Unix, Posix, X/Open APIs

API Application Programmer’s Interface

Includes such things as

- System configuration
- File Manipulation
- Process Creation
- Process Control
- Interprocess Communications
- Network Communications

Unix, Posix, X/Open APIs

API Application Programmer's Interface

Includes such things as

System configuration	Process Control
File Manipulation	Interprocess Communications
Process Creation	Network Communications

Posix (1988) Portable Operating System Interface (1003.1) (a family of related standards) specifies Unix interfaces. (*not an implementation*)

Unix, Posix, X/Open APIs

API Application Programmer's Interface

Includes such things as

- System configuration
- File Manipulation
- Process Creation
- Process Control
- Interprocess Communications
- Network Communications

Posix (1988) Portable Operating System Interface (1003.1) (a family of related standards) specifies Unix interfaces. (*not an implementation*)

Ansi C (1989) the ANSI STD X3.159-1989 standard specifying the C programming language was approved. The international ISO/IEC 9899:1998 was approved simultaneously.

Unix, Posix, X/Open APIs

API Application Programmer's Interface

Includes such things as

- System configuration
- File Manipulation
- Process Creation
- Process Control
- Interprocess Communications
- Network Communications

Posix (1988) Portable Operating System Interface (1003.1) (a family of related standards) specifies Unix interfaces. (*not an implementation*)

Ansi C (1989) the ANSI STD X3.159-1989 standard specifying the C programming language was approved. The international ISO/IEC 9899:1998 was approved simultaneously.

X/Open a European consortium founded to identify and promote open standards; specifications published as the "X Portability Guide, Issue 3". Contains descriptions of additional features not in Posix. (XPG4, 1992, is the most recent)

Unix, Posix, X/Open APIs

API Application Programmer's Interface

Includes such things as

System configuration	Process Control
File Manipulation	Interprocess Communications
Process Creation	Network Communications

Posix (1988) Portable Operating System Interface (1003.1) (a family of related standards) specifies Unix interfaces. (*not an implementation*)

Ansi C (1989) the ANSI STD X3.159-1989 standard specifying the C programming language was approved. The international ISO/IEC 9899:1998 was approved simultaneously.

X/Open a European consortium founded to identify and promote open standards; specifications published as the "X Portability Guide, Issue 3". Contains descriptions of additional features not in Posix. (XPG4, 1992, is the most recent)

FIPS Federal Information Processing Standard. Requires some features that Posix has as optional.

Implementations

Unix Implementations include

7^{th} Edition	AT&T, 1979

Implementations

Unix Implementations include

7^{th} Edition	AT&T, 1979
System III,V	AT&T Commercial

Implementations

Unix Implementations include

7^{th} Edition	AT&T, 1979
System III,V	AT&T Commercial
4.x BSD	Berkeley Software Distribution, 1983,1986
	(*intent to produce an AT&T license-free distribution*)

Implementations

Unix Implementations include

7^{th} Edition	AT&T, 1979
System III,V	AT&T Commercial
4.x BSD	Berkeley Software Distribution, 1983,1986
	(*intent to produce an AT&T license-free distribution*)
Linux	Linus Torvald’s free Unix-like system with GNU utilities

Implementations

Unix Implementations include

7^{th} Edition	AT&T, 1979
System III,V	AT&T Commercial
4.x BSD	Berkeley Software Distribution, 1983,1986
	(*intent to produce an AT&T license-free distribution*)
Linux	Linus Torvald’s free Unix-like system with GNU utilities
NetBSD	BSD NET/2 distribution

Implementations

Unix Implementations include

7^{th} Edition	AT&T, 1979
System III,V	AT&T Commercial
4.x BSD	Berkeley Software Distribution, 1983,1986
	(intent to produce an AT&T license-free distribution)
Linux	Linus Torvald's free Unix-like system with GNU utilities
NetBSD	BSD NET/2 distribution
Hurd	Free Software Foundation's free unix (still in development)

Implementations

Unix Implementations include

7^{th} Edition	AT&T, 1979
System III,V	AT&T Commercial
4.x BSD	Berkeley Software Distribution, 1983,1986
	(*intent to produce an AT&T license-free distribution*)
Linux	Linus Torvald’s free Unix-like system with GNU utilities
NetBSD	BSD NET/2 distribution
Hurd	Free Software Foundation’s free unix (still in development)
Solaris	Sun

Implementations

Unix Implementations include

7^{th} Edition	AT&T, 1979
System III,V	AT&T Commercial
4.x BSD	Berkeley Software Distribution, 1983,1986
	(intent to produce an AT&T license-free distribution)
Linux	Linus Torvald’s free Unix-like system with GNU utilities
NetBSD	BSD NET/2 distribution
Hurd	Free Software Foundation’s free unix (still in development)
Solaris	Sun
Irix	Silicon Graphics

Definitions and API

Unix Kernel the “real” operating system (*as opposed to a shell*). Manages and implements the API, process switching and control, files, etc.

Definitions and API

Unix Kernel the "real" operating system *(as opposed to a shell)*. Manages and implements the API, process switching and control, files, etc.

kernel mode process has called a system kernel function; it is suspended while the kernel function execus. Maintains a wall between user processes and the o/s. When complete, the user process is made available for subsequent execution. Thus, using system resources is "expensive", requiring two context switches.

Definitions and API

Unix Kernel the “real” operating system *(as opposed to a shell)*. Manages and implements the API, process switching and control, files, etc.

kernel mode process has called a system kernel function; it is suspended while the kernel function execus. Maintains a wall between user processes and the o/s. When complete, the user process is made available for subsequent execution. Thus, using system resources is “expensive”, requiring two context switches.

API Return Values If an API function returns -1, then an error/failure occurred. A global integer variable `errno` is set to a constant indicating error status.

Example 1 *EPIPE: attempt to write to a “pipe” with no reader*

Example 2 *EACCESS: process does not have access permission to perform the requested operation*

Definitions and API

Unix Kernel the "real" operating system *(as opposed to a shell)*. Manages and implements the API, process switching and control, files, etc.

kernel mode process has called a system kernel function; it is suspended while the kernel function execus. Maintains a wall between user processes and the o/s. When complete, the user process is made available for subsequent execution. Thus, using system resources is "expensive", requiring two context switches.

API Return Values If an API function returns -1, then an error/failure occurred. A global integer variable `errno` is set to a constant indicating error status.

Example 1 *EPIPE: attempt to write to a "pipe" with no reader*

Example 2 *EACCESS: process does not have access permission to perform the requested operation*

Use the following to obtain an explanatory message:

perror("optional leader string") to print to stdout or

strerror("optional leader string") to return a string

Unix Limits

APIs and user programs often run under limitations

#define/const	compile time options	what features are available?
#define/const	compile time limits	lengths of short, long
query functions	run-time limits	how many chars in a filename? path?

Unix Limits

APIs and user programs often run under limitations

#define/const	compile time options	what features are available?
#define/const	compile time limits	lengths of short, long
query functions	run-time limits	how many chars in a filename? path?

	SystemV has historically allowed only	*14 chars in a filename*
Example 3	*BSD allows*	*255 chars in a filename*
	Linux allows	*4096 chars in a filename*

Unix Limits

APIs and user programs often run under limitations

#define/const	compile time options	what features are available?
#define/const	compile time limits	lengths of short, long
query functions	run-time limits	how many chars in a filename? path?

Example 3	*SystemV has historically allowed only*	*14 chars in a filename*
	BSD allows	*255 chars in a filename*
	Linux allows	*4096 chars in a filename*

Query functions include

sysconf()	non-file/directory limit
fpathconf()	file/directory related limit (filedes)
pathconf()	file/directory related limit (char*)

Using Unix Limit Functions

```
#include <unistd.h>
long sysconf(int name);
long pathconf(const char *pathname,int name);
long fpathconf(int fildes,int name);
```

All three functions return -1 and set `errno` upon failure.

name	*description*
_PC_MAX_CANON	: must be a terminal "file"
_PC_MAX_INPUT	: must be a terminal "file"
_PC_MAX_DISABLE	: must be a terminal "file"
_PC_LINK_MAX	: can be a file or directory
_PC_PATH_MAX	: must be a directory
_PC_APE_BUF	: must be a pipe, fifo, directory
	: (if directory, any fifo in that directory)
_PC_CHOWN_RESTRICTED	: must be either a file or a directory

(see more limits on website) (see seelimits.c)

Unix Filesystem

Unix/Posix/Linux files use seven types to handle a variety of information:

Unix Filesystem

Unix/Posix/Linux files use seven types to handle a variety of information:

Regular	(-)	text, binary, executables, shell scripts

Unix Filesystem

Unix/Posix/Linux files use seven types to handle a variety of information:

Regular	(-)	text, binary, executables, shell scripts
Directory	(d)	contains filenames and where-to-find information *(inode)*

Unix Filesystem

Unix/Posix/Linux files use seven types to handle a variety of information:

Regular	(-)	text, binary, executables, shell scripts
Directory	(d)	contains filenames and where-to-find information *(inode)*
Link	(l)	Hard links *(see later)*

Unix Filesystem

Unix/Posix/Linux files use seven types to handle a variety of information:

Regular	(-)	text, binary, executables, shell scripts
Directory	(d)	contains filenames and where-to-find information *(inode)*
Link	(l)	Hard links *(see later)*
Character Devices	(c)	for physical devices that transmit data a byte at a time.

Unix Filesystem

Unix/Posix/Linux files use seven types to handle a variety of information:

Regular	(-)	text, binary, executables, shell scripts
Directory	(d)	contains filenames and where-to-find information *(inode)*
Link	(l)	Hard links *(see later)*
Character Devices	(c)	for physical devices that transmit data a byte at a time.
Block Devices	(b)	for physical devices that transmit blocks of data, such as ha

Unix Filesystem

Unix/Posix/Linux files use seven types to handle a variety of information:

Regular	(-)	text, binary, executables, shell scripts
Directory	(d)	contains filenames and where-to-find information *(inode)*
Link	(l)	Hard links *(see later)*
Character Devices	(c)	for physical devices that transmit data a byte at a time.
Block Devices	(b)	for physical devices that transmit blocks of data, such as ha
Sockets	(s)	Interprocess communications.

Unix Filesystem

Unix/Posix/Linux files use seven types to handle a variety of information:

Regular	(-)	text, binary, executables, shell scripts
Directory	(d)	contains filenames and where-to-find information *(inode)*
Link	(l)	Hard links *(see later)*
Character Devices	(c)	for physical devices that transmit data a byte at a time.
Block Devices	(b)	for physical devices that transmit blocks of data, such as ha
Sockets	(s)	Interprocess communications.
Named Pipes	(p)	Interprocess communications.

(the letter in parentheses is shown by ls -lsa)

Unix Filesystem, con’t.

Note: some devices may have both block and character device files that can access them

Unix Filesystem, con't.

Note: some devices may have both block and character device files that can access them

Typically one makes filesystems with mknod and ln. Filesystems include

/ /usr /tmp swap /home

From a user's viewpoint, except for swap, all of these appear under / *(root)*.

Unix Filesystem, con't.

Note: some devices may have both block and character device files that can access them

Typically one makes filesystems with mknod and ln. Filesystems include

/ /usr /tmp swap /home

From a user's viewpoint, except for swap, all of these appear under / *(root)*.

A **filesystem** resides in one **partition**.

Unix Filesystem, con't.

Note: some devices may have both block and character device files that can access them

Typically one makes filesystems with mknod and ln. Filesystems include

/ /usr /tmp swap /home

From a user's viewpoint, except for swap, all of these appear under / *(root)*.

A **filesystem** resides in one **partition**.

mknod comes as both a system function and a shell command; it is used to create a file system node (file, device special file, or named pipe)

Unix Filesystem, con’t.

Note: some devices may have both block and character device files that can access them

Typically one makes filesystems with mknod and ln. Filesystems include

/	/usr	/tmp	swap	/home

From a user’s viewpoint, except for swap, all of these appear under / *(root)*.

A **filesystem** resides in one **partition**.

mknod comes as both a system function and a shell command; it is used to create a file system node (file, device special file, or named pipe)

ln, a shell command, makes hard links between files

Unix Filesystem, con't.

Note: some devices may have both block and character device files that can access them

Typically one makes filesystems with mknod and ln. Filesystems include

/ /usr /tmp swap /home

From a user's viewpoint, except for swap, all of these appear under / *(root)*.

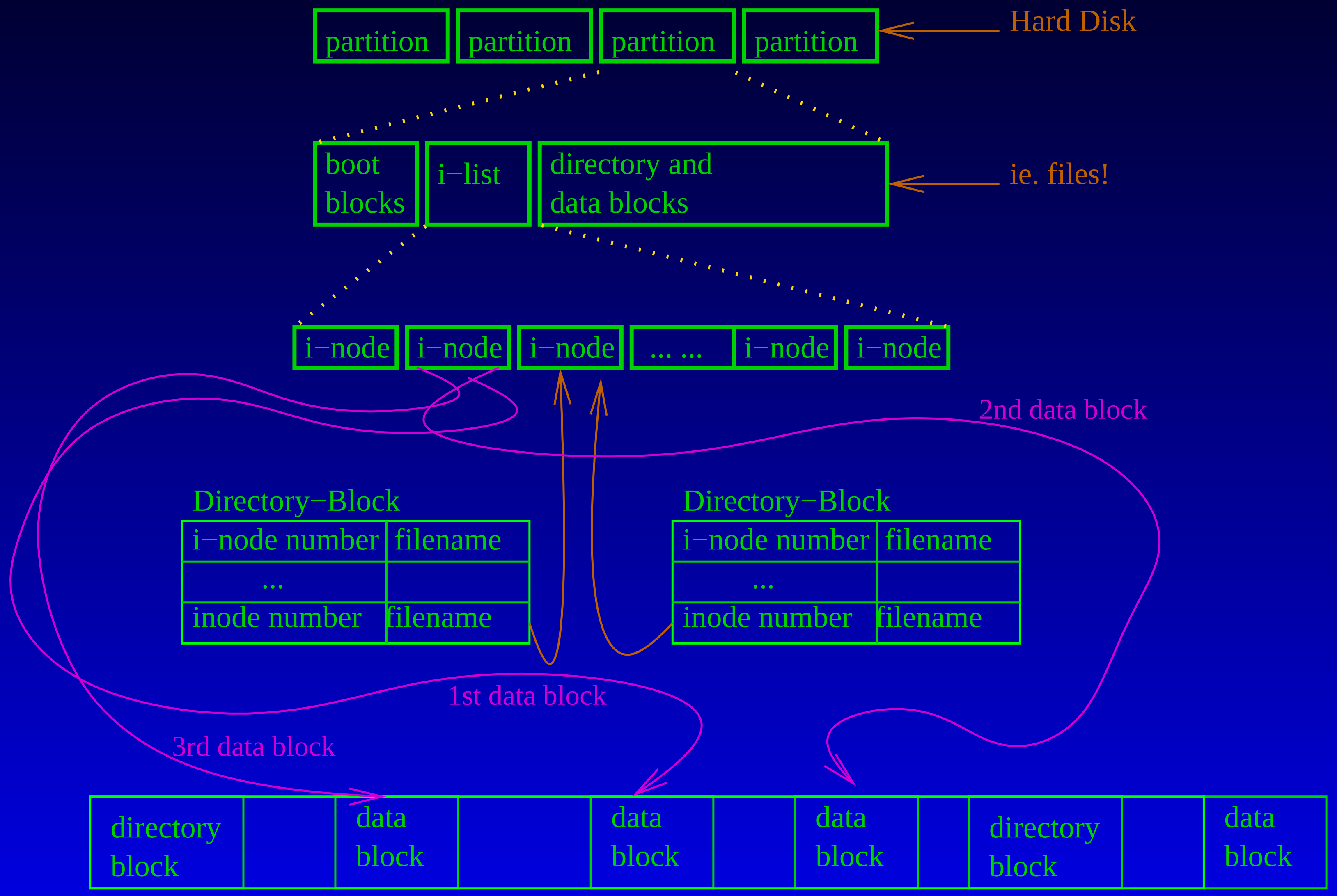
A **filesystem** resides in one **partition**.

mknod comes as both a system function and a shell command; it is used to create a file system node (file, device special file, or named pipe)

ln, a shell command, makes hard links between files

link, a system function, also makes hard links between files

Unix Filesystem: Hierarchy



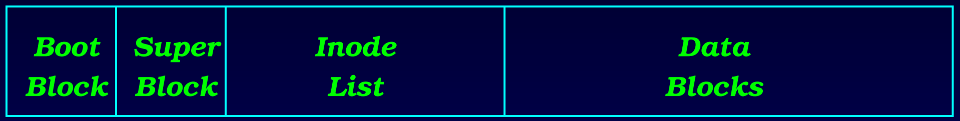
Directory−Block

i−node number	filename
...	
inode number	filename

Directory−Block

i−node number	filename
...	
inode number	filename

Unix Filesystem: Boot Blocks



Boot Block

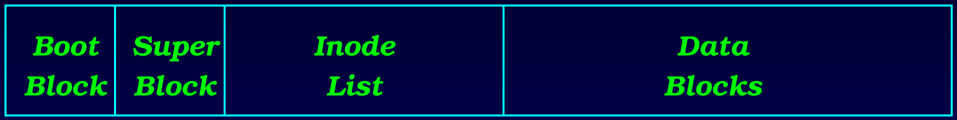
Super Block

Inode List

Data Blocks

- Boot blocks appear at the beginning of every file system

Unix Filesystem: Boot Blocks



Boot Block

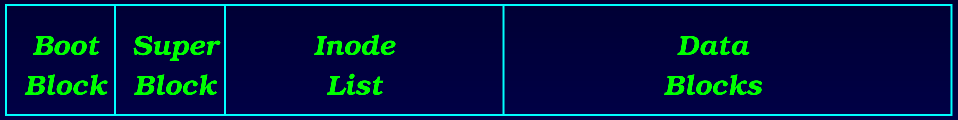
Super Block

Inode List

Data Blocks

- Boot blocks appear at the beginning of every file system
- Boot blocks may contain **bootstrap** code which is used to boot (initialize) the operating system

Unix Filesystem: Boot Blocks



Boot Block

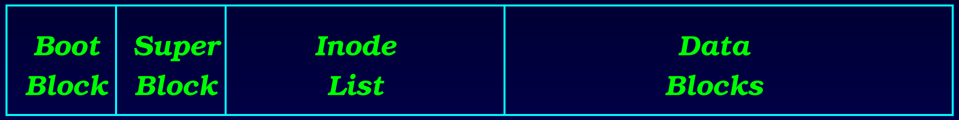
Super Block

Inode List

Data Blocks

- Boot blocks appear at the beginning of every file system
- Boot blocks may contain **bootstrap** code which is used to boot (initialize) the operating system
- Every filesystem has a (possibly empty) boot block

Unix Filesystem: Boot Blocks



Boot Block

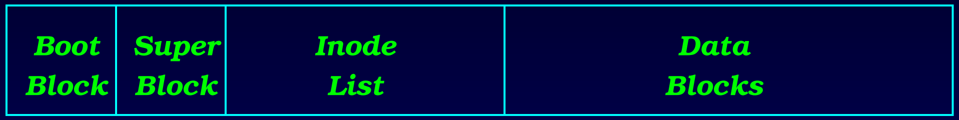
Super Block

Inode List

Data Blocks

- Boot blocks appear at the beginning of every file system
- Boot blocks may contain **bootstrap** code which is used to boot (initialize) the operating system
- Every filesystem has a (possibly empty) boot block
- The **superblock** contains metadata on the filesystem: how large it is, how many files it can store, where free space is on the file system, etc.

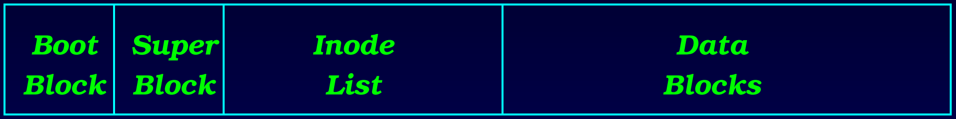
Unix Filesystem: Boot Blocks



Boot Block | Super Block | Inode List | Data Blocks

- Boot blocks appear at the beginning of every file system
- Boot blocks may contain **bootstrap** code which is used to boot (initialize) the operating system
- Every filesystem has a (possibly empty) boot block
- The **superblock** contains metadata on the filesystem: how large it is, how many files it can store, where free space is on the file system, etc.
- One i-node is the root inode of the file system; a mount system call makes it accessible

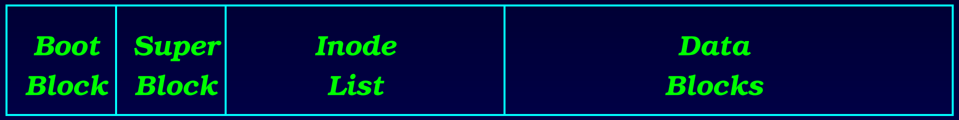
Unix Filesystem: Boot Blocks



Boot Block | Super Block | Inode List | Data Blocks

- Boot blocks appear at the beginning of every file system
- Boot blocks may contain **bootstrap** code which is used to boot (initialize) the operating system
- Every filesystem has a (possibly empty) boot block
- The **superblock** contains metadata on the filesystem: how large it is, how many files it can store, where free space is on the file system, etc.
- One i-node is the root inode of the file system; a mount system call makes it accessible
- Data blocks start at the end of the i-node list. An allocated data block will belong to only one file in the file system.

Unix Filesystem: Boot Blocks



Boot Block

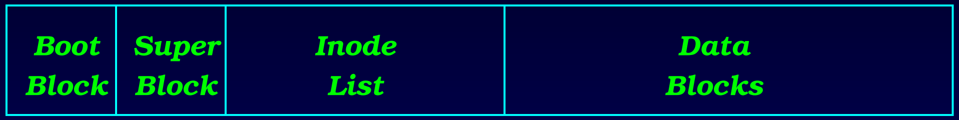
Super Block

Inode List

Data Blocks

- Boot blocks appear at the beginning of every file system
- Boot blocks may contain **bootstrap** code which is used to boot (initialize) the operating system
- Every filesystem has a (possibly empty) boot block
- The **superblock** contains metadata on the filesystem: how large it is, how many files it can store, where free space is on the file system, etc.
- One i-node is the root inode of the file system; a mount system call makes it accessible
- Data blocks start at the end of the i-node list. An allocated data block will belong to only one file in the file system.
- A file system's data blocks are all the same size *(typically some multiple of 512 bytes)*.

Unix Filesystem: Boot Blocks



Boot Block | Super Block | Inode List | Data Blocks

- Boot blocks appear at the beginning of every file system
- Boot blocks may contain **bootstrap** code which is used to boot (initialize) the operating system
- Every filesystem has a (possibly empty) boot block
- The **superblock** contains metadata on the filesystem: how large it is, how many files it can store, where free space is on the file system, etc.
- One i-node is the root inode of the file system; a mount system call makes it accessible
- Data blocks start at the end of the i-node list. An allocated data block will belong to only one file in the file system.
- A file system's data blocks are all the same size *(typically some multiple of 512 bytes)*.

  The larger a data block is, the more efficient data transfers typically are.

Unix Filesystem: Boot Blocks

Boot Block	*Super Block*	*Inode List*	*Data Blocks*

- Boot blocks appear at the beginning of every file system
- Boot blocks may contain **bootstrap** code which is used to boot (initialize) the operating system
- Every filesystem has a (possibly empty) boot block
- The **superblock** contains metadata on the filesystem: how large it is, how many files it can store, where free space is on the file system, etc.
- One i-node is the root inode of the file system; a mount system call makes it accessible
- Data blocks start at the end of the i-node list. An allocated data block will belong to only one file in the file system.
- A file system's data blocks are all the same size *(typically some multiple of 512 bytes)*.

  The larger a data block is, the more efficient data transfers typically are.

  The smaller a data block is, the more efficient storage space utilization is.

Unix Filesystem: Superblocks

- Superblocks contain an array of free i-node indices.

Unix Filesystem: Superblocks

- Superblocks contain an array of free i-node indices.
- There are often several copies scattered about the partition to facilitate recovery. Partitions with damaged superblocks cannot be mounted; *fsck* will often automatically run to recover the partition.

Unix Filesystem: Superblocks

- Superblocks contain an array of free i-node indices.
- There are often several copies scattered about the partition to facilitate recovery. Partitions with damaged superblocks cannot be mounted; *fsck* will often automatically run to recover the partition.
- Superblocks contain the mounting point *(file system status)*.

Unix Filesystem: Superblocks

- Superblocks contain an array of free i-node indices.
- There are often several copies scattered about the partition to facilitate recovery. Partitions with damaged superblocks cannot be mounted; *fsck* will often automatically run to recover the partition.
- Superblocks contain the mounting point *(file system status)*.
- When asked to provide an i-node for a new file, the kernel...

Unix Filesystem: Superblocks

- Superblocks contain an array of free i-node indices.
- There are often several copies scattered about the partition to facilitate recovery. Partitions with damaged superblocks cannot be mounted; *fsck* will often automatically run to recover the partition.
- Superblocks contain the mounting point *(file system status)*.
- When asked to provide an i-node for a new file, the kernel...
  - Searches the superblock's free inode list *(its not comprehensive)*

Unix Filesystem: Superblocks

- Superblocks contain an array of free i-node indices.
- There are often several copies scattered about the partition to facilitate recovery. Partitions with damaged superblocks cannot be mounted; *fsck* will often automatically run to recover the partition.
- Superblocks contain the mounting point *(file system status)*.
- When asked to provide an i-node for a new file, the kernel...
  - Searches the superblock's free inode list *(its not comprehensive)*
  - If the free inode list is empty, then the kernel searches the i-node list for free i-nodes

Unix Filesystem: Superblocks

- Superblocks contain an array of free i-node indices.
- There are often several copies scattered about the partition to facilitate recovery. Partitions with damaged superblocks cannot be mounted; *fsck* will often automatically run to recover the partition.
- Superblocks contain the mounting point *(file system status)*.
- When asked to provide an i-node for a new file, the kernel...
  - Searches the superblock's free inode list *(its not comprehensive)*
  - If the free inode list is empty, then the kernel searches the i-node list for free i-nodes
  - Every i-node contains a field: 0=free, not 0=used.

Unix Filesystem: Superblocks

- Superblocks contain an array of free i-node indices.
- There are often several copies scattered about the partition to facilitate recovery. Partitions with damaged superblocks cannot be mounted; *fsck* will often automatically run to recover the partition.
- Superblocks contain the mounting point *(file system status)*.
- When asked to provide an i-node for a new file, the kernel...
  - Searches the superblock's free inode list *(its not comprehensive)*
  - If the free inode list is empty, then the kernel searches the i-node list for free i-nodes
  - Every i-node contains a field: 0=free, not 0=used.
  - The kernel then proceeds to fill the superblock's free i-node list.

Unix Filesystem: Superblocks

- Superblocks contain an array of free i-node indices.
- There are often several copies scattered about the partition to facilitate recovery. Partitions with damaged superblocks cannot be mounted; *fsck* will often automatically run to recover the partition.
- Superblocks contain the mounting point *(file system status)*.
- When asked to provide an i-node for a new file, the kernel...
  - Searches the superblock's free inode list *(its not comprehensive)*
  - If the free inode list is empty, then the kernel searches the i-node list for free i-nodes
  - Every i-node contains a field: 0=free, not 0=used.
  - The kernel then proceeds to fill the superblock's free i-node list.
  - The kernel remembers the highest valued i-node used to fill the superblock's free i-node list; subsequent i-node list scans for free i-nodes will begin from there.

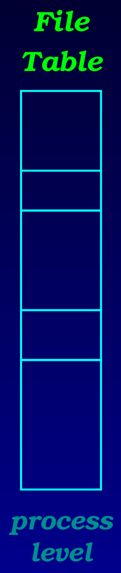
Unix Filesystem: Superblocks

- Superblocks contain an array of free i-node indices.
- There are often several copies scattered about the partition to facilitate recovery. Partitions with damaged superblocks cannot be mounted; *fsck* will often automatically run to recover the partition.
- Superblocks contain the mounting point *(file system status)*.
- When asked to provide an i-node for a new file, the kernel...
  - Searches the superblock's free inode list *(its not comprehensive)*
  - If the free inode list is empty, then the kernel searches the i-node list for free i-nodes
  - Every i-node contains a field: 0=free, not 0=used.
  - The kernel then proceeds to fill the superblock's free i-node list.
  - The kernel remembers the highest valued i-node used to fill the superblock's free i-node list; subsequent i-node list scans for free i-nodes will begin from there.
- When free-ing an i-node, if there's space in the superblock's list, the kernel will put it in there.

Unix Filesystem: Superblocks

- Superblocks contain an array of free i-node indices.
- There are often several copies scattered about the partition to facilitate recovery. Partitions with damaged superblocks cannot be mounted; *fsck* will often automatically run to recover the partition.
- Superblocks contain the mounting point *(file system status)*.
- When asked to provide an i-node for a new file, the kernel...
  - Searches the superblock's free inode list *(its not comprehensive)*
  - If the free inode list is empty, then the kernel searches the i-node list for free i-nodes
  - Every i-node contains a field: 0=free, not 0=used.
  - The kernel then proceeds to fill the superblock's free i-node list.
  - The kernel remembers the highest valued i-node used to fill the superblock's free i-node list; subsequent i-node list scans for free i-nodes will begin from there.
- When free-ing an i-node, if there's space in the superblock's list, the kernel will put it in there.
- The remembered i-node is updated with the newly free'd i-node if the free'd one is smaller.

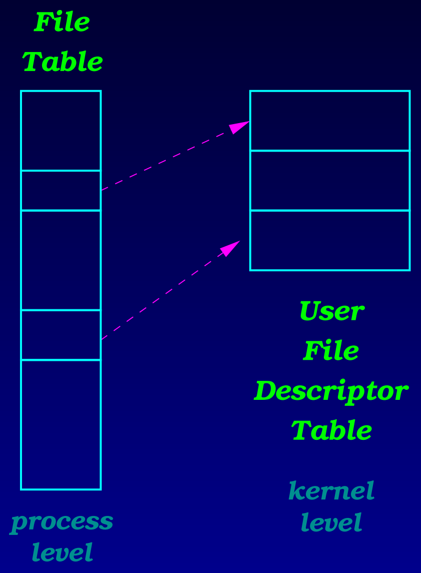
Unix Filesystem Overview



File Table (*process level*) Is allocated on a per process basis.

