# Unix System Programming

# Overview

# UNIX History

- Originally developed in 1969 at Bell Labs by Ken Thompson and Dennis Ritchie.
- 1974: Thompson, Joy, Haley and students at Berkeley develop the Berkeley
- Software Distribution (BSD) of UNIX
- 1978: UNIX Version 7 released
- 1980년 : XENIX (Microsoft)
- two main directions emerge: BSD and what was to become “System V”
  - 1984년 : 4.2 BSD (TCP/IP)
  - 1986년 : 4.3 BSD (NFS)
  - 1995년 : 4.4 BSD-Lite Release 2
  - 1989년 : SVR 4.0
  - 1991년 : SVR 4.0 MP
  - 1992년 : SVR 4.0 ES/MP, UNIXWARE 2.0

# UNIX History

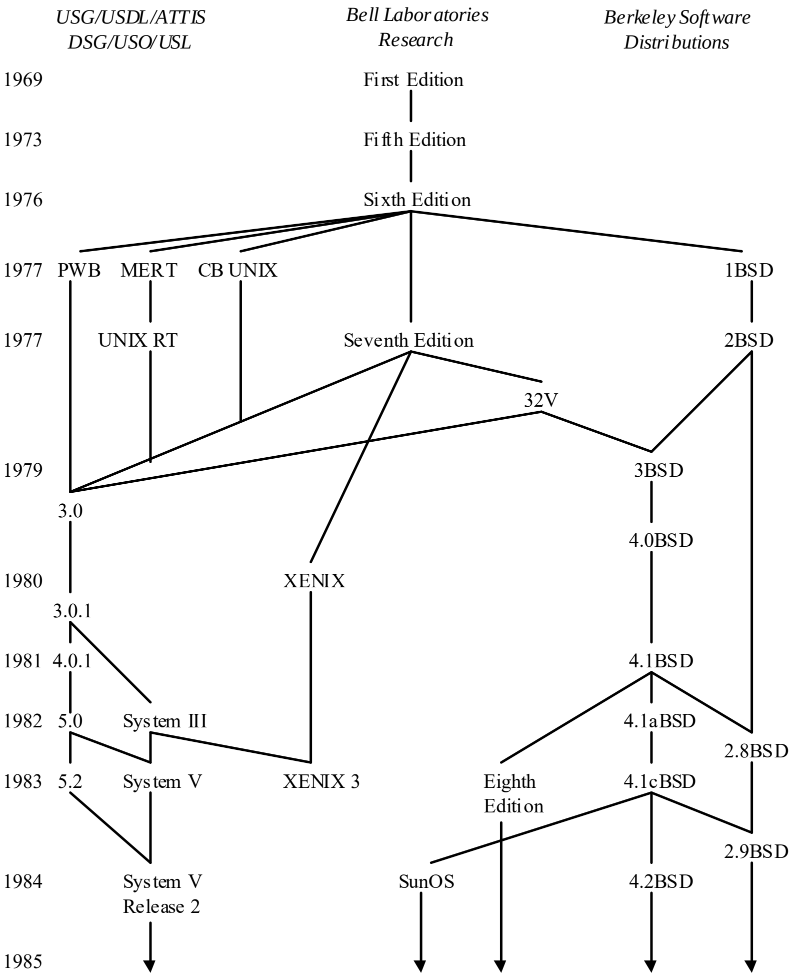


**Figure 1.1** The UNIX system family tree, 1969-1985

# UNIX History

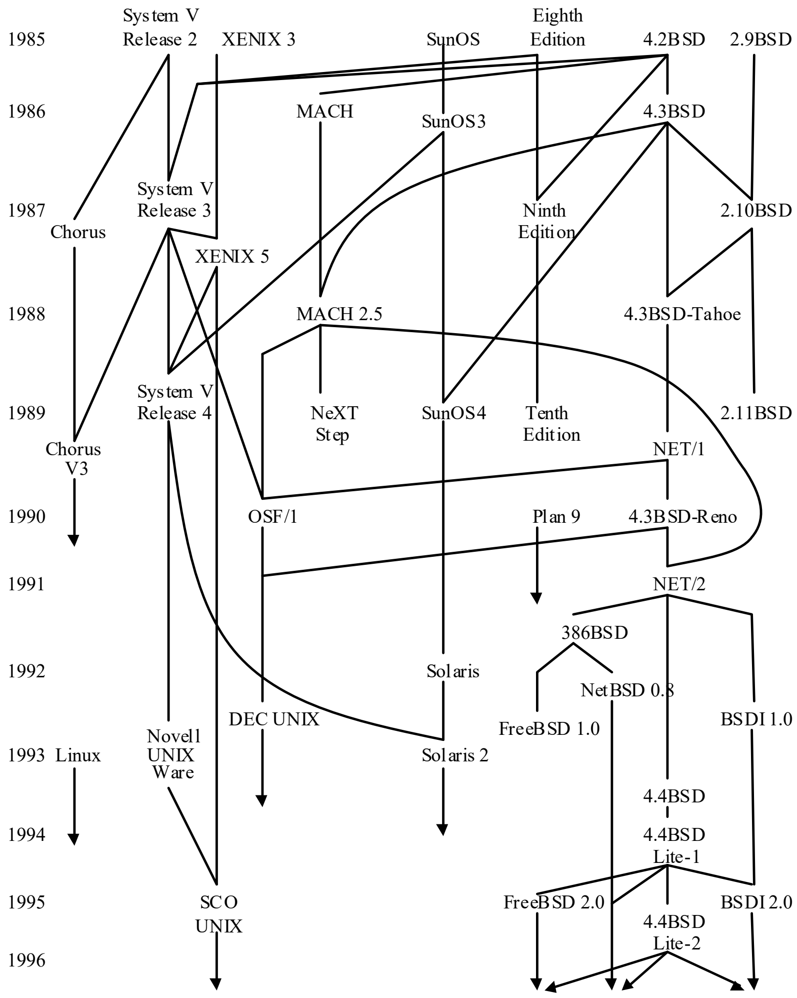


**Figure 1.2** The UNIX system family tree, 1986-1996

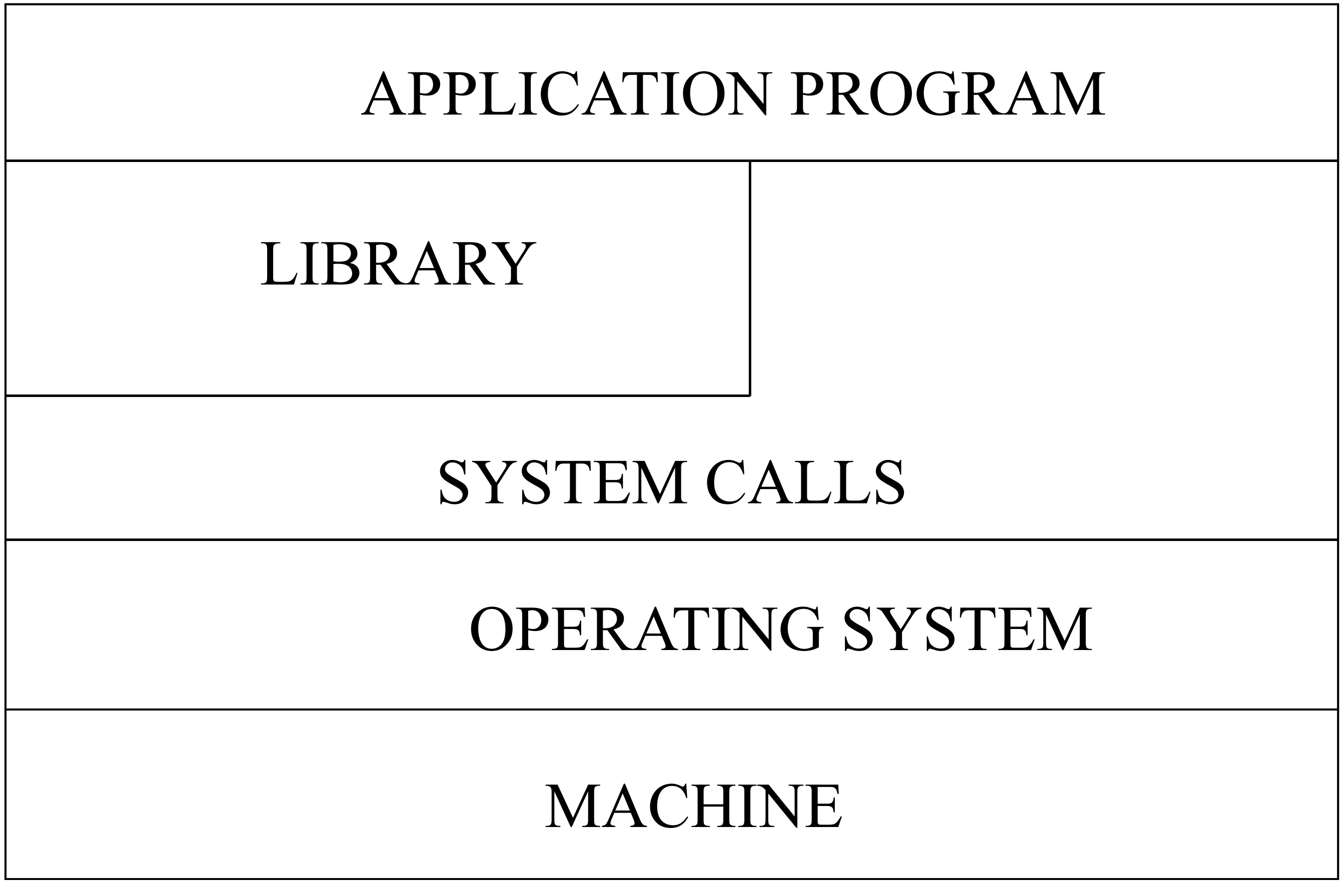
# Unix Standards

- Two distinct Unix flavors coexist: System V and BSD
- The IEEE develop a standard for the Unix libraries called POSIX (Portable Open System Interface)
- In 2002, Open Group, IEEE, ISO/IEC approved revised POSIX standard
- A POSIX-compliant implementation must support the POSIX base standard

# Supported Programming Standards

- POSIX(Portable Operating System Interface)
    - is not an OS, It is a set of OS interface standards
    - 1003.1
        - System interfaces (POSIX.1)
    - 1003.2
        - Shells and Utilities
    - 1003.1b
        - Real-time Extensions[3], [4]
        - RT-Signal, Signal Orders…
    - 1003.1c
        - User-level threads (pthreads, POSIX threads library functions)
    - 1003.1g
        - Networking Standards

