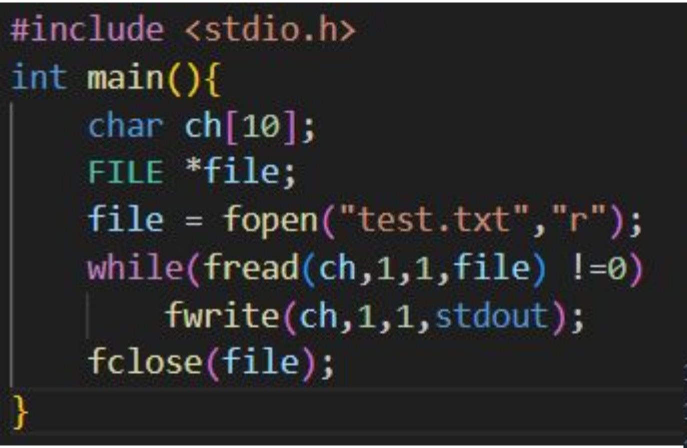
CSIT372 - Advanced Database Systems

Physical and Logical Files, Journey of a Byte

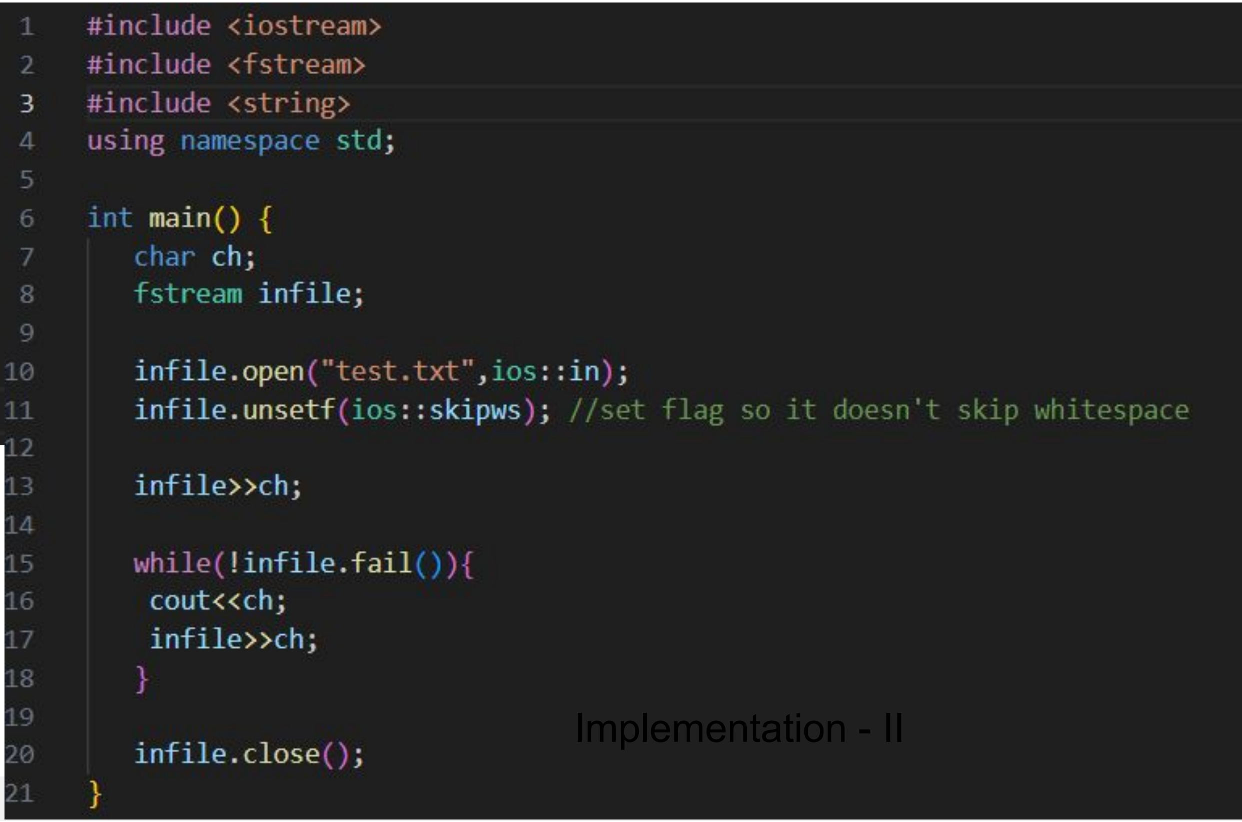
Dr. Prajish Prasad, FLAME University

Reading from a File



```
#include <stdio.h>
int main(){
    char ch[10];
    FILE *file;
    file = fopen("test.txt","r");
    while(fread(ch,1,1,file) !=0)
        fwrite(ch,1,1,stdout);
    fclose(file);
}
```

