File System Implementation

File System Implementation

- The file system is implemented using files, directories and the files that are currently opened.
- Each one will be implemented as a data structure and a set of procedures to manipulate the data structures

The open file table contains a structure for each open file. An open file structure contains the following information

- Current file position
- Other status information about the open file (eg. Whether the file is locked or not)
- Pointer to the file descriptor that is open

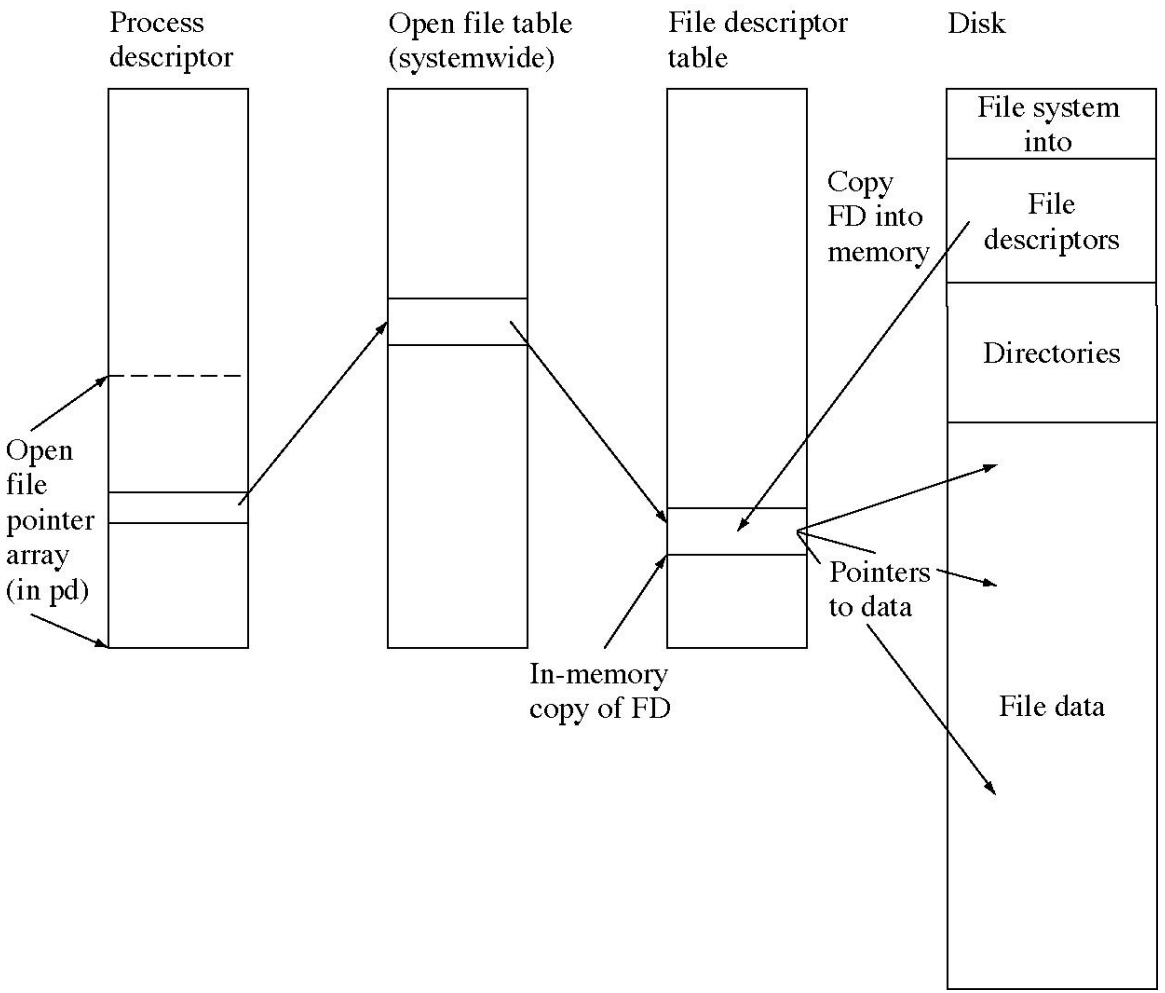
Most open files are connected to files. Files exist on disk and consist of two parts