# System Structure



APPLICATION PROGRAM

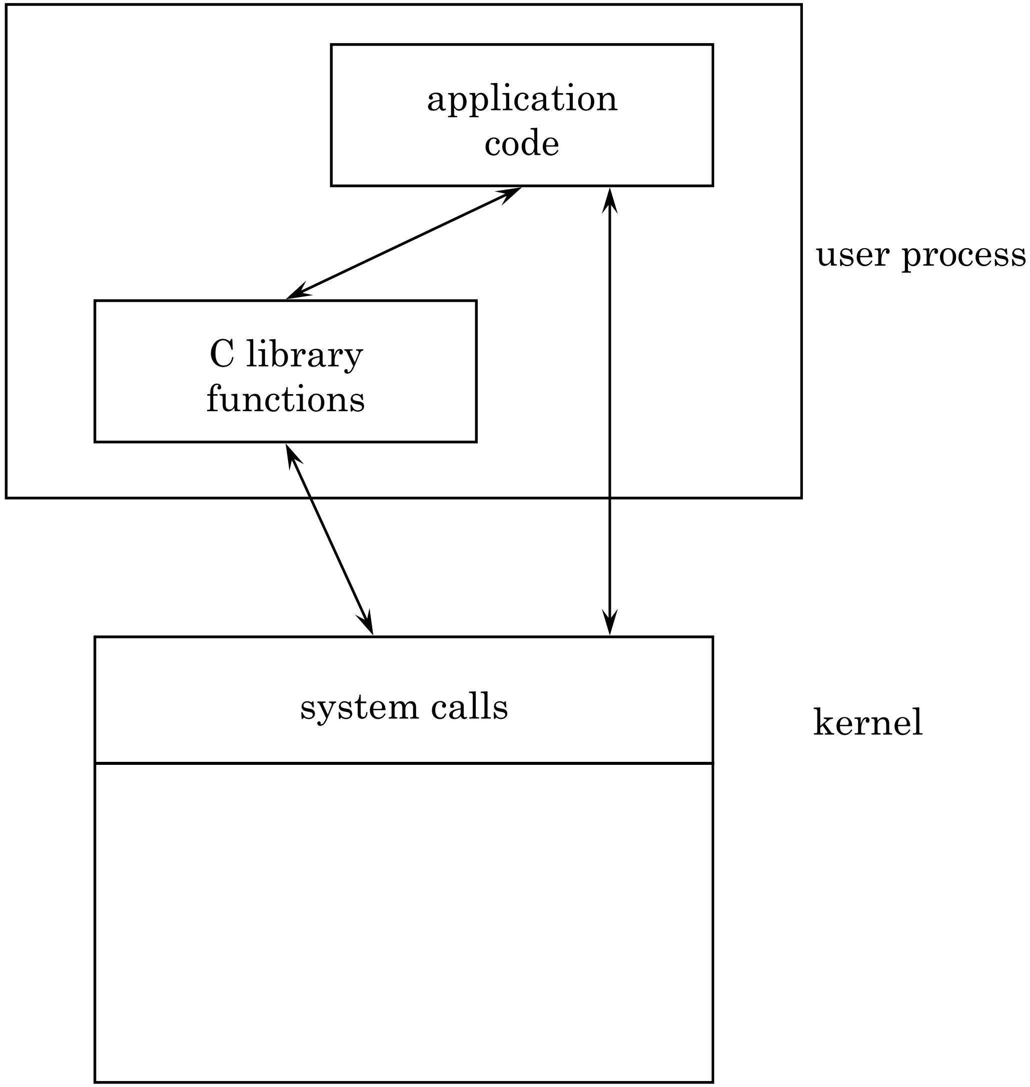
LIBRARY

SYSTEM CALLS

OPERATING SYSTEM

MACHINE

# Library & System call



# System Structure

- APPLICATION PROGRAMS

- LIBRARY
  - library의 일부는 그 속에 system call을 포함(ex. printf, puts)
- SYSTEM CALLS
  - UNIX operating system에 대한 요구
  - Kernel 내부의 instruction이 수행
  - system call이 불려지면 user mode에서 kernel mode로 변경
  - actual machine hardware( memory, disk etc.)는 system call로서만 접근 가능
  - application process와 operating system과의 interface
- OPERATING SYSTEM
  - system resource를 효율적으로 사용하도록 관리
  - process management
  - memory management
  - file management
  - I/O management

