Paper 102: Programming & Problem solving through C

Lecture-22:Unit-III
Files-III

Aiusha V.Hujon, Dept. of Computer Science, SAC

Low level File operations

- In high level disk i/o, data transfers are first stored in a buffer and then this buffer is written into the disk when it is full.
  - These are done automatically
- In low level the buffer must be set for the data, place the appropriate values in it before writing, and take them out after writing.
- Thus, the buffer in the low level I/O functions is very much a part of the program, rather than being invisible as in high level disk I/O functions

Advantages of Low level disk I/O functions

- Since these functions are similar to the methods that the OS uses to write to the disk, they are more efficient than the high level disk I/O functions.
  - Since there are fewer layers of routines to go through
  - low level I/O functions operate faster than their high level counterparts.

Declaring the Buffer

```
char buffer[512];
```

- This is the buffer in which the data read from the disk will be placed.
- The size of this buffer is important for efficient operation.
- Depending on the operating system, buffers of certain sizes are handled more efficiently than others.

Opening a File

- As in high level disk I/O, the file must be opened before we can access it.
- This is done using the statement
  - inhandle = open (source, modes);

O-flags

- In low level file operation use file handles
  - O_APPEND - Opens a file for appending
  - O_CREAT - Creates a new file for writing (has no effect if file already exists)
  - O_RDONLY - Creates a new file for reading only
  - O_RDWR - Creates a file for both reading and writing
  - O_WRONLY - Creates a file for writing only
  - O_BINARY - Creates a file in binary mode
  - O_TEXT - Creates a file in text mode
- These 'O-flags' are defined in the file "fcntl.h", this file must be included in the program while using low level disk I/O.
- When two or more O-flags are used together, they are combined using the bitwise OR operator (|)

Opening a File

- The other statement used in our program to open the file is,
- outhandle = open (target, O_CREAT | O_BINARY | O_WRONLY, S_IWRITE) ;
  - Note that since the target file is not existing when it is being opened we have used the O_CREAT flag, and since we want to write to the file and not read from it, therefore we have used O_WRONLY. And finally, since we want to open the file in binary mode we have used O_BINARY.
- Whenever O_CREAT flag is used, another argument must be added to open( ) function to indicate the read/write status of the file to be created.
  - This argument is called 'permission argument'.

Opening a File

- Permission arguments could be any of the following:
  - S_IWRITE - Writing to the file permitted
  - S_IREAD - Reading from the file permitted
- To use these permissions, both the files "types.h" and "stat.h" must be #included in the program along with "fcntl.h".

File Handles

- Instead of returning a FILE pointer as fopen( ) did, in low level disk I/O, open( ) returns an integer value called 'file handle'.
- This is a number assigned to a particular file, which is used thereafter to refer to the file.
- If open( ) returns a value of -1, it means that the file couldn't be successfully opened.

Interaction between Buffer and File

- The following statement reads the file or as much of it as will fit into the buffer:
- bytes = read (inhandle, buffer, 512) ;
- The read( ) function takes three arguments.
  - The first argument is the file handle
  - The second is the address of the buffer and
  - The third is the maximum number of bytes we want to read.
- The read( ) function returns the number of bytes actually read.

Interaction between Buffer and File

- The following statement writes into the file
- write (outhandle, buffer, bytes) ;
- The write( ) function takes three arguments.
  - The first argument is the file handle
  - The second is the address of the buffer and
  - The third is the maximum number of bytes we want to write.
- The write( ) function returns the number of bytes actually written.

Example

```
/* File-copy program which copies text, .com and .exe files */
#include "fcntl.h"
#include "types.h" /* if present in sys directory use "c:tc\\include\\sys\\types.h" */
#include "stat.h" /* if present in sys directory use  "c:\\tc\\include\\sys\\stat.h" */
main ( int argc, char *argv[ ] )
{
char buffer[ 512 ], source [ 128 ], target [ 128 ] ;
int inhandle, outhandle, bytes ;
printf ( "\nEnter source file name" ) ;
gets ( source ) ;
inhandle = open ( source, O_RDONLY | O_BINARY ) ;
if ( inhandle == -1 )
{
puts ( "Cannot open file" ) ;
exit(1 ) ;
}
```

```
printf ( "\nEnter target file name" ) ;
gets ( target ) ;
outhandle = open ( target, O_CREAT | O_BINARY | O_WRONLY, S_IWRITE ) ;
if ( inhandle == -1 )
{
puts ( "Cannot open file" ) ;
close ( inhandle ) ;
exit( 1) ;
}
while ( 1 )
{
bytes = read ( inhandle, buffer, 512 ) ;
if ( bytes > 0 )
      write ( outhandle, buffer, bytes ) ;
else
      break ;
}
close ( inhandle ) ;
close ( outhandle ) ;
}
```

Another example

```
/*fwrite.c  program to illustrate buffered write*/
#include<io.h>
#include<stdio.h>
void main()
{
    FILE *fp;
    fp=fopen("testbuf.txt","wb");
    fwrite("1. This is fwrite\n",1,18,fp);
    write(fileno(fp),"2. This is write\n",17);
    fclose(fp);
}
```

Testbuf.txt

- 2. This is write
- 1. This is fwrite