- File descriptor : contains all the meta data about the file
- File data: Kept in disk blocks

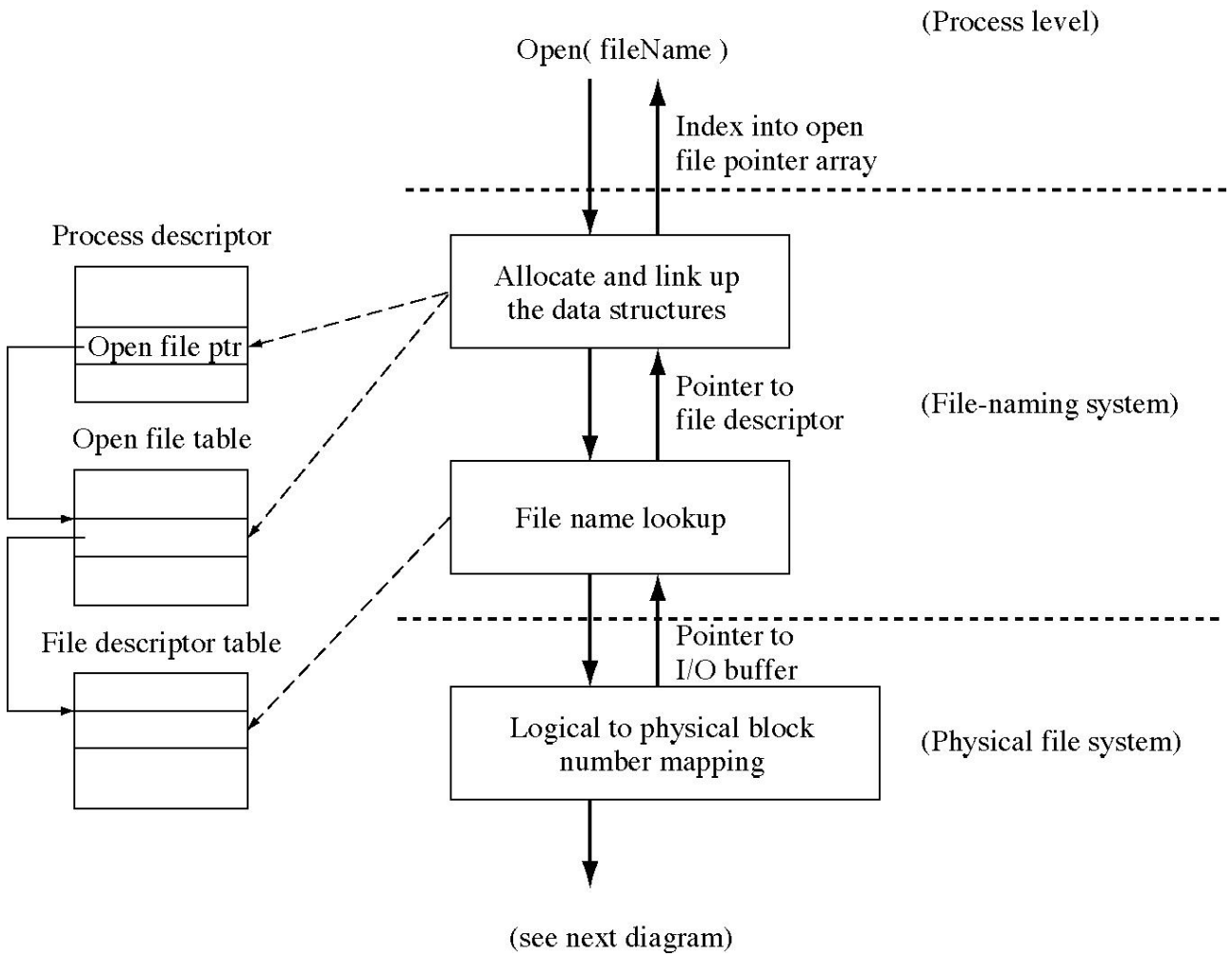
File Descriptor data structure contains the following information

