Problems on Disk Drives

Given a Western Digital Caviar AC2850 Disk Drive with the following specifications :

Capacity = 850MB
Minimum seek time = 1 msec
Average seek time = 10 msec
Maximum seek time = 22 msec

Spindle speed = 4500 rpm
Average rotational delay = 6.6 msec
Maximum transfer rate = 13.3 msec/track or 2419 bytes/msec

bytes per sector = 512
tracks per cylinder = 16
sectors per track = 63
cylinders = 1654

1) Suppose that we want to store a file with 60,000 fixed-length data records where each record requires 80 bytes and records are not allowed to span two sectors - How many cylinders are required for this file ?

Answer :

- Each sector can hold $\lfloor 512/80 \rfloor = 6$ records
- The file requires $60{,}000/6 = 10{,}000$ sectors
- One cylinder can hold $63 \times 16 = 1008$ sectors
- So the number of cylinders required is approximately $10{,}000/1008 =$ **9.93 cylinders**.

2) How much internal fragmentation is caused by the fact that records are not allowed to span to sectors ?

Answer :

- Each sector contains $512 - 6 \times 80 = 32$ unused bytes
- Since the file requires 10,000 sectors, then $32 \times 10,000 =$ **320,000 bytes** are lost due to internal fragmentation

3) Assume that the file could be read sector by sector sequentially. How long would it take to read the entire 60,000 records in the worst case ?

Answer :

- When read sequentially, a track would be read in, at most,
  Maximum seek time + Maximum rotational delay + Maximum transfer rate = 22 msec + (6.6 msec $\times 2$ (in the worst case the disk has to make a full rotation)) + 13.3 msec = 48.5 msec
- Since each cylinder contains 16 tracks, the file holds on 9.93 cylinders $\times 16$ tracks/cylinder = 158.88 tracks $\approx$ 159 tracks
- In the worst case, the file would, therefore, be read sequentially in 159×48.5 msec = 7711.5 msec = **7.72 seconds**

4) Assume that everytime a new sector is read, its access is random rather than sequential. How long would it take to read the entire file in the worst case ?

Answer :

- Each sector, accessed randomly, would be read in
  Maximum seek time + Maximum rotation delay + Maximum transfer rate/sector = 22 msec + (6.6 msec $\times 2$) + 512/2419 msec = 35.42 msec

  Note that the maximum transfer rate per sector was calculated as follow :
  2419 bytes $\longrightarrow$ 1 msec
  512 bytes (1 sector) $\longrightarrow 1 \times 512/2419$ msec
- Since the file requires 10,000 sectors, then in the worst case, the file would be read in
  $10,000 \times 35.42 = 345,200$ msec = 354.2 sec $\approx$ **6 minutes**

Note the huge difference between sequential and random access.

Problems on Tape Units

Given a 9-track tape of :

- density : 1600 bits per inch (bpi) per track
- speed : 150 inches per second (ips)
- interblock gap size : 0.5 inch

1) Suppose we want to store a file with fixed-length data records where each record requires 80 bytes (the same file as before). How much tape is needed if we use a blocking factor of 25 ?

Answer :

- The physical length of a data block is
  b = $(25 \times 80)/1600 = 2000/1600 = 1.25$ inch
- The number of data blocks required to hold the file is
  n = 60,000/25 = 2400
- The size of an interblock gap is 0.5 inch
- The space required for storing the file is :
  $s = n \times (b + g) = 2400 \times (1.25 + 0.5) =$
  4200 inches = 4200/12 feet = **350 feet**

2) What is the tape's effective recording density ?

Answer :

- number of bytes per block / number of inches required to store a block
  $= 25\times$ 80 /1.75 = **1142.9 bytes/inch**

3) What is the tape's effective transmission rate ?

Answer :

- effective recording rate $\times$ tape speed
  $= 1142.9 \times 150 = 171,435$ bytes/sec $\approx$ **171.44 Kbytes/sec**

C++

Operator Overloading

Definition : Operator overloading is the syntactic possibility C++ offers to redefine the actions of an operator in a given context.

Why to use overloading ?