Implementation - I



```
#include <iostream>
#include <fstream>
#include <string>
using namespace std;

int main() {
    char ch;
    fstream infile;

    infile.open("test.txt",ios::in);
    infile.unsetf(ios::skipws); //set flag so it doesn't skip whitespace

    infile>>ch;

    while(!infile.fail()){
     cout<<ch;
     infile>>ch;
    }

    infile.close();
}
```

Implementation - II

What are similarities/differences?

Physical and Logical Files

- Physical File: A collection of bytes stored on a disk or tape
- Logical File: A "Channel" (like a telephone line) that hides the details of the file's location and physical format to the program

Physical and Logical Files

- When a program wants to use a particular file, “test.txt”, the operating system must find the physical file called “test.txt” and make the hookup by assigning a logical file to it.
- This logical file has a logical name which is what is used inside the program.

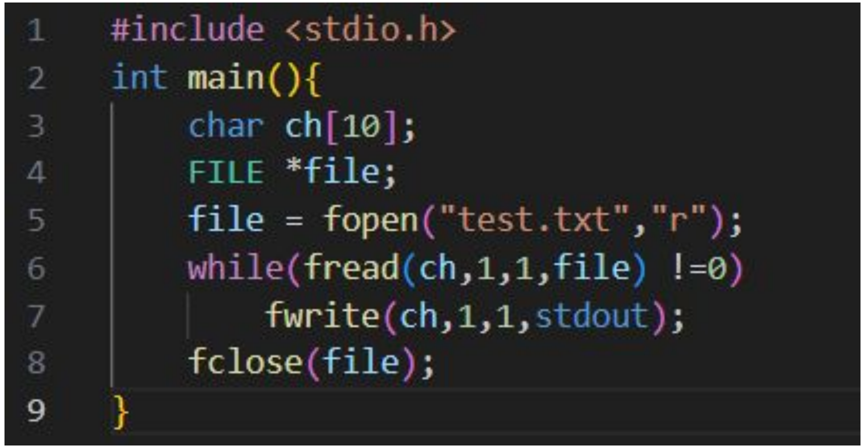
Material taken from - https://www.site.uottawa.ca/~nat/Courses/csci2111.html

Physical and Logical Files

- The program (application) send (or receives) bytes to (from) a file through the logical file. The program knows nothing about where the bytes go (came from)
- The operating system is responsible for associating a logical file in a program to a physical file
- Reading/Writing does through the OS

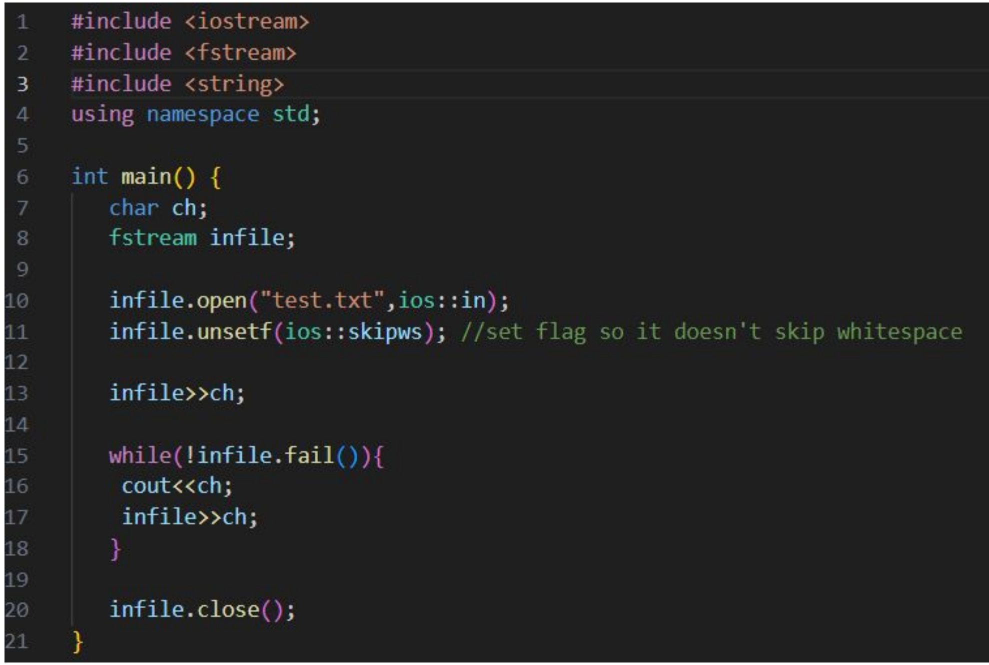
Material taken from - https://www.site.uottawa.ca/~lucia/courses/2131-05/CSI2131lect2005reduc.pdf