# MAN

- Section 번호 별 구성

1 User commands

2 System calls

3 Libraries Functions

4 Special Files

5 File Formats

6 Games

7 Miscellaneous

8 Administration and Privileged Commands

9 Kernel References Guide

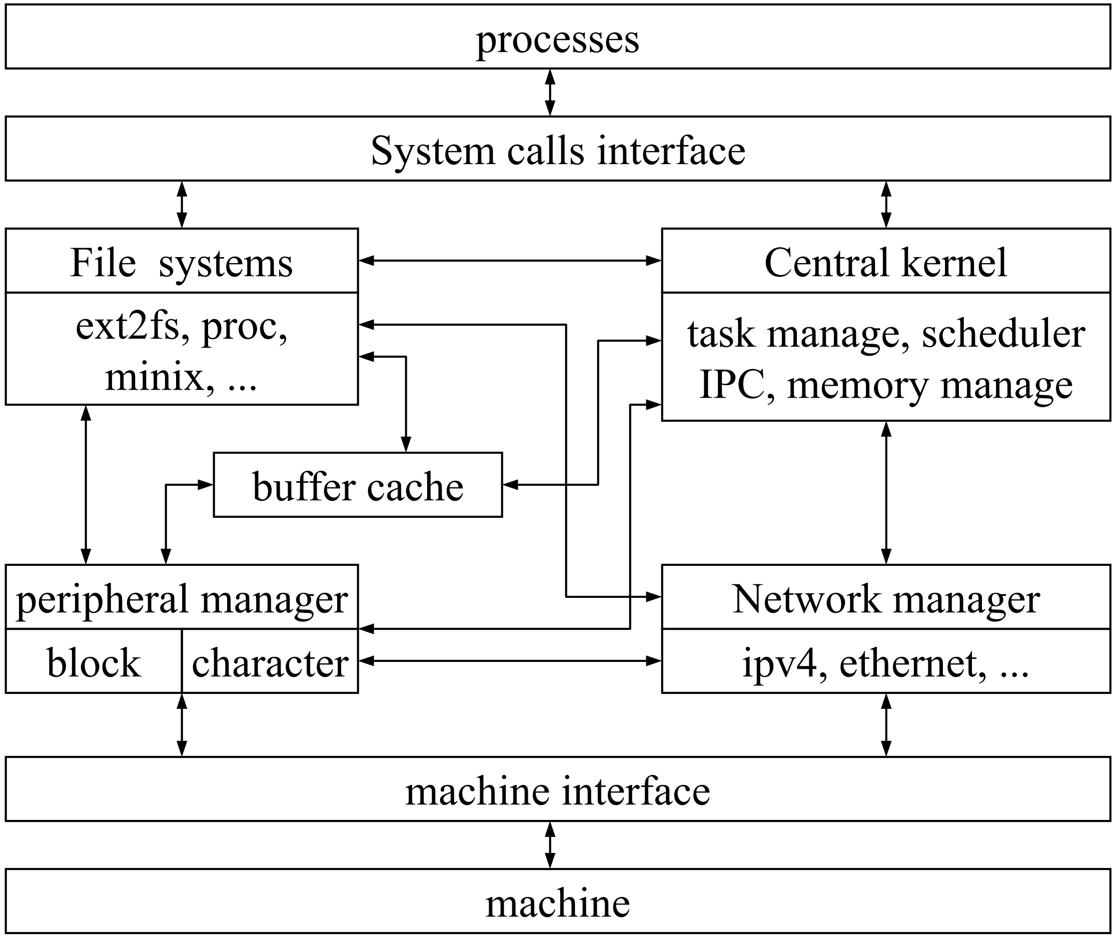
# MAN

- 1 User Commands
  - ex) ls, ps, cat, cp
- 2 System Calls
  - ex) open, read, write, fork
- 3 Library Functions
  - 3C C Standard library, libc에 포함
  - ex) strcpy, strcat, printf, gets
  - 3M Mathematical library, compile시 -lm으로 library를 연결해야한다.
  - ex) sin, cos
  - 3F FORTRAN library
  - 3X various special library
- man page 사용
  - man [section] name
  - $ man –s 2 write - system call write
  - $ man –s 3C printf - library printf

# MAN

- NAME
  - command name
- SECTION
  - manual의 section 번호
- NAME
  - command가 하는 일을 간략히 설명
- SYNOPSIS
  - coding하는 형식
- DESCRIPTION
  - 무엇을 하는 가에 대한 자세한 설명
- RETURN VALUE
  - return code가 무엇을 의미하는가에 대한 설명
- ERRORS
  - 각 error code에 대한 설명
- EXAMPLE
- CONFORMING TO
- SEE ALSO
  - 관련 있는 system call이나 library들

# System Structure



# Kernel mode & User mode

- kernel mode
  - privileged mode
  - no restriction is imposed on the kernel of the system
  - may use all the instructions of the processor
  - manipulate the whole of the memory
  - talk directly to the peripheral controllers

# Kernel mode & User mode

- user mode
  - normal execution mode for a process
  - has no privileges
    - certain instructions are forbidden
    - only allowed to zones allocated to it
    - cannot interact with the physical machine
  - process carries out operations in its own environment, without interfering with other processes
  - process may be interrupted at any moment

# System Calls

- system call
  - system call is a request transmitted from the process to the kernel
    - process in user mode cannot directly access the machine resources
  - the kernel deals with the request in kernel mode, without any restrictions, and sends the result to the process
- trap instruction
  - causes the CPU to switch into kernel mode
    - Intel CPU: int 0x80

