DHRUV GARG 16 BCE 1190

FILE ACCESS

LAB 9

WRITE A FILE AND NOTE THE TIME TAKEN

Step 1: Create a random file “dumpfile2” using the following command:

```
vmdhruv@ubuntu:~$ od -A n -t d -N 1000 /dev/urandom > dumpfile2
```

Step 2: Run “dd” command to write the files and see the times taken.

```
vmdhruv@ubuntu:~$ dd if=dumpfile2 of=speetest bs=1M count=100 conv=
fdatasync
0+1 records in
0+1 records out
3063 bytes (3.1 kB, 3.0 KiB) copied, 0.121736 s, 25.2 kB/s
```

Step 3: Repeat the same multiple times and note the time taken. Change the Block size and count size and notice the time. Check if it went through the cache.

```
vmdhruv@ubuntu:~$ dd if=dumpfile2 of=speetest bs=1M count=200 conv=
fdatasync
0+1 records in
0+1 records out
3063 bytes (3.1 kB, 3.0 KiB) copied, 0.0754932 s, 40.6 kB/s
vmdhruv@ubuntu:~$ dd if=dumpfile2 of=speetest bs=1M count=600 conv=
fdatasync
0+1 records in
0+1 records out
3063 bytes (3.1 kB, 3.0 KiB) copied, 0.0158075 s, 194 kB/s
vmdhruv@ubuntu:~$ dd if=dumpfile2 of=speetest bs=1M count=10000 con
v=fdatasync
0+1 records in
0+1 records out
3063 bytes (3.1 kB, 3.0 KiB) copied, 0.0960681 s, 31.9 kB/s
vmdhruv@ubuntu:~$ dd if=dumpfile2 of=speetest bs=1M count=20000 con
v=fdatasync
0+1 records in
0+1 records out
3063 bytes (3.1 kB, 3.0 KiB) copied, 0.0429687 s, 71.3 kB/s
vmdhruv@ubuntu:~$
```

ALLOCATING MEMORY TO A PROGRAM

CODE: allocate.c

```
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
int main(int argc, char** argv)
{
int max = -1; int mb = 0;  char* buffer;
if(argc > 1)      max = atoi(argv[1]);
while((buffer=malloc(1024*1024)) != NULL &&
mb != max)
{
memset(buffer, 0, 1024*1024); mb++;
printf("Allocated %d MB\n", mb);
}
return 0; }
```

SCREENSHOTS

```
vmdhruv@ubuntu:~$ gedit allocate.c &
[1] 2858
vmdhruv@ubuntu:~$ gcc -o allocate allocate.c
[1]+  Done                    gedit allocate.c
vmdhruv@ubuntu:~$ ./allocate
Allocated 1 MB
Allocated 2 MB
Allocated 3 MB
Allocated 4 MB
Allocated 5 MB
Allocated 6 MB
Allocated 7 MB
Allocated 8 MB
Allocated 9 MB
Allocated 10 MB
Allocated 11 MB
Allocated 12 MB
Allocated 13 MB
Allocated 14 MB
Allocated 15 MB
Allocated 16 MB
Allocated 17 MB
Allocated 18 MB
Allocated 19 MB
Allocated 20 MB
```

The program gets killed eventually, before it reaches its desired memory allocation.

```
Allocated 1140 MB
Allocated 1141 MB
Allocated 1142 MB
Allocated 1143 MB
Allocated 1144 MB
Allocated 1145 MB
Allocated 1146 MB
Allocated 1147 MB
Allocated 1148 MB
Allocated 1149 MB
Allocated 1150 MB
Allocated 1151 MB
Allocated 1152 MB
Allocated 1153 MB
Allocated 1154 MB
Allocated 1155 MB
Allocated 1156 MB
Allocated 1157 MB
Allocated 1158 MB
Allocated 1159 MB
Allocated 1160 MB
Allocated 1161 MB
Killed
vmdhruv@ubuntu:~$
```