- It keeps track of where the next read/write will occur in the file.
- These indices are known as **file descriptors**.

Unix Filesystem Overview

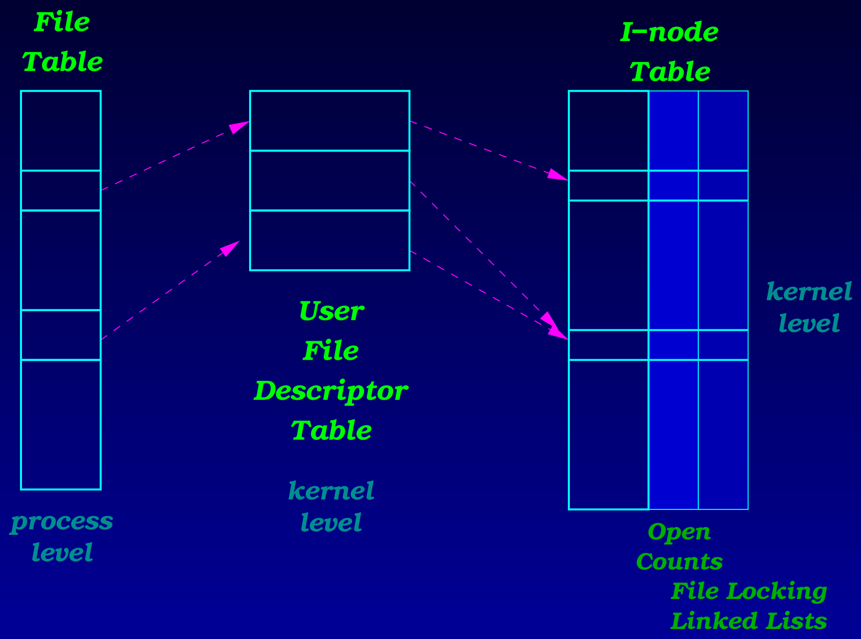


File Table (*process level*) Is allocated on a per process basis.

- It keeps track of where the next read/write will occur in the file.
- These indices are known as **file descriptors**.

User File Descriptor Table (*kernel level*) Holds indices into the user file descriptor table. Every open file has an entry in this kernel-level table.

Unix Filesystem Overview



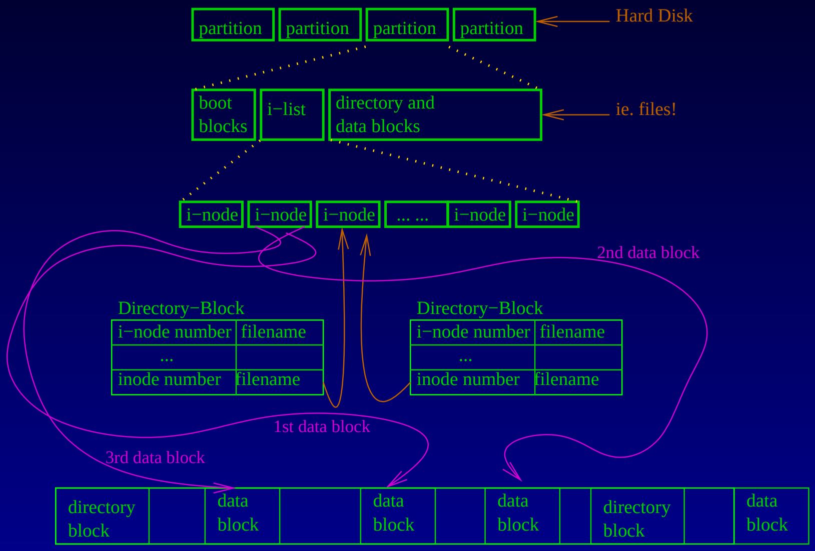
File Table (*process level*) Is allocated on a per process basis.

- It keeps track of where the next read/write will occur in the file.
- These indices are known as **file descriptors**.

User File Descriptor Table (*kernel level*) Holds indices into the user file descriptor table. Every open file has an entry in this kernel-level table.

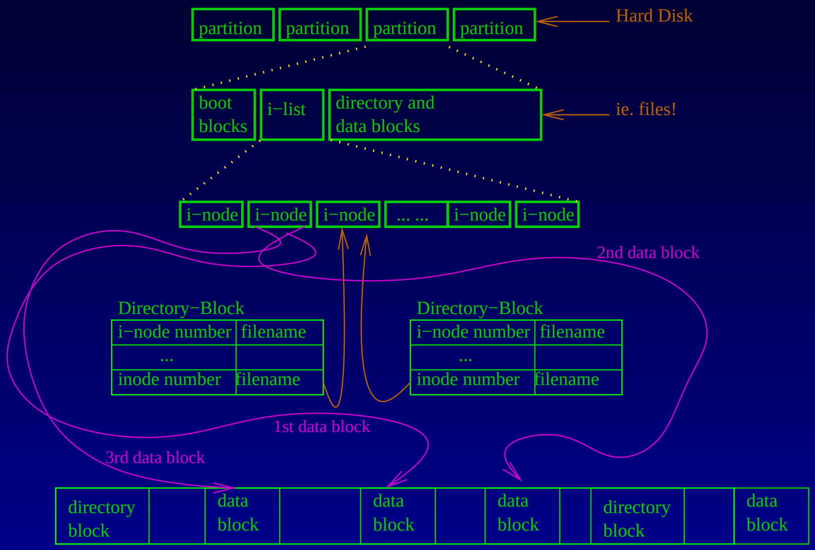
i-node Table (*kernel level*) Describes the file permissions, etc, and indexes associated data blocks.

Unix Filesystem: Inodes



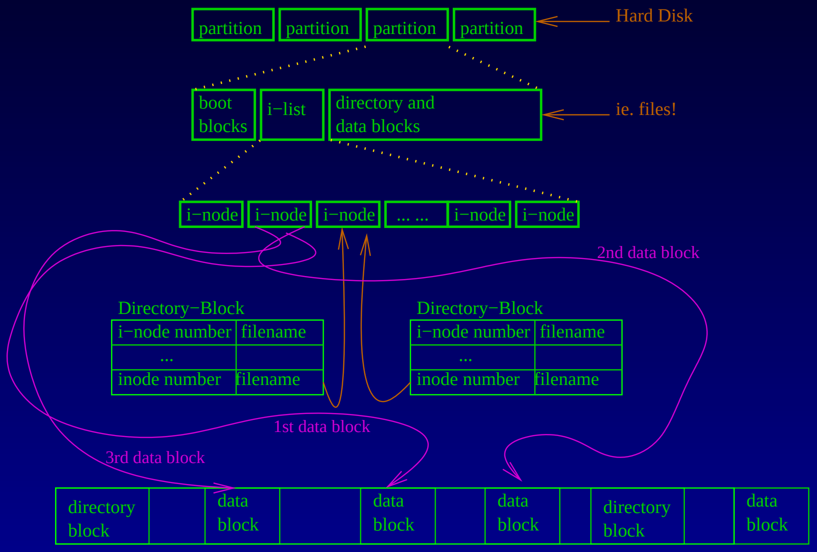
- Every i-node has a link count; in the diagram above, two directory blocks are shown with access to the same file (inode).

Unix Filesystem: Inodes



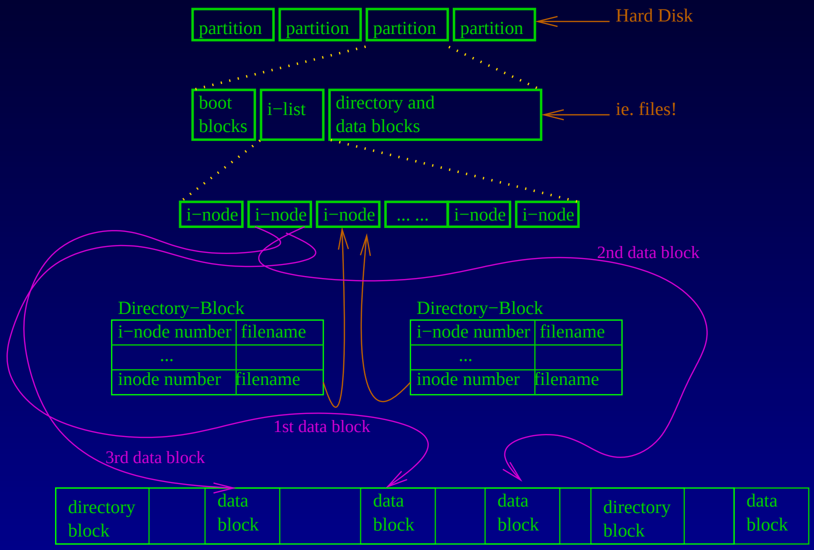
- Every i-node has a link count; in the diagram above, two directory blocks are shown with access to the same file (inode).
- Deletion of data blocks is *prevented* until the *link count* goes to zero.

Unix Filesystem: Inodes



- Every i-node has a link count; in the diagram above, two directory blocks are shown with access to the same file (inode).
- Deletion of data blocks is *prevented* until the *link count* goes to zero.
- I-nodes contain: file type, file access permission, size of file, pointers to data blocks for the file, etc.