Write a program for writing your information onto a file character by character



```c
#include <stdio.h>
int main(){
    char ch[10];
    FILE *file;
    file = fopen("test.txt","r");
    while(fread(ch,1,1,file) !=0)
        fwrite(ch,1,1,stdout);
    fclose(file);
}
```

Implementation - I



```cpp
#include <iostream>
#include <fstream>
#include <string>
using namespace std;

int main() {
    char ch;
    fstream infile;

    infile.open("test.txt",ios::in);
    infile.unsetf(ios::skipws); //set flag so it doesn't skip whitespace

    infile>>ch;

    while(!infile.fail()){
     cout<<ch;
     infile>>ch;
    }

    infile.close();
}
```

Implementation - II

View #1:
File is seen as a stream of bytes

Journey of a Byte

What happens when the program statement:

write(textfile, ch, 1) is executed ?

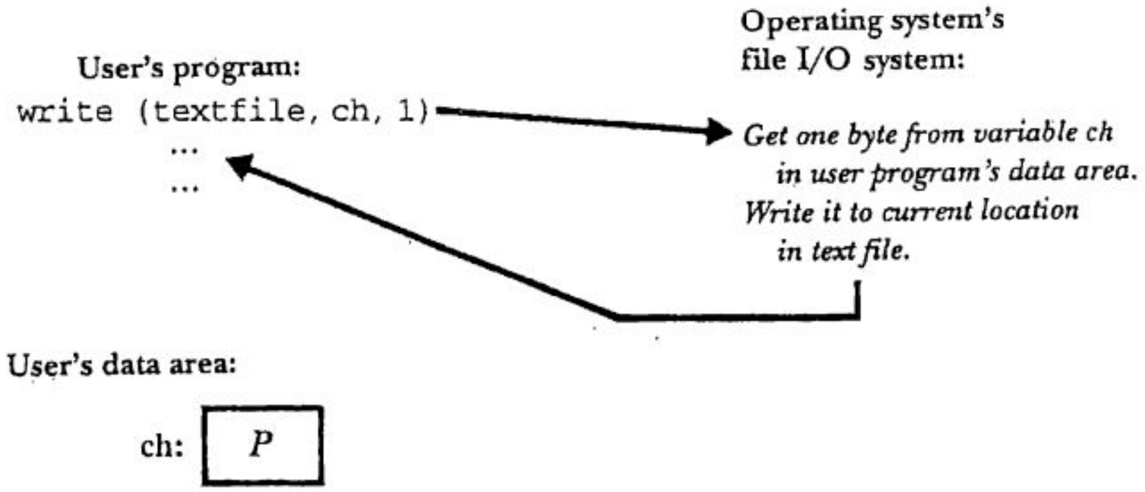
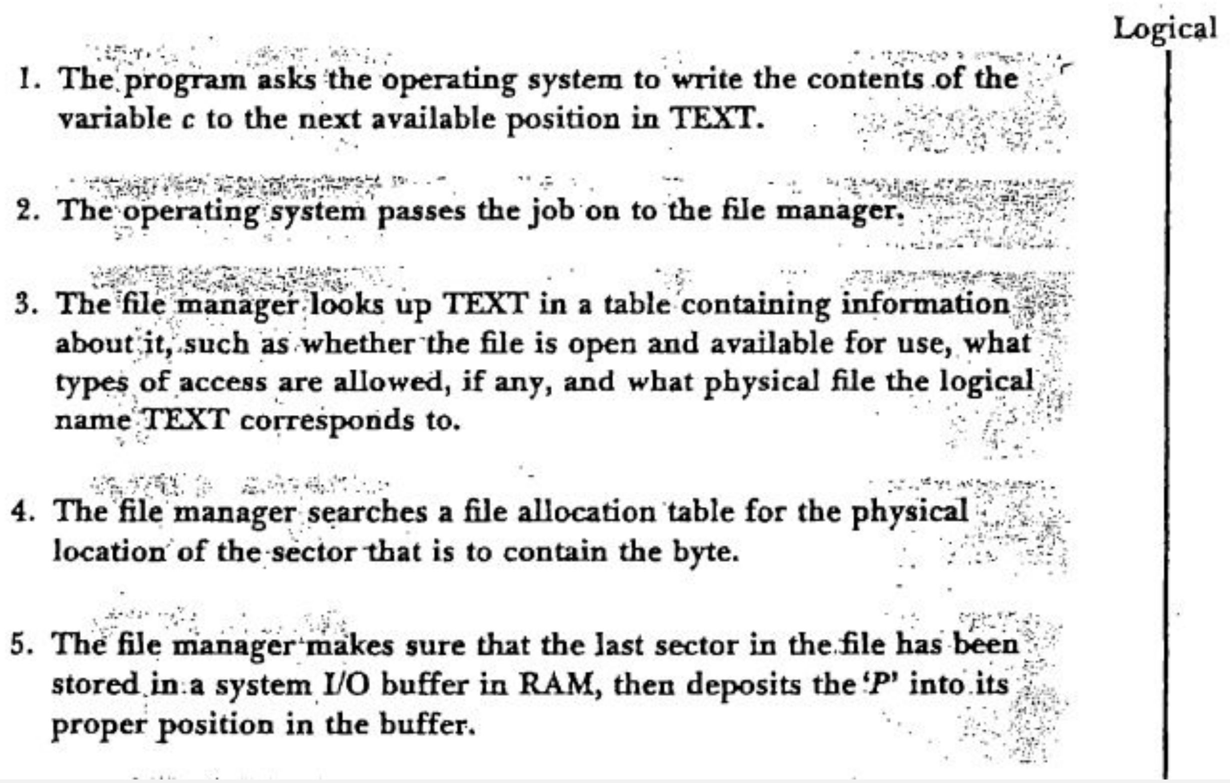

User's program:
write (textfile, ch, 1)
...
...
Operating system's file I/O system:
Get one byte from variable ch in user program's data area.
Write it to current location in text file.
User's data area:
ch:
P

Figure 3.18 The write statement tells the operating system to send one character to disk and gives the operating system the location of the character. The operating system takes over the job of writing, and then returns control to the calling program.



Logical

1. The program asks the operating system to write the contents of the variable *c* to the next available position in TEXT.

2. The operating system passes the job on to the file manager.

3. The file manager looks up TEXT in a table containing information about it, such as whether the file is open and available for use, what types of access are allowed, if any, and what physical file the logical name TEXT corresponds to.

4. The file manager searches a file allocation table for the physical location of the sector that is to contain the byte.

5. The file manager makes sure that the last sector in the file has been stored in a system I/O buffer in RAM, then deposits the '*P*' into its proper position in the buffer.

