CS380 Algorithm Design & Analysis

Assignment 2: Disk Scheduling

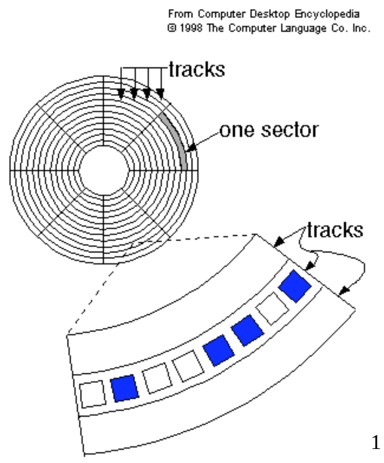
Date Assigned: Tuesday, February 17, 2009
Part 1 Due: Tuesday, February 24, 2009
Part 2 Due: Tuesday, March 3, 2009
Total Points: 40pts (15 points for part 1, 25 points for part 2)

Interactive time-shared computer systems often encounter a problem in scheduling I/O tasks for a magnetic disk. It is common to find such systems serving many terminals but having only one, or perhaps a few magnetic disk drives.

For example, such a system might service 200 terminals at any one time and have a single magnetic disk of several hundred gigabytes capacity. At any time the users could be running programs, compiling programs, and text editing where each of these activities could and do generate a mix of I/O requests.

The servicing of these requests can create a bottleneck that substantially affects the system performance. The I/O requests generated by all of the computing activities of the users are intermixed and at any time the disk can expect to have a group of I/O requests waiting for execution. Since disks can only handle one request at a time, the requests must be put into some order for service. The order chosen, called the disk-scheduling policy, is the subject of this assignment.

For this assignment, we will consider the disk to be a single platter or surface with a read/write head that can move radially across the surface of the disk. The disk surface is divided up into a set of concentric circles called tracks and each track is divided into segments called sectors.


From Computer Desktop Encyclopedia
© 1998 The Computer Language Co. Inc.
tracks
one sector
tracks

[1]

[1] http://dictionary.zdnet.com/definition/magnetic+disk.html

Our disk for this assignment has 200 tracks (0 - 199) and 20 sectors (0 - 19) per track. By the time an I/O request arrives from the user to the disk, it takes the following form:

```
r1 = (id#,R/W,Sr,Tr)

where  id#  - is the user's unique identification number (0 - 47)
       R/W  - is the request to read or write to the disk
       Sr   - is the sector number (0 - 19)
       Tr   - is the track number (0 - 199)
```

The time (t) that it takes the disk to satisfy an I/O request is the sum of three component times:

```
Ti/o = