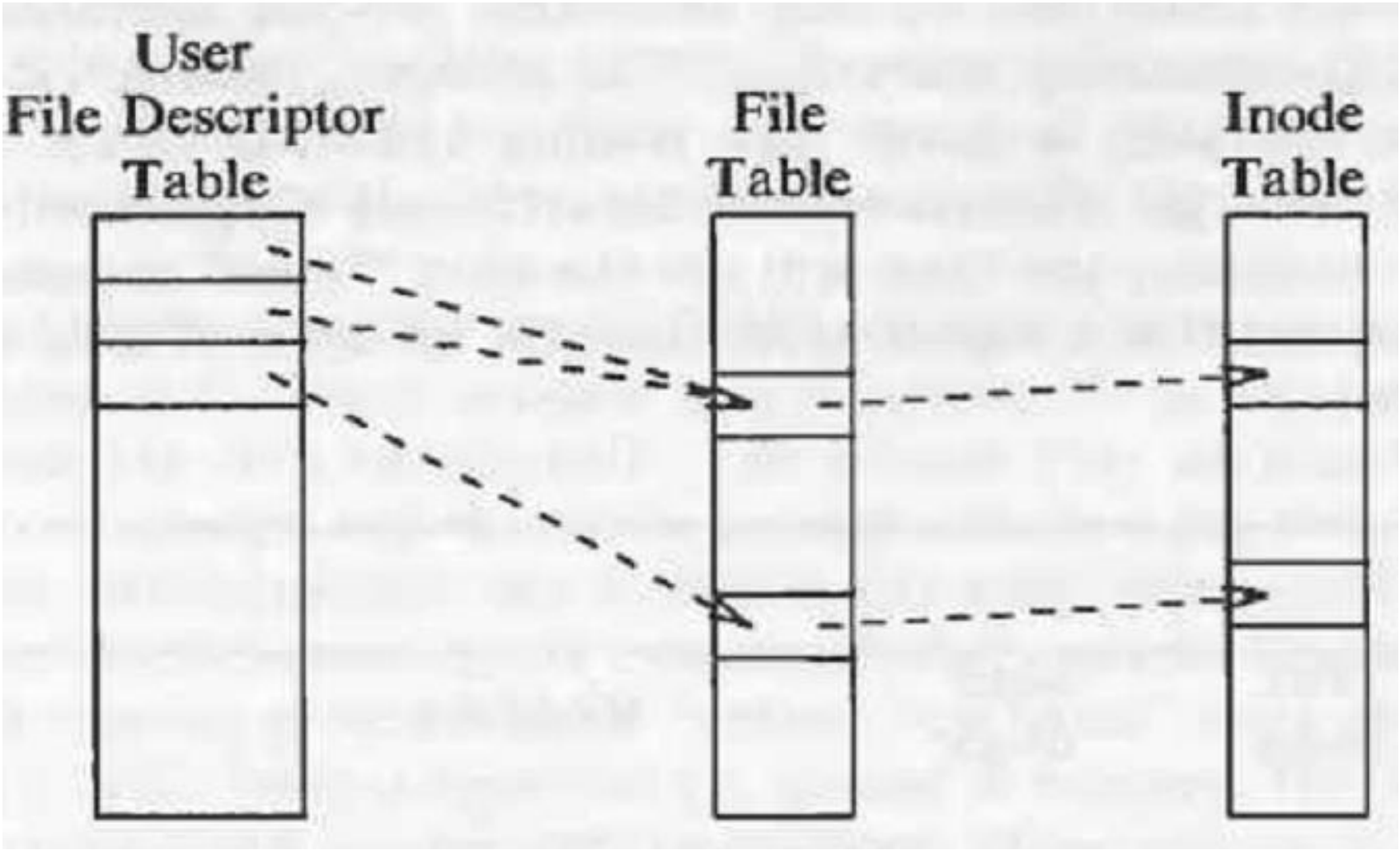
- **open**
- opens a file (preparation for read/write)
-returns a *fd*
-(file descriptor)

- **read**
- reads data from the file specified by *fd* to the buffer

- **close**
-closes the file

- these system calls are processed in the kernel

- **bcopy** is a library function


User
File Descriptor
Table
File
Table
Inode
Table

Process Environment:

A Program is an executable file.

A Process is an instance of the program in execution

Many processes can execute simultaneously in UNIX. This is due to Multiprogramming or multitasking

Various system calls allows to create, terminate , synchronize and respond to Events in operating system.

fork, exec, exit, wait etc

```
main(argc, argv)
        int argc;
        char *argv[];
{
        /* assume 2 args:  source file and target file */
        if (fork() == 0)
                execl("copy", "copy", argv[1], argv[2], 0);
        wait((int *) 0);
        printf("copy done\n");
}
```

Building Block Primitives:

The philosophy of UNIX is to provide operating system primitives that enables user to build small programs upon which more complex programs can be build.

One of the primitive is I/O Redirection

```
ls > dout.txt
a.out < in.txt
nroff –mm < doc1 > doc1.txt  2>error
```

Standard in, standard out, standard error

Pipes

```
ls | grep httpd
```

Operating System Services:

Controlling execution of processes by allowing their creation, termination, suspension, and communication

Scheduling processes fairly for execution on the CPU

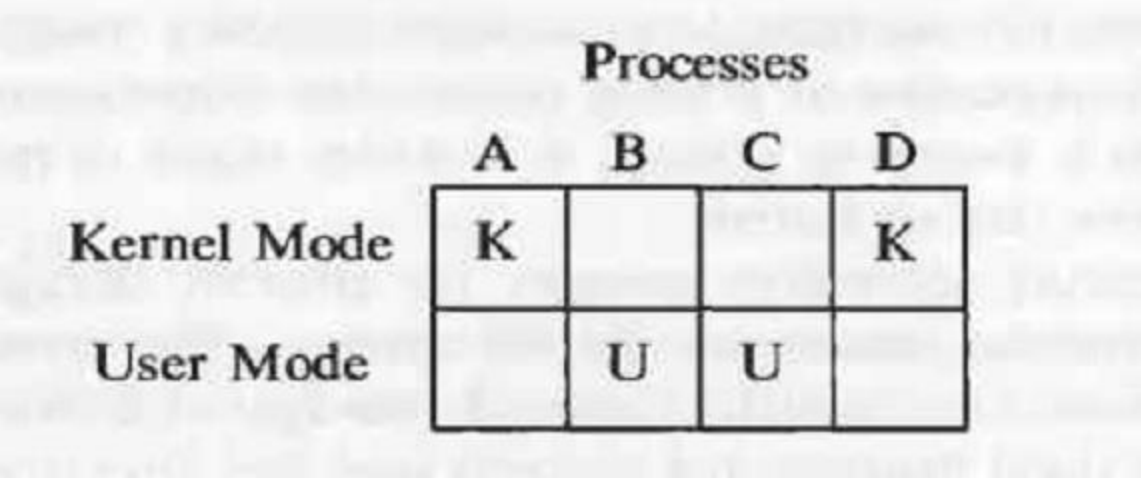
Allocating Main memory for executing processes, swapping

Allocating secondary storage for efficient storage of user data

Allowing processes controlled access to devices such as disk, Terminals, network, tape drives

Assumption about hardware:

Execution mode – User mode and kernel mode



	Processes			
	A	B	C	D
Kernel Mode	K			K
User Mode		U	U	

Interrupts and Exceptions

UNIX allows devices to interrupt the CPU asynchronously.

On interrupt :

OS saves the context of the currently executing Process(Frozen image of process).

Determines the Sources of interrupt

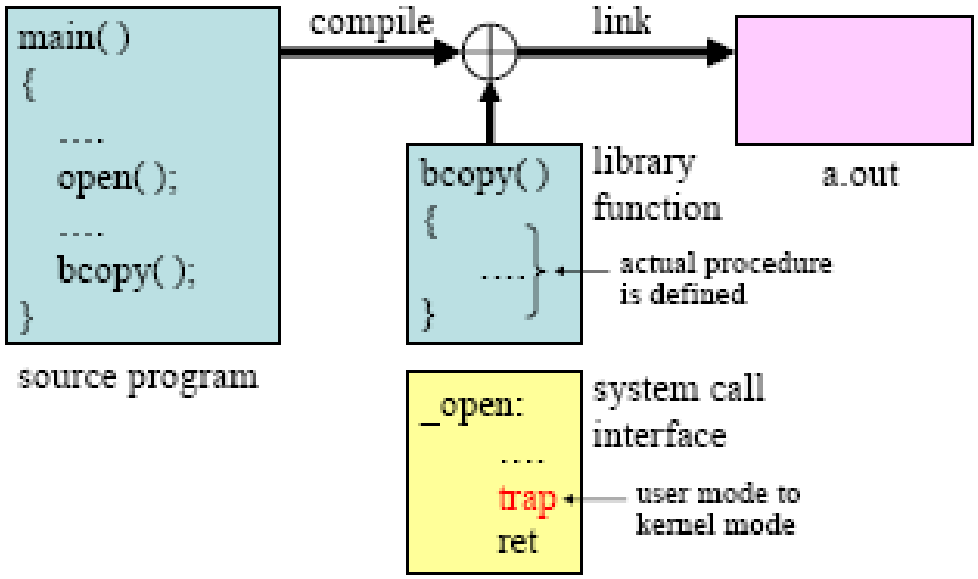
Executes the Interrupt Service Routine corresponding to the interrupt.