Unix Filesystem: Inodes



partition partition partition partition

Hard Disk

boot blocks

i-list

directory and data blocks

ie. files!

i-node i-node i-node ... ... i-node i-node

2nd data block

Directory-Block

i-node number	filename
...	
inode number	filename

Directory-Block

i-node number	filename
...	
inode number	filename

1st data block

3rd data block

directory block

data block

data block

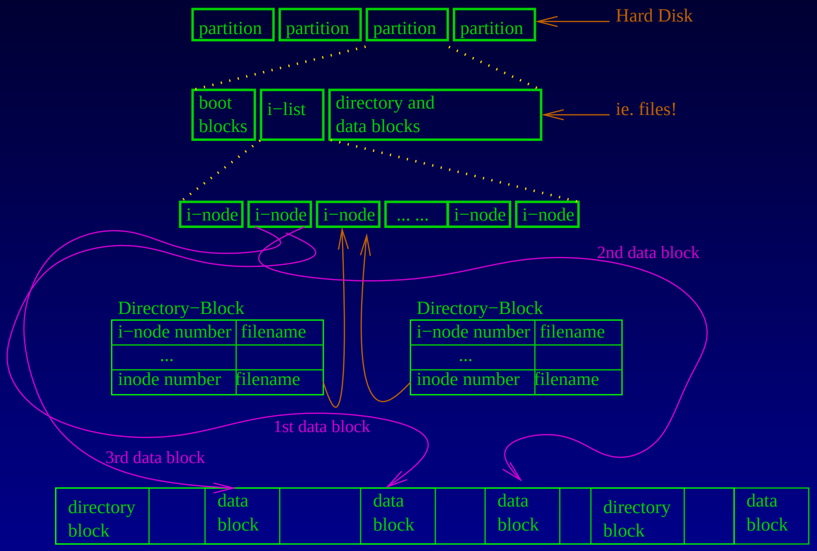
data block

directory block

data block

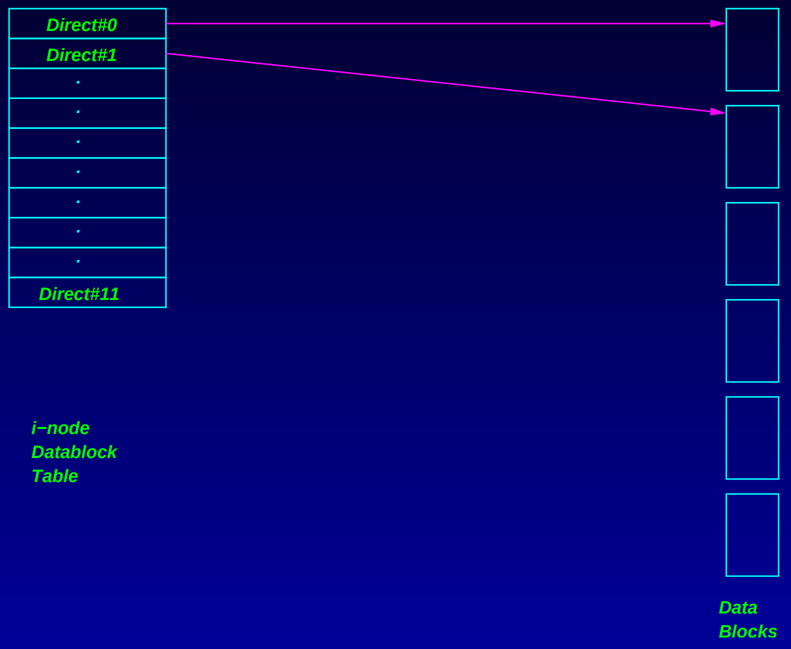
- Every i-node has a link count; in the diagram above, two directory blocks are shown with access to the same file (inode).
- Deletion of data blocks is *prevented* until the *link count* goes to zero.
- I-nodes contain: file type, file access permission, size of file, pointers to data blocks for the file, etc.
- Since i-node numbers are partition-oriented, one *cannot hard-link to other filesystems*.

Unix Filesystem: Inodes



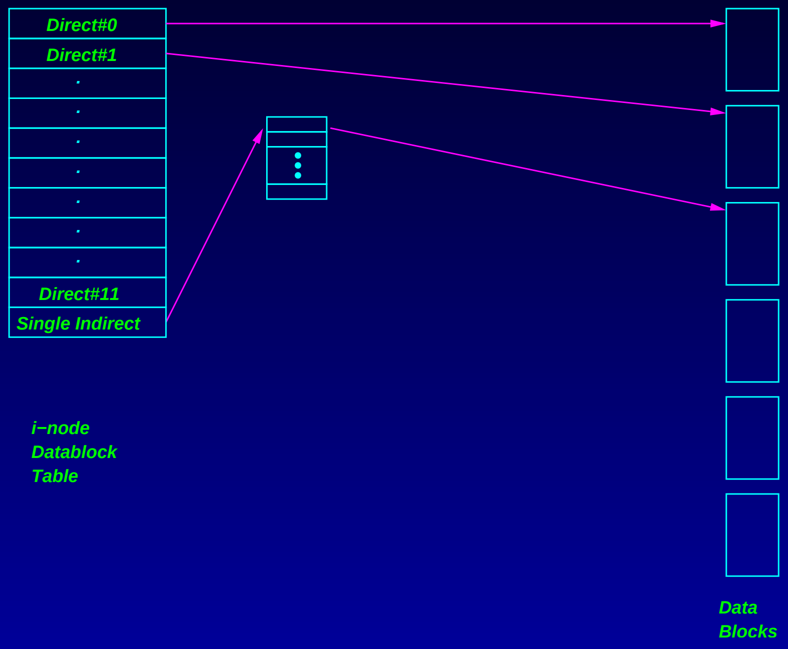
- Every i-node has a link count; in the diagram above, two directory blocks are shown with access to the same file (inode).
- Deletion of data blocks is *prevented* until the *link count* goes to zero.
- I-nodes contain: file type, file access permission, size of file, pointers to data blocks for the file, etc.
- Since i-node numbers are partition-oriented, one *cannot hard-link to other filesystems*.
- Use df -i to determine the quantities of used and free i-nodes on your system.

Inodes and Data Blocks



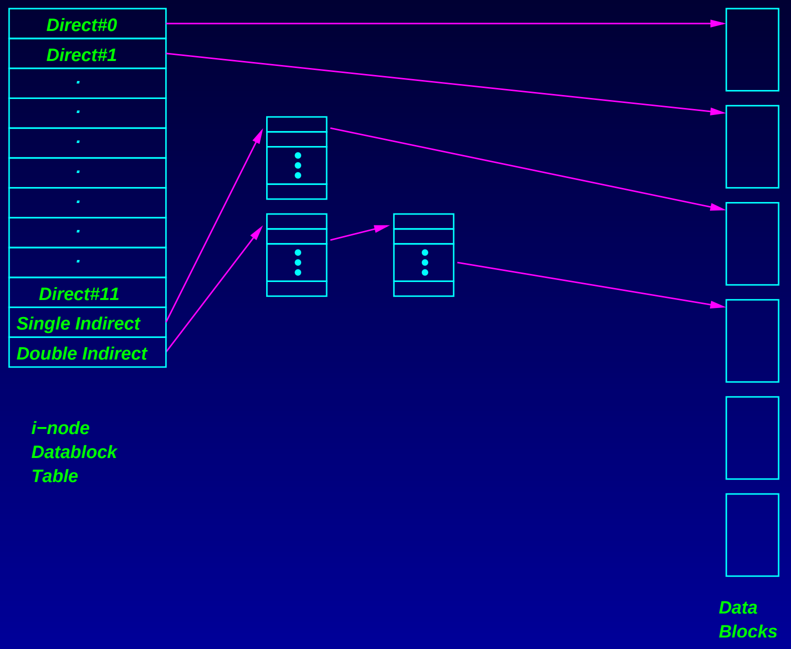
- Each i-node contains a table of twelve slots for **direct indexing** – each such slot may hold a direct index to a data block.

Inodes and Data Blocks



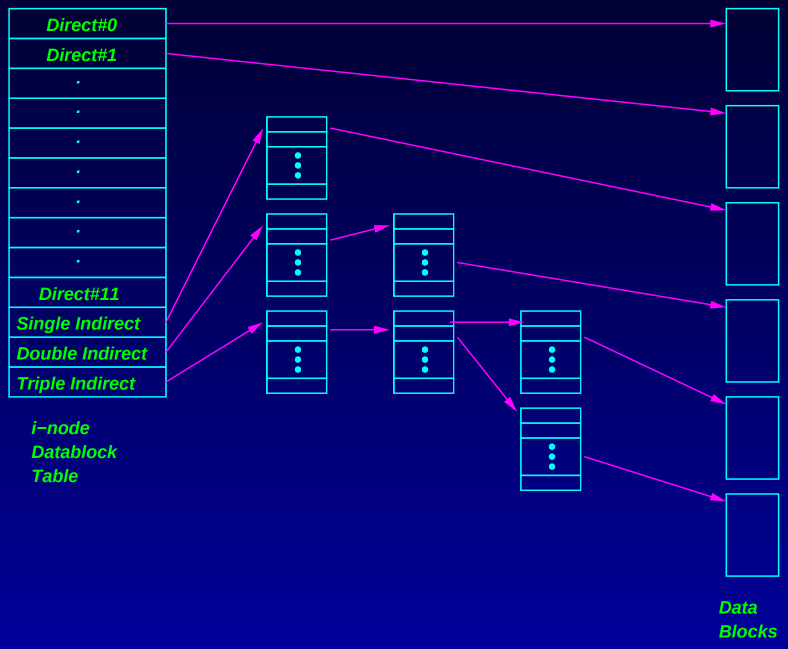
- Each i-node contains a table of twelve slots for **direct indexing** – each such slot may hold a direct index to a data block.
- If the file requires more than twelve data blocks, then the next slot is used: **single indirect**. This slot indexes a data block which contains nothing but a table of additional index slots for data blocks.

Inodes and Data Blocks



- Each i-node contains a table of twelve slots for **direct indexing** – each such slot may hold a direct index to a data block.
- If the file requires more than twelve data blocks, then the next slot is used: **single indirect**. This slot indexes a data block which contains nothing but a table of additional index slots for data blocks.
- If the single indirect method's slots are all filled up, then another slot is used, the **double indirect** slot. The double indirect slot indexes additional data blocks which each hold nothing but datablock index slots.

Inodes and Data Blocks



- Each i-node contains a table of twelve slots for **direct indexing** – each such slot may hold a direct index to a data block.
- If the file requires more than twelve data blocks, then the next slot is used: **single indirect**. This slot indexes a data block which contains nothing but a table of additional index slots for data blocks.
- If the single indirect method's slots are all filled up, then another slot is used, the **double indirect** slot. The double indirect slot indexes additional data blocks which each hold nothing but datablock index slots.
- Finally, if the double indirect slots are exhausted, the **triple indirect** indexing is used.