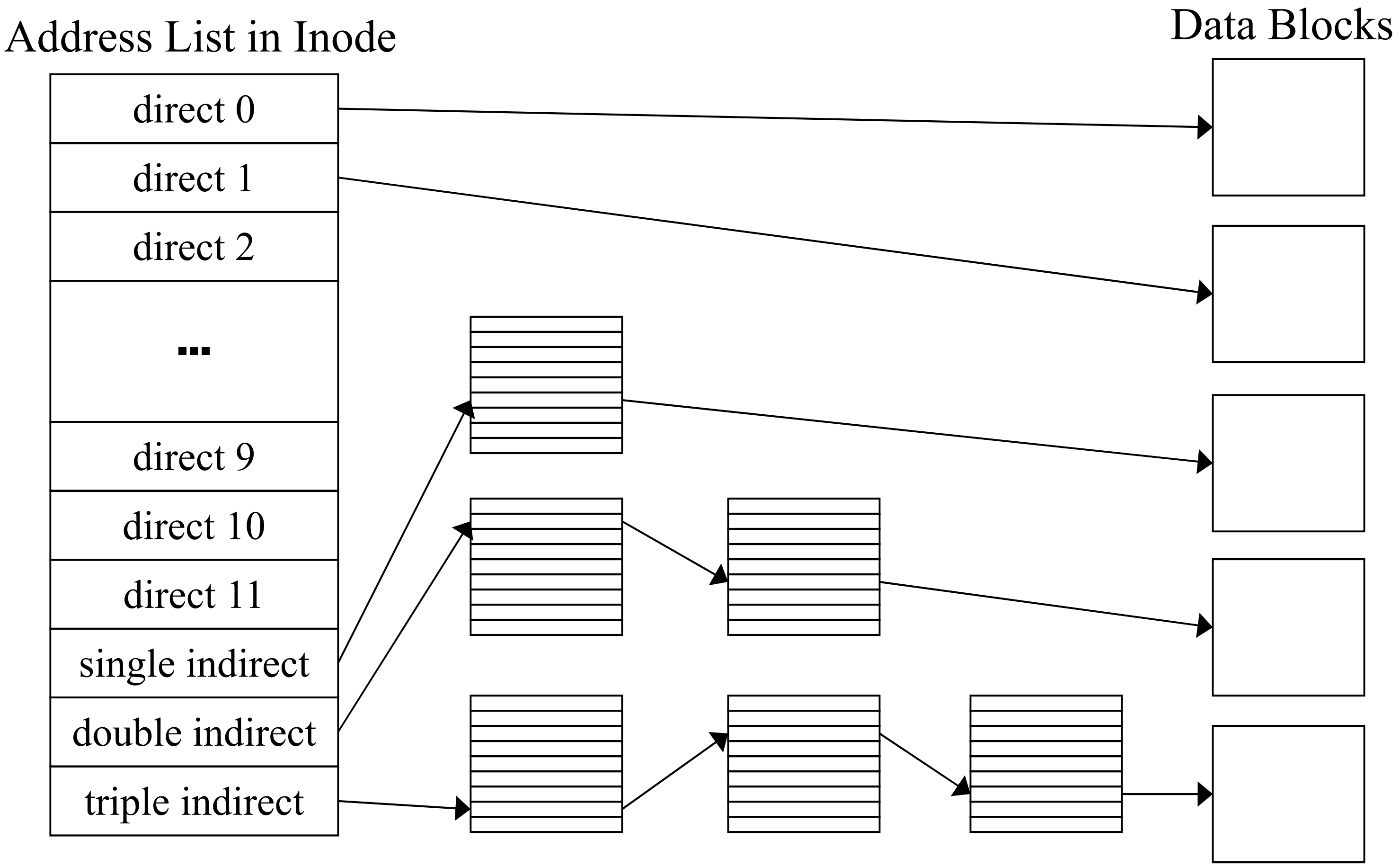
# File System

- inode table
  - contains a part of a table of inodes of the file system
- data blocks
  - used to store data contained in files and directories
  - also used to store indirect blocks

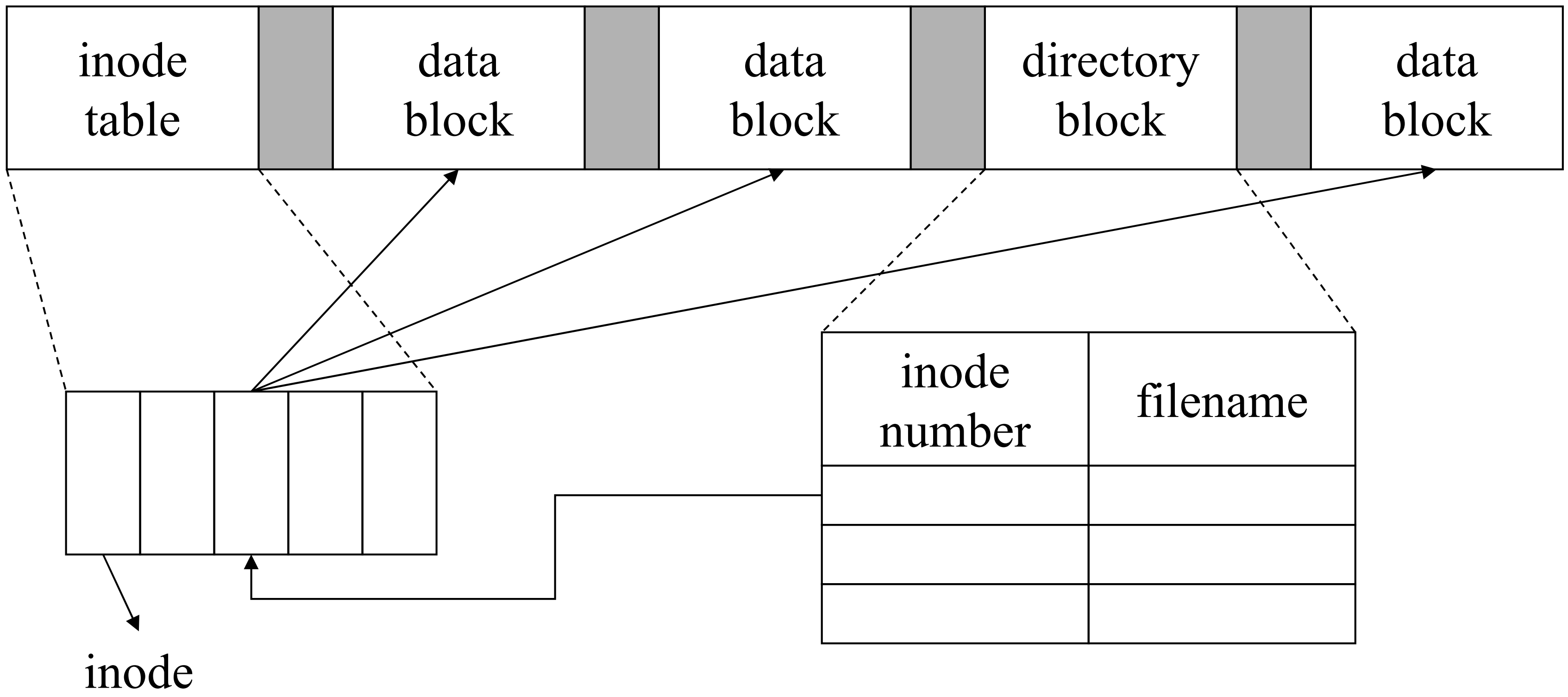
# Inode

- inode
  - internal representation of a file
  - every file has one inode
  - inode contains
    - type/permission
    - user(UID), group(GID)
    - file size, number of blocks, link counter
    - access time, modification time, change time
    - list of addresses of data blocks

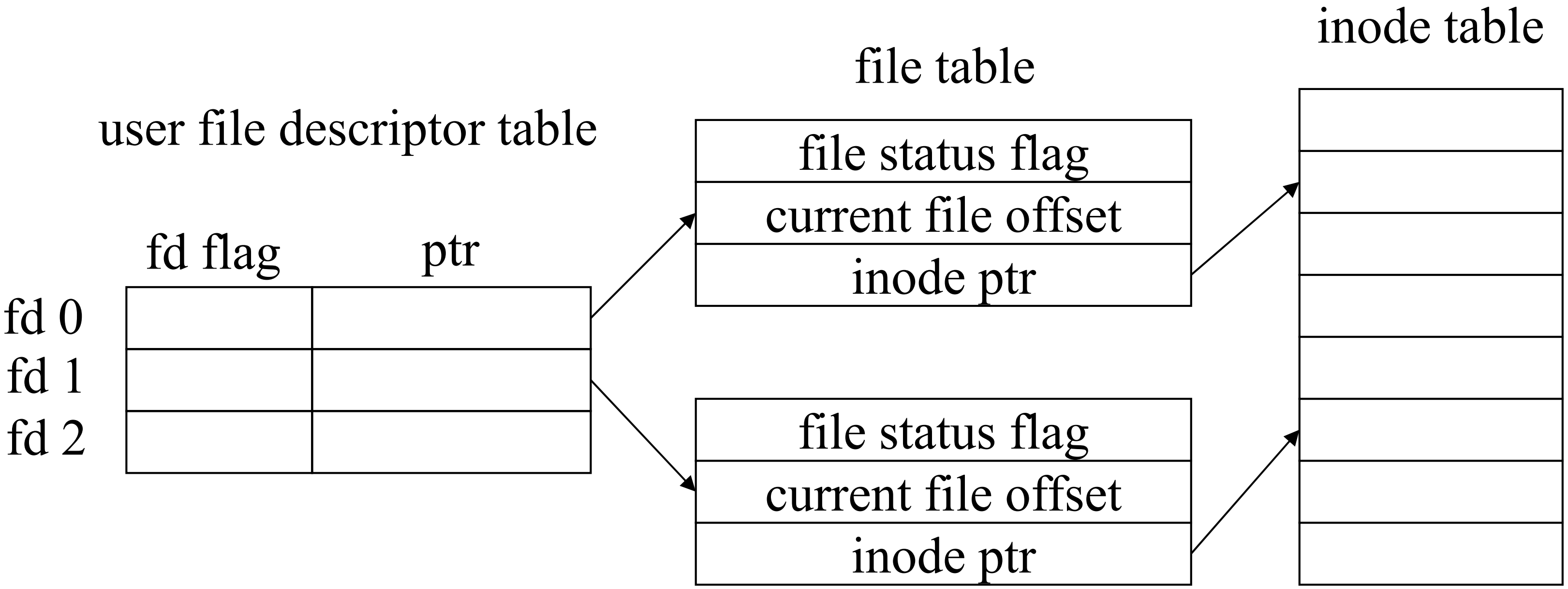
# Inode and Data Blocks



# File and Directory Blocks



# Kernel Data Structure for Open Files



# Kernel Data Structure for Open Files

- user file descriptor table
  - allocated per process
  - identifies all open files for a process
  - when a process “open” or “creat” a file, the kernel allocates an entry
  - return value of “open” and “creat” is the index into the user file descriptor table
  - contains pointer to file table entry