Restores the context of process.

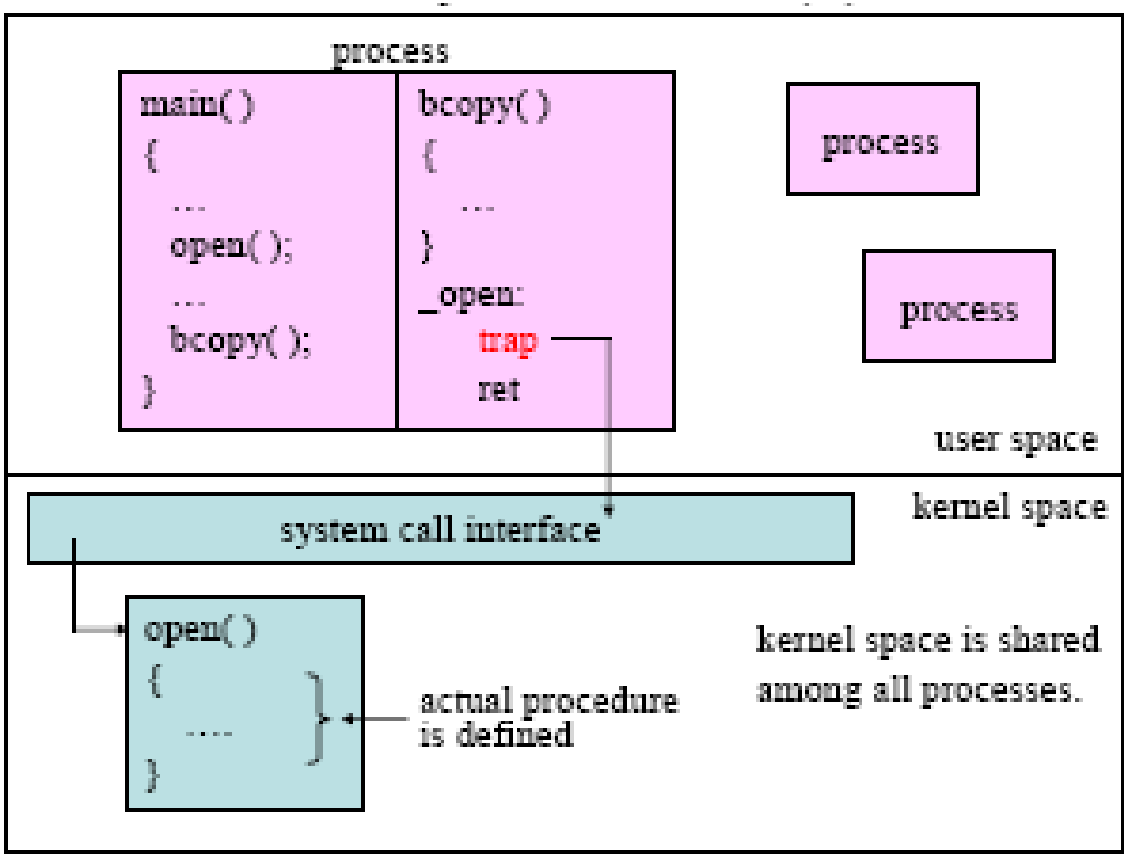
Process continues its execution as if nothing has happened.

Hardware usually priorities the interrupts as per their importance (importance of the device connected to it)

An Example of System Calls and Library Functions



An Example of System Calls and Library Functions



Process Control Subsystem

• The process control subsystem is responsible for process synchronization, interprocess communication, memory management, and process scheduling.

– system calls: *fork, exec, exit, wait, brk*, and *signal*

• The file subsystem and the process control subsystem interact when loading a file into memory for execution.

Memory Management Module

• The memory management module controls the allocation of memory.

• If at any time the system does not have enough physical memory for all processes, the kernel moves them between main memory and secondary memory.
– 2 policies: *swapping* and *demand paging*
– *swapper* process

Scheduler Module

• The scheduler module allocates the CPU to processes.