- Owner of the file
- File protection information
- Time of creation, last modification and last use
- Other file meta data
- Location of the file data

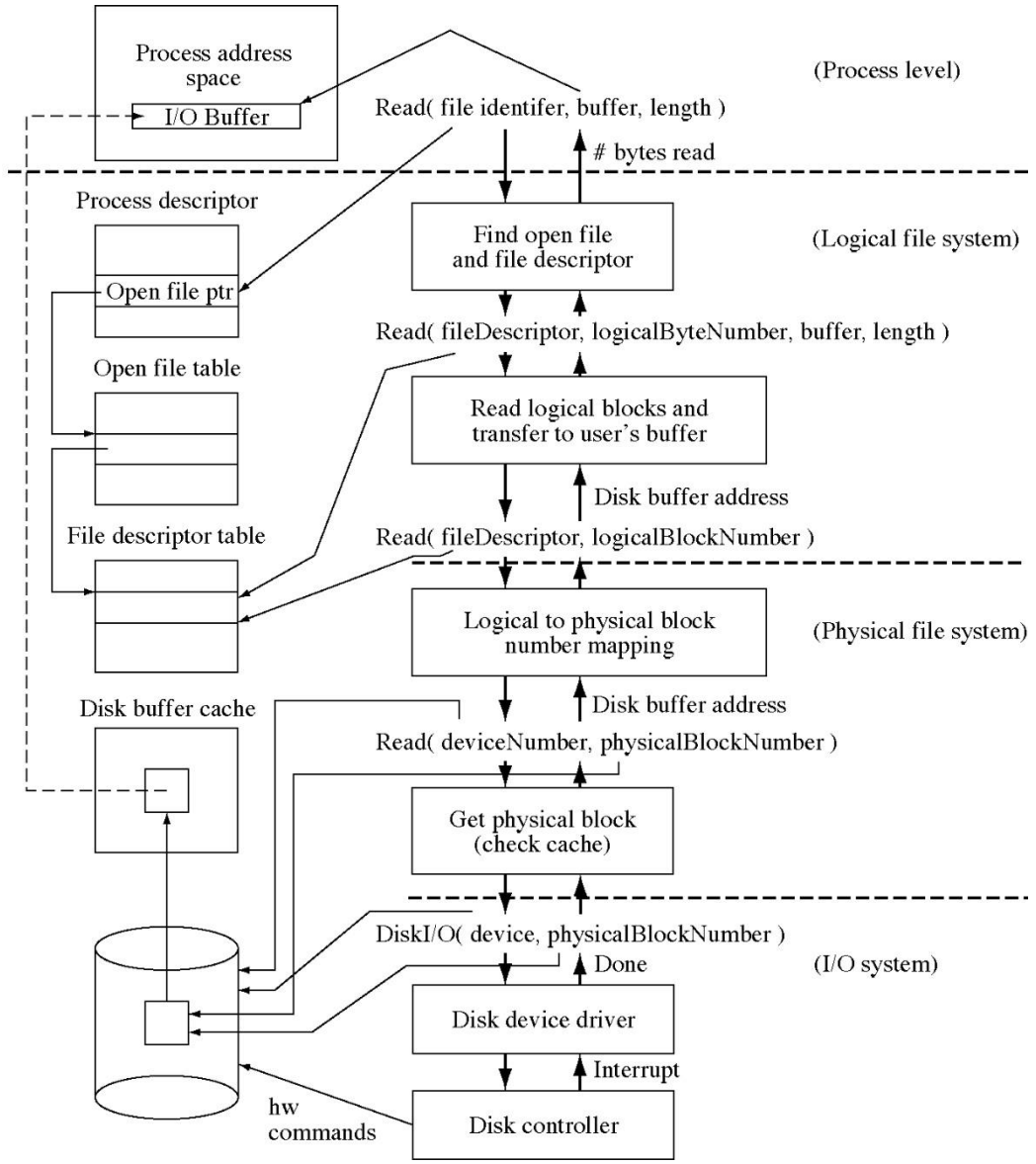
File system data structures



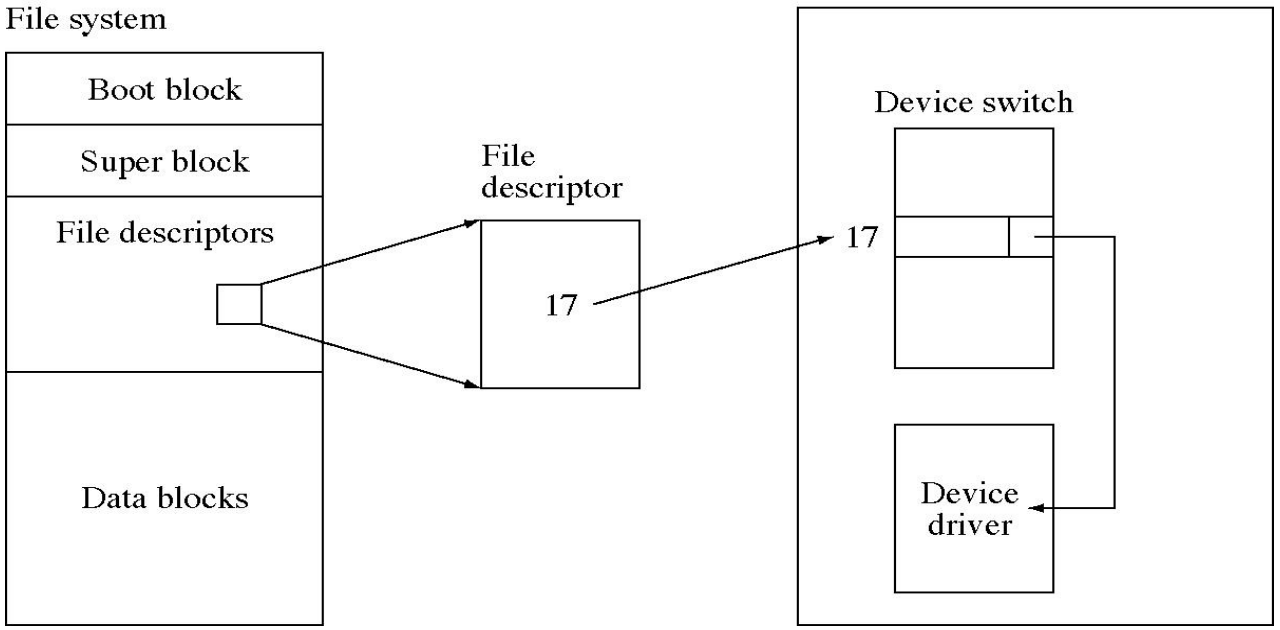
Flow of control for an open



Flow of control for a read



Connecting files and devices



Directory Implementation

A directory is a table that maps component names to file descriptors. The steps involved in a path lookup are

1) Let FD be the root if the path starts with '/' or FD be current directory and start at the beginning of the path name.

2) If the path name is at the end then return FD.

3) Isolate the next component in the path name and let it be C. Move past the component in the path name.

4) if FD is not a directory, then return an error.

5) Search through the directory FD for the component name C. This involves a loop that reads the next name/fd pair and compares the name with C. Loop until a match is found or the end of the directory is reached.

6) If no match is found, then return an error.

7) If a match was found, then its associated file descriptor becomes the new FD. Go back to step2.