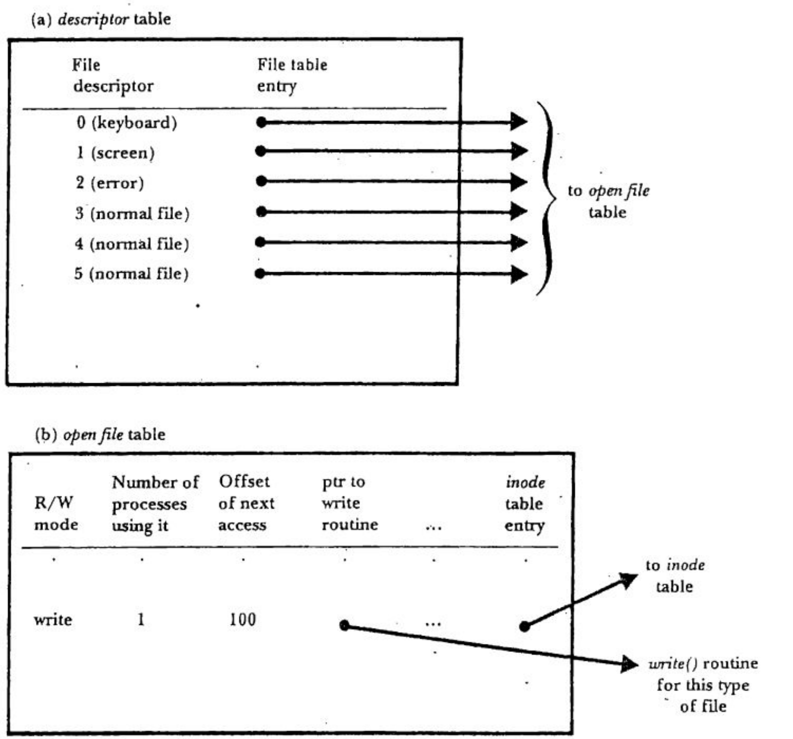


Figure 3.24 Descriptor table and open file table.