Hard Links

```
ln target newname
```

- **hard links** create a new entry in a directory block to the same i-node that the target has

Hard Links

ln target newname

- **hard links** create a new entry in a directory block to the same i-node that the target has
- A directory or a file may, thereby, be referred to by different names

Hard Links

```
ln target newname
```

- **hard links** create a new entry in a directory block to the same i-node that the target has
- A directory or a file may, thereby, be referred to by different names
- Programs may modify their behavior based on which name is used to call them:

```
int main(int argc,char **argv)
{
printf("argv[0]<%s>\n",argv[0]); }
```

Hard Links

```
ln target newname
```

- **hard links** create a new entry in a directory block to the same i-node that the target has
- A directory or a file may, thereby, be referred to by different names
- Programs may modify their behavior based on which name is used to call them:

  ```
  int main(int argc,char **argv)
  {
  printf("argv[0]<%s>\n",argv[0]); }
  ```

- Making a hard link increments the target inode’s *link count*

Hard Links

```
ln target newname
```

- **hard links** create a new entry in a directory block to the same i-node that the target has
- A directory or a file may, thereby, be referred to by different names
- Programs may modify their behavior based on which name is used to call them:

  ```
  int main(int argc,char **argv)
  {
  printf("argv[0]<%s>\n",argv[0]); }
  ```

- Making a hard link increments the target inode’s *link count*
- There is only one copy of the file’s contents on the disk!

Symbolic Links

A **symbolic link** is a special file which contains a path name which may reference another file on any filesystem. (SYS-V, BSD, not Posix, Linux)

Symbolic Links

A **symbolic link** is a special file which contains a path name which may reference another file on any filesystem. (SYS-V, BSD, not Posix, Linux)

To create: `ln -s -real-path-filename- -fake-path-filename-`

Symbolic Links

A **symbolic link** is a special file which contains a path name which may reference another file on any filesystem. (SYS-V, BSD, not Posix, Linux)

To create:	`ln -s -real-path-filename- -fake-path-filename-`
To remove:	`rm -symbolic-link-filename-`

Symbolic Links

A **symbolic link** is a special file which contains a path name which may reference another file on any filesystem. (SYS-V, BSD, not Posix, Linux)

To create: `ln -s -real-path-filename- -fake-path-filename-`

To remove: `rm -symbolic-link-filename-`

Programs may modify their behavior based on which name is used to call them. Examples: vi/view/gvim, sh/-, ksh/-, csh/-, compress/decompress, etc

Device Files

To make a **character device** file: use

```
mknod /dev/cdsk c major-device-number minor-device-number.
```

Device Files

To make a **character device** file: use

```
mknod /dev/cdsk c major-device-number minor-device-number.
```

major device number index to a kernel table of all device driver functions known to the system

minor device number data passed on to the device driver; the driver can use this to modify its behavior

Device Files

To make a **character device** file: use

`mknod /dev/cdsk c major-device-number minor-device-number`.

major device number index to a kernel table of all device driver functions known to the system

minor device number data passed on to the device driver; the driver can use this to modify its behavior

To make a **block device** file: use

`mknod /dev/bdsk b major minor`.

Device Files

To make a **character device** file: use

```
mknod /dev/cdsk c major-device-number minor-device-number.
```

major device number index to a kernel table of all device driver functions known to the system

minor device number data passed on to the device driver; the driver can use this to modify its behavior

To make a **block device** file: use

```
mknod /dev/bdsk b major minor.
```

Look at /etc/fstab for a list of filesystems.

Device Files

To make a **character device** file: use

mknod /dev/cdsk c major-device-number minor-device-number.

major device number index to a kernel table of all device driver functions known to the system

minor device number data passed on to the device driver; the driver can use this to modify its behavior

To make a **block device** file: use

mknod /dev/bdsk b major minor.

Look at /etc/fstab for a list of filesystems.

These two uses of mknod are available *only to the superuser*

(*the superuser is the system administrator; ie. root*)

Understanding /etc/fstab

or, *how to mount filesystems*

The format of fstab will resemble:

filesystem	*mount point*	*type*	*options*	*frequency*	*passno*
block device	directory	ext3	rw	1=dumping support	controls fsck
remote filesystem	path	ext2	ro	0=no dumping	0=no check
		ntfs	noauto		1=root fs
		proc	grpid		2=other fs
		vfat	nodev		
		(etc)	raw		
			(etc)		

(check the mount command for more on this)

/etc/fstab: an example

```
/dev/mapper/vg_xorn-lv_home /home ext4 defaults 1 2
```

/dev/mapper/vg_xorn-lv_home   This device actually is linked to /dev/dm-2.
These are block-oriented, read-write drivers.

/etc/fstab: an example

```
/dev/mapper/vg_xorn-lv_home /home ext4 defaults 1 2
```

/dev/mapper/vg_xorn-lv_home
This device actually is linked to /dev/dm-2. These are block-oriented, read-write drivers.

/home
This is the mounting point. /home would be an actual directory if there was no device mounted against it.

/etc/fstab: an example

```
/dev/mapper/vg_xorn-lv_home /home ext4 defaults 1 2
```

/dev/mapper/vg_xorn-lv_home This device actually is linked to /dev/dm-2. These are block-oriented, read-write drivers.

/home This is the mounting point. /home would be an actual directory if there was no device mounted against it.

ext4 The type of filesystem in use (handles 1 EiB and 16TiB files)

/etc/fstab: an example

```
/dev/mapper/vg_xorn-lv_home /home ext4 defaults 1 2
```

/dev/mapper/vg_xorn-lv_home	This device actually is linked to /dev/dm-2. These are block-oriented, read-write drivers.
/home	This is the mounting point. /home would be an actual directory if there was no device mounted against it.
ext4	The type of filesystem in use (handles 1 EiB and 16TiB files)
defaults	Default filesystem settings For Ext3 file systems: equivalent to rw,suid,dev,exec,auto,nouser,async

/etc/fstab: an example

```
/dev/mapper/vg_xorn-lv_home /home ext4 defaults 1 2
```

/dev/mapper/vg_xorn-lv_home	This device actually is linked to /dev/dm-2. These are block-oriented, read-write drivers.
/home	This is the mounting point. /home would be an actual directory if there was no device mounted against it.
ext4	The type of filesystem in use (handles 1 EiB and 16TiB files)
defaults	Default filesystem settings For Ext3 file systems: equivalent to rw,suid,dev,exec,auto,nouser,async
1	dump frequency controls archiving schedule for the partition (see man dump)

/etc/fstab: an example

```
/dev/mapper/vg_xorn-lv_home /home ext4 defaults 1 2
```

/dev/mapper/vg_xorn-lv_home	This device actually is linked to /dev/dm-2. These are block-oriented, read-write drivers.
/home	This is the mounting point. /home would be an actual directory if there was no device mounted against it.
ext4	The type of filesystem in use (handles 1 EiB and 16TiB files)
defaults	Default filesystem settings For Ext3 file systems: equivalent to rw,suid,dev,exec,auto,nouser,async
1	dump frequency controls archiving schedule for the partition (see man dump)
2	Controls the order in which fsck checks the device/partition for errors at boot time. The root device should be 1. Other partitions should be either 2: to check after root or 0: to disable checking for that partition altogether.

Device Names

/dev/fd0	first floppy disk drive
/dev/fb0	first framebuffer drive. The framebuffer is a character device and is on major node 29 and minor 0.
/dev/hda	The master IDE drive on the primary IDE controller.
/dev/hdb	The slave drive on the primary controller.
/dev/hdc	and /dev/hdd are the master and slave devices on the secondary controller.
/dev/ht0	First IDE tape drive
/dev/js0	First analog joy stick (subsequent ones are /dev/js1, /dev/js2, etc). They are character devices on major node 15. The analogue joysticks start at minor node 0 and go up to 127. Digital joysticks start at minor node 128.
/dev/loop0	first loopback device
/dev/lp	line printer
/dev/md0	First metadisk group. Metadisks are related to RAID (Redundant Array of Independent Disks) devices. (see http://www.tldp.org/HOWTO/Software-RAID-HOWTO.html). It is a character device on major node 14, minor node 0.
/dev/mixer	part of the OSS (Open Sound System) driver (see http://www.opensound.com)
/dev/null	the bit bucket/trashcan. It is a character device on major node 1, minor node 3.

Device Names , con't.

/dev/pda — Parallel port IDE disks. Named similarly to disks on the internal IDE controllers (/dev/hd*).

/dev/pda — Parallel port IDE disks. Named similarly to internal IDE controllers (/dev/hd*)
Block devices on major node 45. Minor nodes:
The first device is /dev/pda and it is on minor node 0.

/dev/pcd0 — parallel port CD ROM drive. These are numbered from 0 onwards.
All are block devices on major node 46.
/dev/pcd0 is on minor node 0; subsequent drives use minor nodes 1, 2, 3 etc.

/dev/pt0 — Parallel port tape devices. Tapes do not have partitions so these are just numbered sequentially. They are character devices on major node 96.
The minor node numbers start from 0 for /dev/pt0 , etc.

/dev/parport0 — The raw parallel ports. Most devices needing parallel ports have their own drivers. This device permits direct access to the port.
It is a character device on major node 99 with minor node 0.
Subsequent devices after the first are numbered sequentially, incrementing the minor node.

/dev/random or /dev/urandom — These are kernel random number generators.
/dev/random is a non-deterministic generator (cannot guess next number)
/dev/urandom works similarly, but will return numbers using pseudo-random number generator after system entropy is used up.

Device Names , con’t.

/dev/sda The first SCSI drive on the first SCSI bus. The following drives are named similar to IDE drives. /dev/sdb is the second SCSI drive, /dev/sdc is the third SCSI drive, and so forth.

/dev/tty## terminals

/dev/ttyS0 The first serial port. Many times this it the port used to connect an external modem to your syster

/dev/ttyUSB USB serial converters, modems

/dev/zero This is a simple way of getting many 0s. Every time you read from this device it will return 0. This can be useful sometimes, for example when you want a file of fixed length but don’t really c

FIFO files

FIFO files are *named pipes*: they provide a temporary buffer for one process to write into and for another process to read.