C++ operators do not, generally, apply to class objects : they only apply to pre-defined types. For example, the equality operator "==" does not accept strings of characters as operands - This means that instead of having the possibility to write :
`if (a==b)` ... (a and b are strings)
one needs to write more complicated expressions such a
`if (strcmp(a.string,b.string) == 0)` ... or
`if (compstr(a,b) == 0)` ...

Operator overloading, however, allows "==" to be redefined for strings and, expressions such as `if (a==b)` to be written.

Example

Let `"rat"` be the class of rational numbers, i.e., fractions with numerator and denominator, both of which being integer.

```
class rat
{
   int z;          // numerator
   int n;          // denominator
   public :
   rat(int zr=0,int nr=1): z(zr),n(nr) {}   // constructor

   void show()
   {
      if ((long)z*n<0)  // negative fraction
         cout << "-";   // prints the - sign
      if (z*n == 0)     // the fraction has value 0
         cout << 0;
```

```
      else               // prints the fraction, otherwise
         cout << (z>0 ? z:-z) << "/" << (n>0 ? n:-n);
   }
};
```

Notes

1. In the constructor ":z(zr),n(nr)" corresponds to the initialization list, equivalent to

   ```
   {
     z = zr;
     n = nr;
   }
   ```

   in the body of the constructor

2. `cout << (z>0 ? z:-z)  <==>`

   ```
   if (z>0)
      cout << z;
   else
      cout << -z;
   ```

Overloading the ++ operator

```
rat rat::operator++()   // prefix version
{
   z += n;              // adds 1 to z/n
   return(*this);        // returns the copy of
}                       //    the modified object


rat rat::operator++(int)   // suffix version
{
   rat tmp=*this;          // saves the old value
                           //    of the object
   z += n;                 // adds 1 to z/n
```

```
   return(tmp);             // returns the copy of the
}                           //    non-modified object
```

Usage

```
rat a1(3,5), a2(3,5), b;

b = ++a1;        // calls the prefix version of operator ++
                 // a1 is increased before being assigned
                 //    to b which receives value 8/5

b = a2++;        // calls the suffix version of operator ++
                 // b receives the value of object a2
                 //    before a2 is increased, i.e., b
                 //    receives value 3/5
```

Note : In the suffix version, the "int" formal parameter is present to indicate that "++" comes after the object rather than before - that is its only function.

Example

```
   cout << "a\tb\n";
   b = ++a1;
   a1.show();
   cout << "\t";
   b.show();
   cout << "\n";

   b = a2++;
   a2.show();
   cout << "\t";
   b.show();
```

will print :

```
  a     b
 8/5   8/5
 8/5   3/5
```

Overloading of the "put to" operator <<

Let the class `"Person"` be defined in `person.h` as :

```
class Person
{
   public :
     // constructors and destructor
     Person();
     Person(char const *n, char const *a, char const *p);
     ~Person();

     // interface functions
     void setname(char const *n);
     void setaddress(char const *a);
     void setphone(char const *p);
     char const *getname(void) const;
     char const *getaddress(void) const;
     char const *getphone(void) const;

   private :
     // data fields
     char *name;
     char *address;
     char *phone;
};
```

Note : The `getname`, `getaddress` and `getphone` functions are `const` **member functions** which is why the `const` keyword occurs after the parameter list - const member functions do not alter the data fields of its object, but only inspects them.

If we want to call the following code fragment,

```
   Person kr("Ken Ross","unknown",unknown");
   cout << "Name,address and phone number of person kr:\n" << kr << "\n");
```

Then we need to overload `<<` so that it can take `kr` as an argument.

The statement `cout << kr` invokes the operator `<<` and its two operands : an `ostream` & and a `Person` &. The proposed action is defined in a class-less operator function operator `<<()` expecting 2 arguments :

```
//declaration in, say, person.h
ostream &operator<<(ostream &, Person const &);

// definition in some source file
ostream &operator<<(ostream &stream, Person const &pers)
{
   return
   (
      stream << "Name:    " << pers.getname()
             << "Address: " << pers.getaddress()
             << "Phone:   " << pers.getphone()
   );
}
```
}