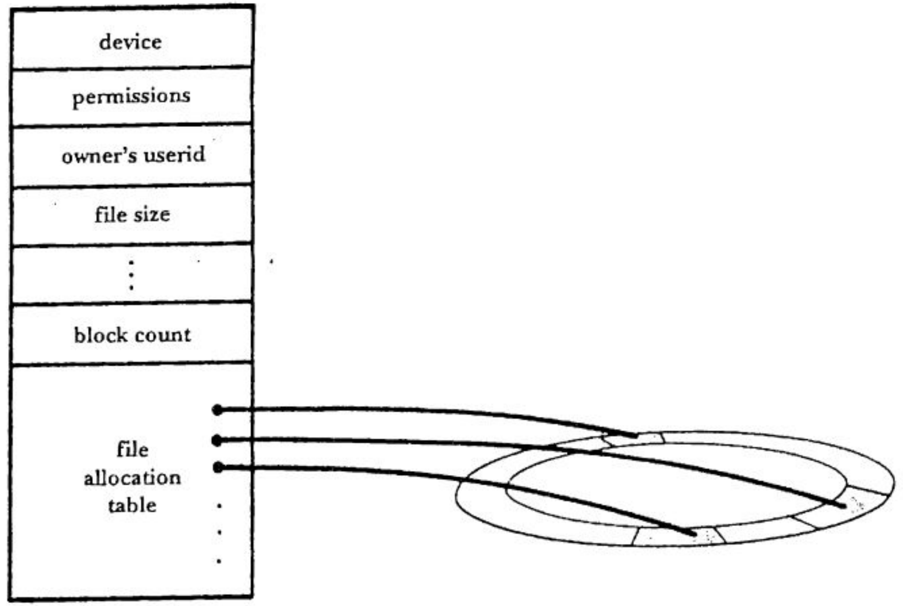
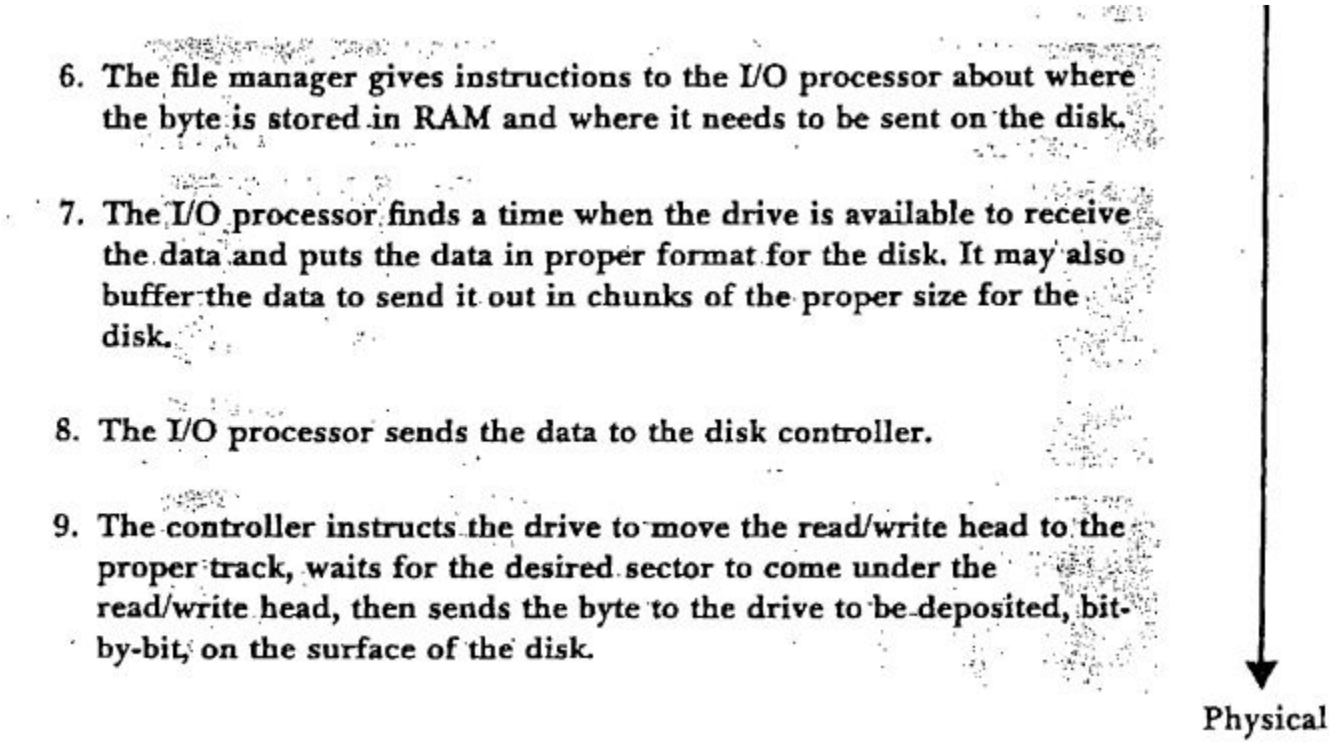