FIFO files

FIFO files are *named pipes*: they provide a temporary buffer for one process to write into and for another process to read.

Both processes must have opened a FIFO file (one read, one write) *before any data passes through*.

First-in, first-out, like a queue.

FIFO files

FIFO files are *named pipes*: they provide a temporary buffer for one process to write into and for another process to read.

Both processes must have opened a FIFO file (one read, one write) *before any data passes through*.

First-in, first-out, like a queue.

To create: `mkfifo -path-to-file-` (SYS-V,BSD,linux)

FIFO files

FIFO files are *named pipes*: they provide a temporary buffer for one process to write into and for another process to read.

Both processes must have opened a FIFO file (one read, one write) *before any data passes through*.

First-in, first-out, like a queue.

To create:	`mkfifo -path-to-file-`	(SYS-V,BSD,linux)
	`mknod -path-to-file- p`	(SYS-V,linux)

FIFO files

FIFO files are *named pipes*: they provide a temporary buffer for one process to write into and for another process to read.

Both processes must have opened a FIFO file (one read, one write) *before any data passes through*.

First-in, first-out, like a queue.

To create:	mkfifo -path-to-file-	(SYS-V,BSD,linux)
	mknod -path-to-file- p	(SYS-V,linux)
To remove:	rm -path-to-file-	

Filesystems, con't.

. *current subdirectory*

.. *parent subdirectory*

Posix doesn't require actual files named . and .., but relative path names must resolve correctly as if they were actual files

Filesystems, con’t.

. *current subdirectory*

.. *parent subdirectory*

Posix doesn’t require actual files named . and .., but relative path names must resolve correctly as if they were actual files

A filename may not exceed NAME_MAX chars.

Filesystems, con't.

. *current subdirectory*

.. *parent subdirectory*

Posix doesn't require actual files named . and .., but relative path names must resolve correctly as if they were actual files

A filename may not exceed NAME_MAX chars.

A pathname may not exceed PATH_MAX chars.

Filesystems, con’t.

. *current subdirectory*

.. *parent subdirectory*

Posix doesn’t require actual files named . and .., but relative path names must resolve correctly as if they were actual files

A filename may not exceed NAME_MAX chars.

A pathname may not exceed PATH_MAX chars.

Note: on some systems, these values are *indeterminate*. This condition is indicated by (f)pathconf() returning -1 but errno is not set.

Filesystems, con't.

. *current subdirectory*

.. *parent subdirectory*

Posix doesn't require actual files named . and .., but relative path names must resolve correctly as if they were actual files

A filename may not exceed NAME_MAX chars.

A pathname may not exceed PATH_MAX chars.

Note: on some systems, these values are *indeterminate*. This condition is indicated by (f)pathconf() returning -1 but errno is not set.

These macros may not be the real maximum lengths if they're actually indeterminate.

Some Common Directories and Files

/etc : sysadmin files, some programs

Some Common Directories and Files

/etc : sysadmin files, some programs

/etc/passwd : user info (passwords, office, login, real names, etc)

Some Common Directories and Files

/etc	: sysadmin files, some programs
/etc/passwd	: user info (passwords, office, login, real names, etc)
/etc/group	: associates users with groups

Some Common Directories and Files

/etc	: sysadmin files, some programs
/etc/passwd	: user info (passwords, office, login, real names, etc)
/etc/group	: associates users with groups
/bin	: location of system binaries (programs)

Some Common Directories and Files

/etc	: sysadmin files, some programs
/etc/passwd	: user info (passwords, office, login, real names, etc)
/etc/group	: associates users with groups
/bin	: location of system binaries (programs)
/usr/bin	: location of system binaries (programs)

Some Common Directories and Files

/etc	: sysadmin files, some programs
/etc/passwd	: user info (passwords, office, login, real names, etc)
/etc/group	: associates users with groups
/bin	: location of system binaries (programs)
/usr/bin	: location of system binaries (programs)
/usr/include	: location of standard header files

Some Common Directories and Files

/etc	: sysadmin files, some programs
/etc/passwd	: user info (passwords, office, login, real names, etc)
/etc/group	: associates users with groups
/bin	: location of system binaries (programs)
/usr/bin	: location of system binaries (programs)
/usr/include	: location of standard header files
/usr/lib	: location of libraries (*.a, *.so)

Some Common Directories and Files

/etc : sysadmin files, some programs

/etc/passwd : user info (passwords, office, login, real names, etc)

/etc/group : associates users with groups

/bin : location of system binaries (programs)

/usr/bin : location of system binaries (programs)

/usr/include : location of standard header files

/usr/lib : location of libraries (*.a, *.so)

/usr/local : location of locally installed files and directories

Some Common Directories and Files

/etc	: sysadmin files, some programs
/etc/passwd	: user info (passwords, office, login, real names, etc)
/etc/group	: associates users with groups
/bin	: location of system binaries (programs)
/usr/bin	: location of system binaries (programs)
/usr/include	: location of standard header files
/usr/lib	: location of libraries (*.a, *.so)
/usr/local	: location of locally installed files and directories
/tmp	: temporary files

Some Common Directories and Files

/etc	: sysadmin files, some programs
/etc/passwd	: user info (passwords, office, login, real names, etc)
/etc/group	: associates users with groups
/bin	: location of system binaries (programs)
/usr/bin	: location of system binaries (programs)
/usr/include	: location of standard header files
/usr/lib	: location of libraries (*.a, *.so)
/usr/local	: location of locally installed files and directories
/tmp	: temporary files
/etc/fstab	: filesystem mounting table

Some Common Directories and Files

/etc : sysadmin files, some programs

/etc/passwd : user info (passwords, office, login, real names, etc)

/etc/group : associates users with groups

/bin : location of system binaries (programs)

/usr/bin : location of system binaries (programs)

/usr/include : location of standard header files

/usr/lib : location of libraries (*.a, *.so)

/usr/local : location of locally installed files and directories

/tmp : temporary files

/etc/fstab : filesystem mounting table

/boot : (linux) contains boot files (the kernel, grub, etc)

Some Common Directories and Files

/etc : sysadmin files, some programs

/etc/passwd : user info (passwords, office, login, real names, etc)

/etc/group : associates users with groups

/bin : location of system binaries (programs)

/usr/bin : location of system binaries (programs)

/usr/include : location of standard header files

/usr/lib : location of libraries (*.a, *.so)

/usr/local : location of locally installed files and directories

/tmp : temporary files

/etc/fstab : filesystem mounting table

/boot : (linux) contains boot files (the kernel, grub, etc)

/proc : (linux) files here actually map to memory

Unix/Posix/Linux/etc File Attributes

or, *what’s in an inode anyway?*

*	filetype	regular,dir,char dev,block dev,local socket,named pipe,symlink

Unix/Posix/Linux/etc File Attributes

or, *what's in an inode anyway?*

*	filetype	regular,dir,char dev,block dev,local socket,named pipe,symlink
	access permission	owner,group,other file access

Unix/Posix/Linux/etc File Attributes

or, *what's in an inode anyway?*

*	filetype	regular,dir,char dev,block dev,local socket,named pipe,symlink
	access permission	owner,group,other file access
	hard link count	qty hard links to this file

Unix/Posix/Linux/etc File Attributes

or, *what's in an inode anyway?*

*	filetype	regular,dir,char dev,block dev,local socket,named pipe,symlink
	access permission	owner,group,other file access
	hard link count	qty hard links to this file
	user id	file's user identifier

Unix/Posix/Linux/etc File Attributes

or, *what's in an inode anyway?*

*	filetype	regular,dir,char dev,block dev,local socket,named pipe,symlink
	access permission	owner,group,other file access
	hard link count	qty hard links to this file
	user id	file's user identifier
	group id	file's group identifier

Unix/Posix/Linux/etc File Attributes

or, *what’s in an inode anyway?*

*	filetype	regular,dir,char dev,block dev,local socket,named pipe,symlink
	access permission	owner,group,other file access
	hard link count	qty hard links to this file
	user id	file’s user identifier
	group id	file’s group identifier
	file size	in bytes

Unix/Posix/Linux/etc File Attributes

or, *what's in an inode anyway?*

*	filetype	regular,dir,char dev,block dev,local socket,named pipe,symlink
	access permission	owner,group,other file access
	hard link count	qty hard links to this file
	user id	file's user identifier
	group id	file's group identifier
	file size	in bytes
	last access time	last access time, in seconds since 00:00 UTC, Jan 1, 1970

Unix/Posix/Linux/etc File Attributes

or, *what's in an inode anyway?*

*	filetype	regular,dir,char dev,block dev,local socket,named pipe,symlink
	access permission	owner,group,other file access
	hard link count	qty hard links to this file
	user id	file's user identifier
	group id	file's group identifier
	file size	in bytes
	last access time	last access time, in seconds since 00:00 UTC, Jan 1, 1970
	last modified time	last modified time, in seconds since 00:00 UTC, Jan 1, 1970

Unix/Posix/Linux/etc File Attributes

or, *what's in an inode anyway?*

*	filetype	regular,dir,char dev,block dev,local socket,named pipe,symlink
	access permission	owner,group,other file access
	hard link count	qty hard links to this file
	user id	file's user identifier
	group id	file's group identifier
	file size	in bytes
	last access time	last access time, in seconds since 00:00 UTC, Jan 1, 1970
	last modified time	last modified time, in seconds since 00:00 UTC, Jan 1, 1970
	last change time	file access, uid, gid, hard link changed

Unix/Posix/Linux/etc File Attributes

or, *what’s in an inode anyway?*

*	filetype	regular,dir,char dev,block dev,local socket,named pipe,symlink
	access permission	owner,group,other file access
	hard link count	qty hard links to this file
	user id	file’s user identifier
	group id	file’s group identifier
	file size	in bytes
	last access time	last access time, in seconds since 00:00 UTC, Jan 1, 1970
	last modified time	last modified time, in seconds since 00:00 UTC, Jan 1, 1970
	last change time	file access, uid, gid, hard link changed
*	inode number	system inode of file, in the directory file