# Kernel Data Structure for Open Files

- file table
  - global kernel structure
  - contains the description of all open files in the system
    - file status flag (open mode)
    - current file offset
  - contains pointer to in-core inode table entry

# Kernel Data Structure for Open Files

- in-core inode table
  - global kernel structure
  - when a process opens a file, the kernel converts the filename into an identity pair(device number, inode number)
  - the kernel then loads the corresponding inode into in-core inode table

# Process

- program
  - an executable file residing in a disk
- process
  - an instance of a program in execution
    - at any given time a single instruction is carried out within the process
    - processes are often called “tasks” in Linux source code.
- process descriptor
  - task_struct contains all the information related to a single process

# Process

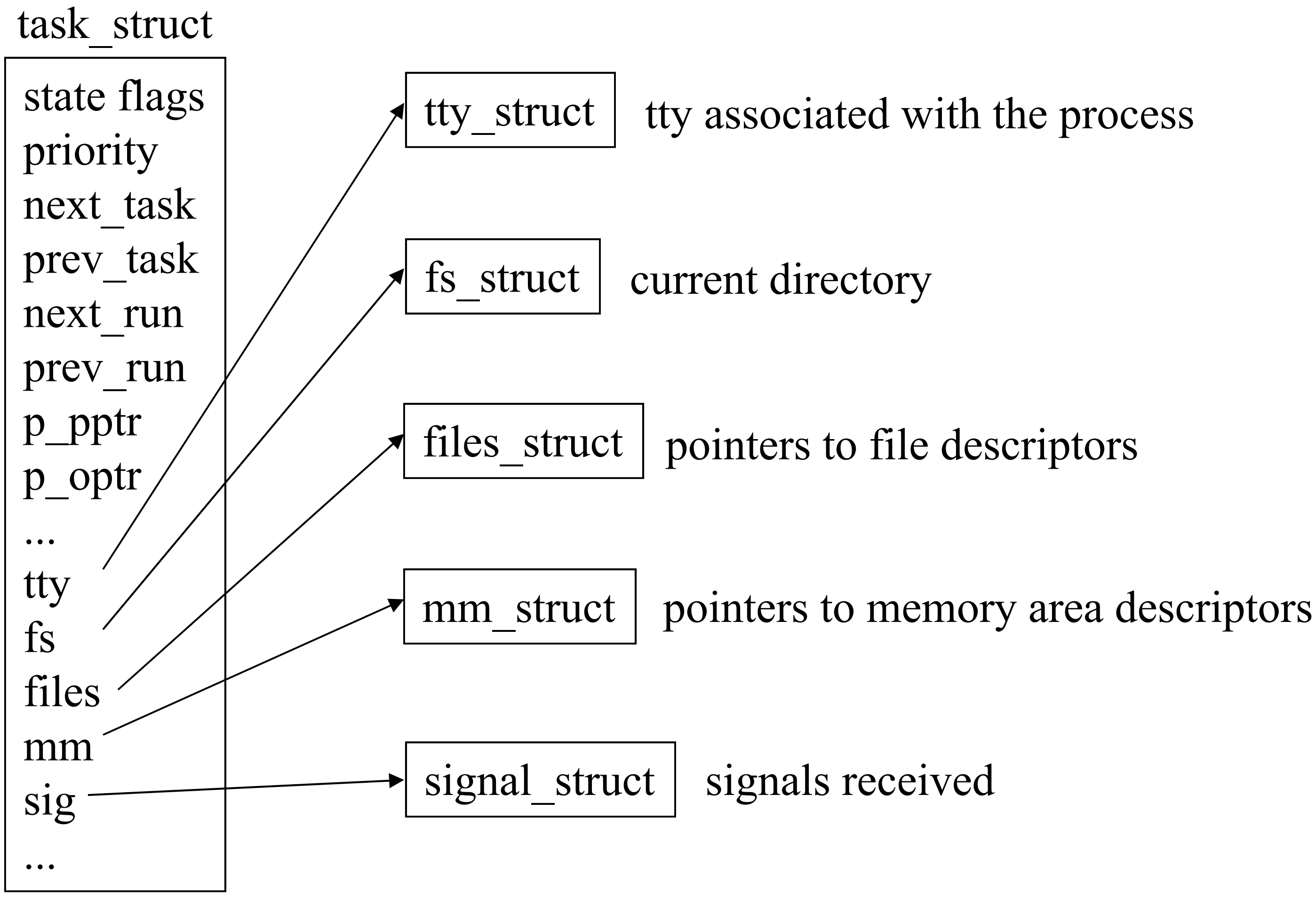
```
#include <unistd.h>
main()
{
    printf("Hello world from process ID %d\n", getpid());
}
```

와 같은 프로그램을 성공적으로 컴파일 하여 a.out이라는 파일이 생성되었으면  a.out이 프로그램이 된다

## PROCESS

- a.out을 실행하면 이것이 process가 된다
- $ a.out
  Hello world from process ID 851
- $ a.out
  Hello world from process ID 852
- 즉 process는 실행중인 program이며 이 들은 process id 로 구분된다.