Free Space Management

Space Allocation

Files are allocated disk spaces by operating system. Operating systems deploy following three main ways to allocate disk space to files.

- Contiguous Allocation
- Linked Allocation
- Indexed Allocation

Contiguous Allocation

- Each file occupies a contiguous address space on disk.
- Assigned disk address is in linear order.
- Easy to implement.
- The difficulty with contiguous allocation is finding space for a new file. If the file to be created is n blocks long, then the OS must search for n free contiguous blocks.
- First-fit, best-fit, and worst-fit strategies are the most common strategies used to select a free hole from the set of available holes.
- External fragmentation is a major issue with this type of allocation technique.
- Another problem with contiguous allocation is determining how much disk space is needed for a file.

Linked Allocation

- Each file carries a list of links to disk blocks.
- Directory contains link / pointer to first block of a file.
- This pointer is initialized to NIL (the end-of-list pointer value) to signify an empty file.
- A write to a file removes the first free block and writes to that block. This new block is then linked to the end of the file.
- To read a file, the pointers are just followed from block to block.
- No external fragmentation
- Effectively used in sequential access file.
- Inefficient in case of direct access file.

Indexed Allocation

- Provides solutions to problems of contiguous and linked allocation.
- A index block is created having all pointers to files.
- Each file has its own index block which stores the addresses of disk space occupied by the file which is an array of disk sector of addresses. The i^{th} entry in the index block points to the i^{th} sector of the file.
- Directory contains the addresses of index blocks of files.
- To read the i^{th} sector of the file, the pointer in the i^{th} index block entry is read to find the desired sector.
- Indexed allocation supports direct access, without suffering from external fragmentation. Any free block anywhere on the disk may satisfy a request for more space.

Free Space List

- Since there is only a limited amount of disk space, it is necessary to reuse the space from deleted files for new files.
- To keep track of free disk space, the system maintains a free-space list.
- The free-space list records all disk blocks that are free (i.e., are not allocated to some file).
- To create a file, the free-space list has to be searched for the required amount of space, and allocate that space to a new file. This space is then removed from the free-space list.
- When a file is deleted, its disk space is added to the free-space list.

Free Space List

Bit-Vector

- Frequently, the free-space list is implemented as a bit map or bit vector. Each block is represented by a 1 bit. If the block is free, the bit is 0; if the block is allocated, the bit is 1.
- For example, consider a disk where blocks 2, 3, 4, 5, 8, 9, 10, 11, 12, 13, 17, 18, 25, 26, and 27 are free, and the rest of the blocks are allocated. The free-space bit map would be: 11000011000000111001111110001111…
- The main advantage of this approach is that it is relatively simple and efficient to find ‘n’ consecutive free blocks on the disk.
- Unfortunately, bit vectors are inefficient unless the entire vector is kept in memory for most accesses.
- Keeping it main memory is possible for smaller disks such as on microcomputers, but not for larger ones.

Linked List

- In this approach all the free disk blocks are linked together, keeping a pointer to the first free block. This block contains a pointer to the next free disk block, and so on.
- In the previous example, a pointer could be kept to block 2, as the first free block. Block 2 would contain a pointer to block 3, which would point to block 4, which would point to block 5, which would point to block 8, and so on.
- This scheme is not efficient; to traverse the list, each block must be read, which requires substantial I/O time.

Grouping

- A modification of the free-list approach is to store the addresses of ‘n’ free blocks in the first free block.
- The first n-1 of these is actually free. The last one is the disk address of another block containing addresses of another n free blocks.
- The importance of this implementation is that addresses of a large number of free blocks can be found quickly.

Counting

- Another approach is to take advantage of the fact that, generally, several contiguous blocks may be allocated or freed simultaneously, particularly when contiguous allocation is used.
- Thus, rather than keeping a list of free disk addresses, the address of the first free block is kept and the number ’n’ of free contiguous blocks that follow the first block.
- Each entry in the free-space list then consists of a disk address and a count.