Figure 3.25 An inode. The inode is the data structure used by Unix to describe the file. It includes the device containing the file, permissions, owner and group IDs, and file allocation table, among other things.



6. The file manager gives instructions to the I/O processor about where the byte is stored in RAM and where it needs to be sent on the disk.

7. The I/O processor finds a time when the drive is available to receive the data and puts the data in proper format for the disk. It may also buffer the data to send it out in chunks of the proper size for the disk.

8. The I/O processor sends the data to the disk controller.

9. The controller instructs the drive to move the read/write head to the proper track, waits for the desired sector to come under the read/write head, then sends the byte to the drive to be deposited, bit-by-bit, on the surface of the disk.

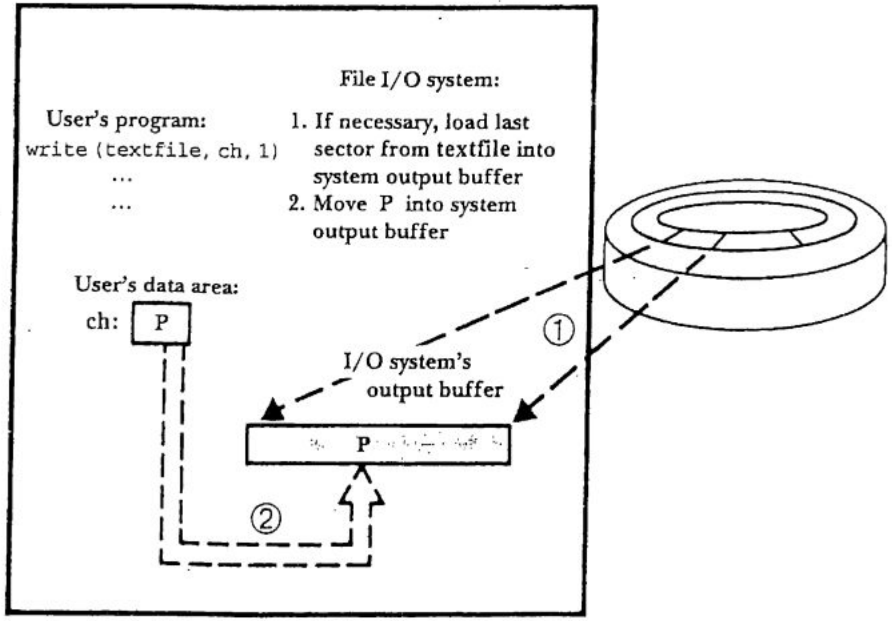

File I/O system:
User's program:
write (textfile, ch, 1)
...
...
1. If necessary, load last sector from textfile into system output buffer
2. Move P into system output buffer
User's data area:
ch: P
①
I/O system's output buffer
P
②

Figure 3.20 The file manager moves *P* from the program's data area to a system output buffer where it may join other bytes headed for the same place on the disk. If necessary, the file manager may have to load the corresponding sector from the disk into the system output buffer.