Unix/Posix/Linux/etc File Attributes

or, *what’s in an inode anyway?*

*	filetype	regular,dir,char dev,block dev,local socket,named pipe,symlink
	access permission	owner,group,other file access
	hard link count	qty hard links to this file
	user id	file’s user identifier
	group id	file’s group identifier
	file size	in bytes
	last access time	last access time, in seconds since 00:00 UTC, Jan 1, 1970
	last modified time	last modified time, in seconds since 00:00 UTC, Jan 1, 1970
	last change time	file access, uid, gid, hard link changed
*	inode number	system inode of file, in the directory file
*	file system id	which filesystem its on

Unix/Posix/Linux/etc File Attributes

or, *what's in an inode anyway?*

*	filetype	regular,dir,char dev,block dev,local socket,named pipe,symlink
	access permission	owner,group,other file access
	hard link count	qty hard links to this file
	user id	file's user identifier
	group id	file's group identifier
	file size	in bytes
	last access time	last access time, in seconds since 00:00 UTC, Jan 1, 1970
	last modified time	last modified time, in seconds since 00:00 UTC, Jan 1, 1970
	last change time	file access, uid, gid, hard link changed
*	inode number	system inode of file, in the directory file
*	file system id	which filesystem its on
*	major/minor dev #	device files only

* : unchanged after file is created

Unix/Posix/Linux/etc File Attributes

or, *what’s in an inode anyway?*

*	filetype	regular,dir,char dev,block dev,local socket,named pipe,symlink
	access permission	owner,group,other file access
	hard link count	qty hard links to this file
	user id	file’s user identifier
	group id	file’s group identifier
	file size	in bytes
	last access time	last access time, in seconds since 00:00 UTC, Jan 1, 1970
	last modified time	last modified time, in seconds since 00:00 UTC, Jan 1, 1970
	last change time	file access, uid, gid, hard link changed
*	inode number	system inode of file, in the directory file
*	file system id	which filesystem its on
*	major/minor dev #	device files only

* : unchanged after file is created

These attributes are assigned by the kernel when the file is created.

Commands&Functions Affecting Attributes

Unix Command	*System Call*	*Attrib Modified*
chmod	chmod	access, last time chgd

Commands&Functions Affecting Attributes

Unix Command	*System Call*	*Attrib Modified*
chmod	chmod	access, last time chgd
chown	chown	user id, group id, last time chgd

Commands&Functions Affecting Attributes

Unix Command	*System Call*	*Attrib Modified*
chmod	chmod	access, last time chgd
chown	chown	user id, group id, last time chgd
chgrp	chown	group id, last time chgd

Commands&Functions Affecting Attributes

Unix Command	*System Call*	*Attrib Modified*
chmod	chmod	access, last time chgd
chown	chown	user id, group id, last time chgd
chgrp	chown	group id, last time chgd
touch	utime	last access, last time chgd

Commands&Functions Affecting Attributes

Unix Command	*System Call*	*Attrib Modified*
chmod	chmod	access, last time chgd
chown	chown	user id, group id, last time chgd
chgrp	chown	group id, last time chgd
touch	utime	last access, last time chgd
ln	link	increments hard link counter

Commands&Functions Affecting Attributes

Unix Command	*System Call*	*Attrib Modified*
chmod	chmod	access, last time chgd
chown	chown	user id, group id, last time chgd
chgrp	chown	group id, last time chgd
touch	utime	last access, last time chgd
ln	link	increments hard link counter
rm	unlink	decrements hard link counter

Commands&Functions Affecting Attributes

Unix Command	*System Call*	*Attrib Modified*
chmod	chmod	access, last time chgd
chown	chown	user id, group id, last time chgd
chgrp	chown	group id, last time chgd
touch	utime	last access, last time chgd
ln	link	increments hard link counter
rm	unlink	decrements hard link counter
(many)	-	file contents

Commands&Functions Affecting Attributes

Unix Command	*System Call*	*Attrib Modified*
chmod	chmod	access, last time chgd
chown	chown	user id, group id, last time chgd
chgrp	chown	group id, last time chgd
touch	utime	last access, last time chgd
ln	link	increments hard link counter
rm	unlink	decrements hard link counter
(many)	-	file contents

Unix Commands are issued via a shell

Commands&Functions Affecting Attributes

Unix Command	*System Call*	*Attrib Modified*
chmod	chmod	access, last time chgd
chown	chown	user id, group id, last time chgd
chgrp	chown	group id, last time chgd
touch	utime	last access, last time chgd
ln	link	increments hard link counter
rm	unlink	decrements hard link counter
(many)	-	file contents

Unix Commands are issued via a shell

System Calls functions invoked internally by some program

Unix Kernel Support for Opening Files

1. Kernel has an open-file table (user file descriptor table)

Unix Kernel Support for Opening Files

1. Kernel has an open-file table (user file descriptor table)
2. Kernel has an i-node table (permissions, times, data block indexing)

Unix Kernel Support for Opening Files

1. Kernel has an open-file table (user file descriptor table)
2. Kernel has an i-node table (permissions, times, data block indexing)
3. Kernel also creates and manages processes.

   Processes have considerable amounts of associated information: memory, file descriptors, shared memory, environment variables, etc. In particular, processes have a *file descriptor table*.

Opening Files, con't.

4. Upon a request to open a file:

Opening Files, con’t.

4. Upon a request to open a file:

   (a) Kernel searches the processes’ file descriptor table for an unused slot. Index to slot returned to the process (ie. the integer file descriptor).

Opening Files, con't.

4. Upon a request to open a file:
    (a) Kernel searches the processes' file descriptor table for an unused slot. Index to slot returned to the process (ie. the integer file descriptor).
    (b) Kernel examines its own file table for an unused slot to be used to reference the file. If found,

Opening Files, con't.

4. Upon a request to open a file:
    (a) Kernel searches the processes' file descriptor table for an unused slot. Index to slot returned to the process (ie. the integer file descriptor).
    (b) Kernel examines its own file table for an unused slot to be used to reference the file. If found,
        i. Process file descriptor slot points to kernel file table slot

Opening Files, con't.

4. Upon a request to open a file:
   - (a) Kernel searches the processes' file descriptor table for an unused slot. Index to slot returned to the process (ie. the integer file descriptor).
   - (b) Kernel examines its own file table for an unused slot to be used to reference the file. If found,
     - i. Process file descriptor slot points to kernel file table slot
     - ii. File table entry points to inode table where inode record is stored

Opening Files, con't.

4. Upon a request to open a file:
   (a) Kernel searches the processes' file descriptor table for an unused slot. Index to slot returned to the process (ie. the integer file descriptor).
   (b) Kernel examines its own file table for an unused slot to be used to reference the file. If found,
      i. Process file descriptor slot points to kernel file table slot
      ii. File table entry points to inode table where inode record is stored
      iii. File table entry gets current file pointer of open file; ie. a count of bytes from the beginning of the file (see ftell).

Opening Files, con’t.

4. con’t.
   (c) con’t.
      iv. File table entry gets type of open (ie. read, write, both)

Opening Files, con’t.

4. con’t.
   (c) con’t.
      iv. File table entry gets type of open (ie. read, write, both)
      v. Reference count in file table entry set to 1.
         Counts qty of file descriptors from any process that are accessing the file.
         (see dup). Child processes are often given duplicate file access, for example.

Opening Files, con't.

4. con't.

   (c) con't.

      iv. File table entry gets type of open (ie. read, write, both)

      v. Reference count in file table entry set to 1.
      Counts qty of file descriptors from any process that are accessing the file.
      (see `dup`). Child processes are often given duplicate file access, for example.

      vi. Reference count of the inode is incremented (ie. how many file table entries access the inode; see `ln`)

Opening Files, con't.

4. con't.
   (c) con't.
      iv. File table entry gets type of open (ie. read, write, both)
      v. Reference count in file table entry set to 1.
         Counts qty of file descriptors from any process that are accessing the file. (see `dup`). Child processes are often given duplicate file access, for example.
      vi. Reference count of the inode is incremented (ie. how many file table entries access the inode; see `ln`)

Once open() succeeds, one may read(), write(), and close() the file.

File descriptors index the file descriptor table → the kernel's file table.

Unix Kernel Support for Closing Files

Upon closing a file:

1. Kernel sets (open) file descriptor slot to unused

Unix Kernel Support for Closing Files

Upon closing a file:

1. Kernel sets (open) file descriptor slot to unused
2. Decrements process file table entry by 1. If non-zero, returns success (0)

Unix Kernel Support for Closing Files

Upon closing a file:

1. Kernel sets (open) file descriptor slot to unused
2. Decrements process file table entry by 1. If non-zero, returns success (0)
3. Process' file table entry marked unused

Unix Kernel Support for Closing Files

Upon closing a file:

1. Kernel sets (open) file descriptor slot to unused
2. Decrements process file table entry by 1. If non-zero, returns success (0)
3. Process' file table entry marked unused
4. Reference count in file inode table decremented. If non-zero, returns success (0)

Unix Kernel Support for Closing Files

Upon closing a file:

1. Kernel sets (open) file descriptor slot to unused
2. Decrements process file table entry by 1. If non-zero, returns success (0)
3. Process' file table entry marked unused
4. Reference count in file inode table decremented. If non-zero, returns success (0)
5. If hard link count of inode $\neq 0$, returns success (0)

Unix Kernel Support for Closing Files

Upon closing a file:

1. Kernel sets (open) file descriptor slot to unused
2. Decrements process file table entry by 1. If non-zero, returns success (0)
3. Process' file table entry marked unused
4. Reference count in file inode table decremented. If non-zero, returns success (0)
5. If hard link count of inode $\neq 0$, returns success (0)
6. Mark inode table entry as unused, de-allocate physical disk storage.

Unix Kernel Support for Closing Files

Upon closing a file:

1. Kernel sets (open) file descriptor slot to unused
2. Decrements process file table entry by 1. If non-zero, returns success (0)
3. Process' file table entry marked unused
4. Reference count in file inode table decremented. If non-zero, returns success (0)
5. If hard link count of inode $\neq 0$, returns success (0)
6. Mark inode table entry as unused, de-allocate physical disk storage.
7. Returns success (0)