DHRUV GARG 16 BCE 1190

TESTING THE TIME TAKEN FOR SEQUENTIAL FILE ACCESS AND RANDOM FILE ACCESS

CODE: randomSeqAccess.cpp

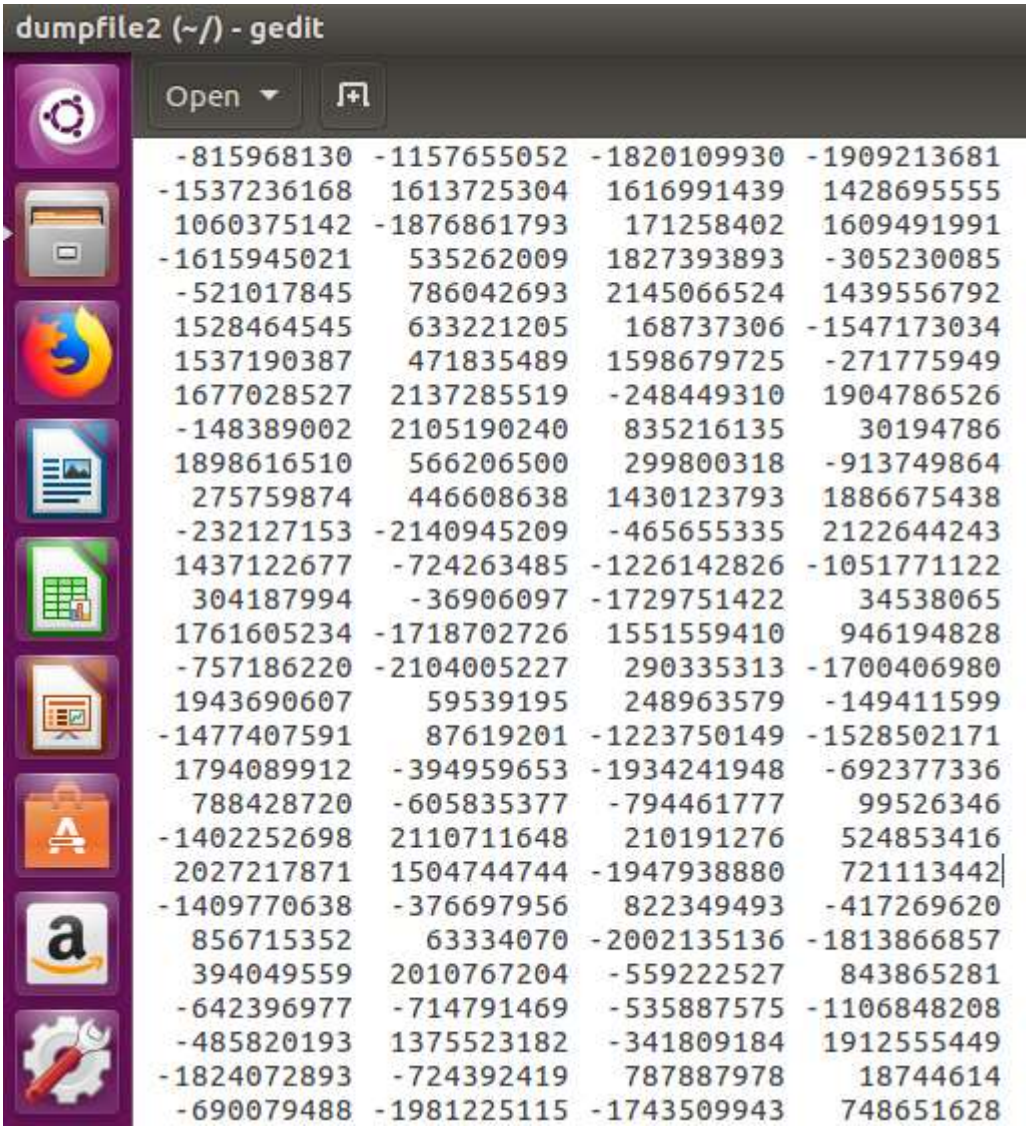
```
#include <iostream>
#include <fstream>
#include <time.h>
#include <stdlib.h>
#include <exception>
using namespace std;

void readseq()  // sequential read
{
   ifstream fin;
   fin.open("notes.txt");
   char temp;
   while( fin.get(temp))
      //cout << temp;
   //cout << endl;
   fin.close();
}
void randomseek() // random seek
{
   srand((double)(clock()));      // seed rand with
current time
   ifstream fin;
   fin.open("notes.txt");
   fin.seekg(0, ios::end);
   int size = fin.tellg();
   fin.clear();
   int *a;
   //cout<<size<<endl ;
   try
   {
      a = new int[size];
   }
   catch(bad_alloc& ba)
   {
      cerr << "Bad alloc caught: " << ba.what() << endl;
   }
   char temp;
   bool used = false, found =false;
   int pos = 0, n = 0;
   while(n < size-1)
   {
      found = false;
      do
      {
         used = false;
         pos = rand()%(size-1);
         for (int i = 0; i < n; ++i)
         {
            if(a[i] == pos)
            {
               used = true;
               break;
            }
         }
         if( !used )
         {
            found = true;
            a[n] = pos;
            n++;
         }
      }
      while( !found );
         fin.seekg(a[n-1], ios::beg);
      fin.get(temp);
      //cout <<temp;
   }
   //cout << endl;
   delete a;
   fin.close();
}
```

```
void timeit( void (*func)(void) )
{
   double time_taken ;
   cout << "Starting process...\n";
   clock_t timer = clock();
   (*func)();
   time_taken = (clock() - timer)/(double)CLOCKS_PER_SEC;
   cout << "Time taken: " << time_taken << endl;
}
```

```
int main()
{
   cout << "\n\nReading file => notes.txt (SEQUENTIAL ACCESS):\n";
   timeit(readseq);
   cout << "\n\nReading file => notes.txt (RANDOM ACCESS):\n";
   timeit(randomseek);
   return 0;
}
```