# Process Descriptor



task_struct

state flags
priority
next_task
prev_task
next_run
prev_run
p_pptr
p_optr
...
tty
fs
files
mm
sig
...

tty_struct tty associated with the process

fs_struct current directory

files_struct pointers to file descriptors

mm_struct pointers to memory area descriptors

signal_struct signals received

# Process States

- process states
  - state of a process is defined by its current activity
  - executing
    - the process is being executed by the processor
  - ready
    - the process could be executed, but another process is currently being executed

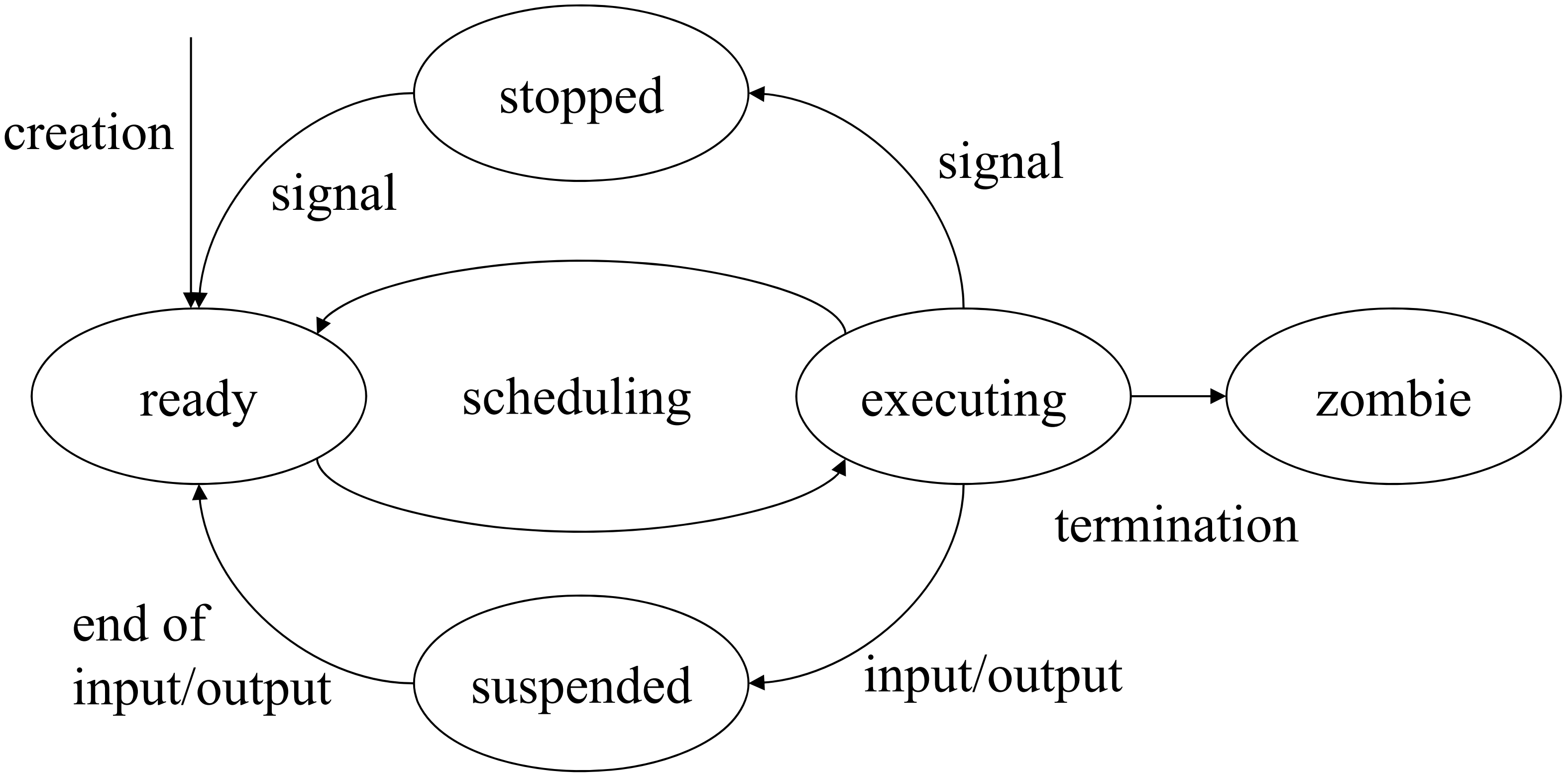
# Process States

- process states (cont’d)
  - suspended
    - the process is suspended (sleeping) until some condition becomes true.
      - hardware interrupt
      - releasing a system resource the process is waiting for
      - delivering a signals, etc.

# Process States

- process states (cont'd)
  - stopped
    - process execution has been stopped
    - caused by signals
      - SIGSTOP, SIGTSTP, SIGTTIN, SIGTTOU
  - zombie
    - the process has finished execution, but it is still referenced in the system

# Process States



# Attributes of a Process

- attributes
  - state
  - identification (unique number)
  - values of the registers, including the program counter
  - user identity under whose name the process is executing
  - information used by the kernel to establish the schedule of the processes
    - priority, execution time

# Attributes of a Process

- attributes (cont'd)
  - information concerning the address space of the process
    - segments for the code, data, stack
  - information concerning the inputs/outputs carried out by the process
    - descriptions of open files, current directory, ...
  - information summarizing resources used by the process

# User Identity

- user identifiers
  - real user id
    - identifier of the user who started up the process
  - effective user id
    - identifier which is used by the system for access control
    - can be different from the real user, especially in the case of programs with the setuid bit set