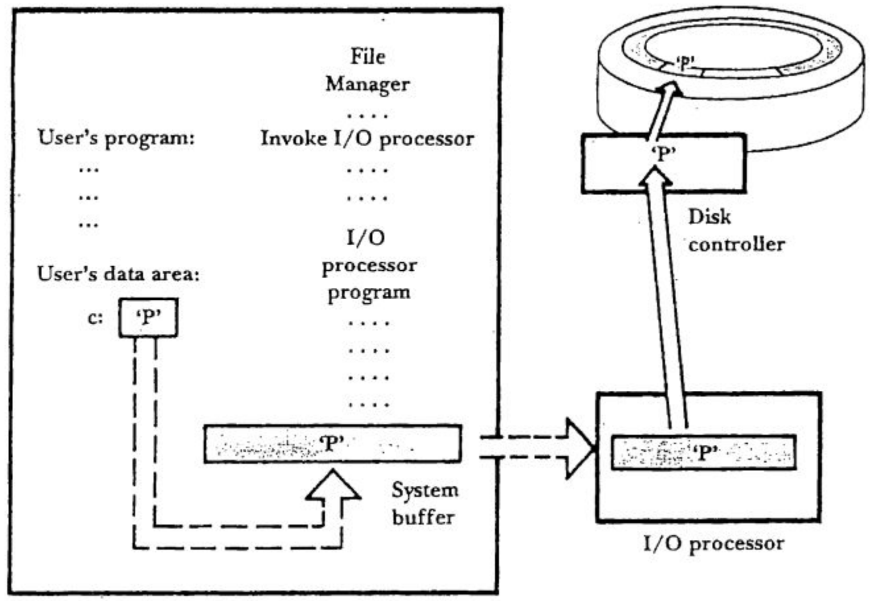


Figure 3.21 The file manager sends the I/O processor instructions in the form of an I/O processor program. The I/O processor gets the data from the system buffer, prepares it for storing on the disk, then sends it to the disk controller, which deposits it on the surface of the disk.

View #1:
File is seen as a stream of bytes

View #2:
File is seen as a collection of records with fields

Field and Record Organization

- Record: A collection of related fields
- Field: Smallest logically meaningful unit of information in a file
- Key: Subset of fields in a record used to (uniquely) identify the record

Field Structures: Think Individually

What are ways of organizing fields?

You have 100 students with the following attributes:

1. Roll No
2. First Name
3. Last Name
4. Age
5. Major

Share:

Fixed-Length Fields

E.g.

Roll No - 5 characters

First Name - 10 characters

Last Name - 10 characters

Age - 3 characters

Major - 15 characters

Field Beginning with Length Indicator

051234505Harry06Potter022115Computer Science

Delimiter at end of fields

12345|Harry|Potter|21|Computer Science|

Store field as keyword = value

RollNo=12345|FirstName=Harry|LastName=Potter|Age=21|Major=Computer Science|

Advantages/Disadvantages?

Types	Advantages	Disadvantages
Fixed		
Length indicator		
Delimited Fields		
Keyword		

Advantages/Disadvantages

Types	Advantages	Disadvantages
Fixed	Easy to Read/Store	Waste space with padding
Length indicator	Easy to jump ahead to the end of the field	Long fields require more than 1 byte to store length
Delimited Fields	May waste less space than with length-based	Have to check every byte of field against the delimiter
Keyword	Fields are self describing, allows for missing fields	Waste space with keywords

Record Structures

1. Fixed-length records
   a. Fixed-length fields
   b. Variable-length fields

2. Records with fixed number of fields

3. Records beginning with length indicator

4. Use an index to keep track of addresses

5. Delimiter at the end of the record