SCENARIO 1: Reading a file named "dumpfile2"



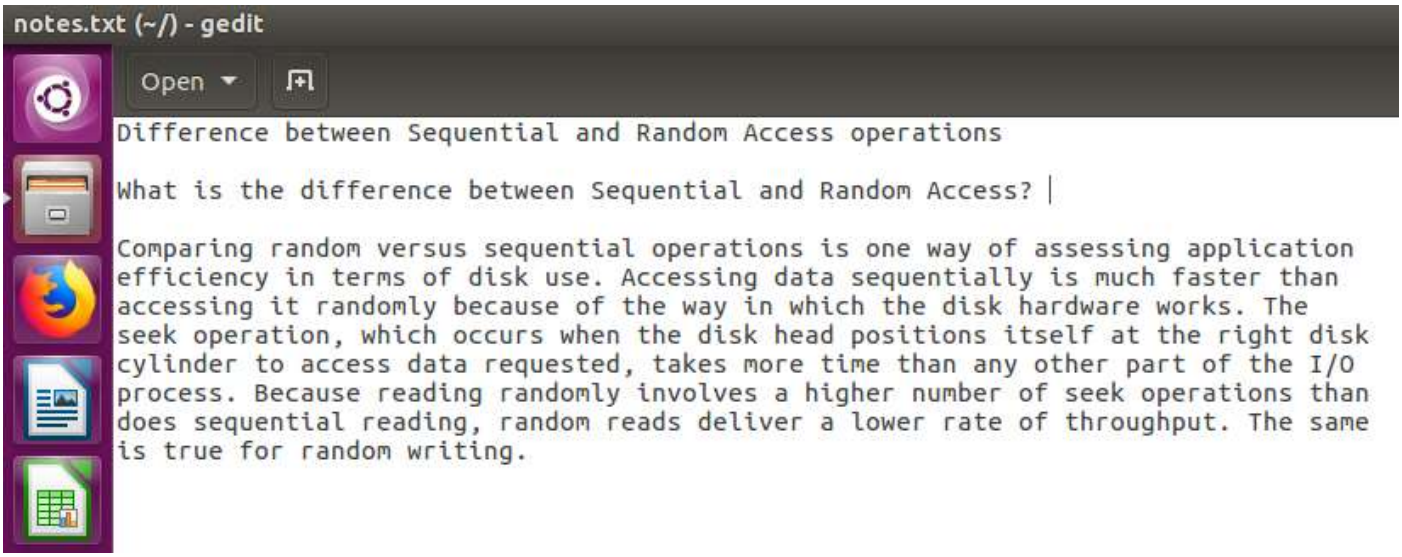
Checking the sequential and random access time

```
vmdhruv@ubuntu:~$ gedit randomSeqAccess.c &
[1] 3541
vmdhruv@ubuntu:~$ g++ -o randomSeqAccess randomSeqAccess.c
[1]+  Done                    gedit randomSeqAccess.c
vmdhruv@ubuntu:~$ ./randomSeqAccess


Reading file => dumpfile2 (SEQUENTIAL ACCESS):
Starting process...
Time taken: 5.7e-05


Reading file => dumpfile2 (RANDOM ACCESS):
Starting process...
Time taken: 0.201426
vmdhruv@ubuntu:~$
```

SCENARIO 2: **Reading a file named "notes.txt"**



Checking the sequential and random access time

```
vmdhruv@ubuntu:~$ gedit randomSeqAccess.c &
[1] 3575
vmdhruv@ubuntu:~$ g++ -o randomSeqAccess randomSeqAccess.c
[1]+  Done                    gedit randomSeqAccess.c
vmdhruv@ubuntu:~$ ./randomSeqAccess


Reading file => notes.txt (SEQUENTIAL ACCESS):
Starting process...
Time taken: 0.000116


Reading file => notes.txt (RANDOM ACCESS):
Starting process...
Time taken: 0.013575
vmdhruv@ubuntu:~$
```