• It schedules them to run in turn until they voluntarily relinquish the CPU while awaiting a resource or until the kernel preempts them when their recent run time exceeds a time quantum.

• The scheduler then chooses the highest priority eligible process to run.

Other Modules

• Interprocess communication

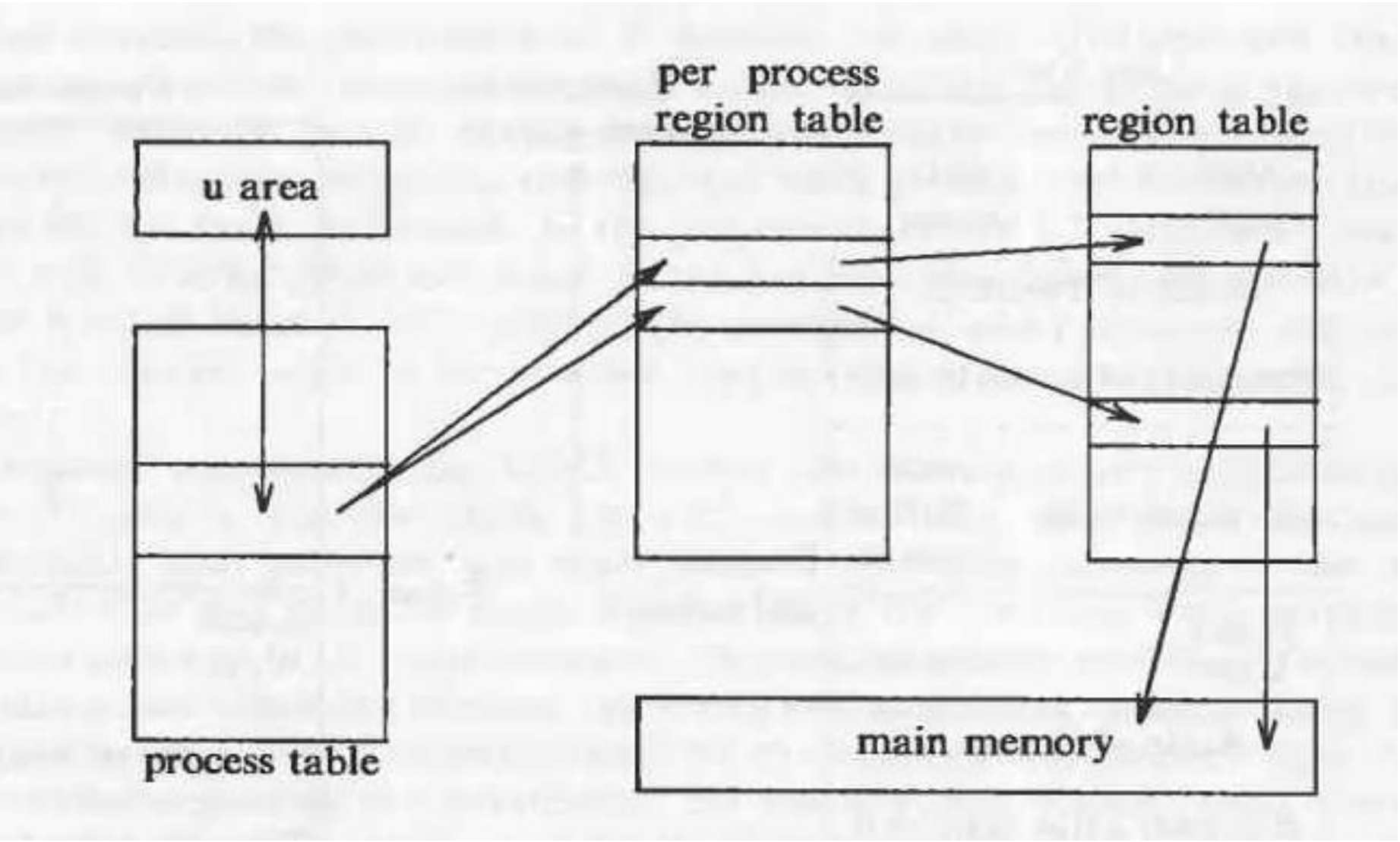
– ranging from asynchronous signaling to synchronous transmission of messages between processes.

• Hardware control:

– responsible for handling interrupts and for communicating with the machine.

– the kernel may resume execution of interrupted process after servicing an interrupt.

– Interrupts are not serviced by special processes but by special functions in the kernel, called in the context of the currently running process.


per process
region table
region table
u area
process table
main memory

Block Diagram of the System Kernel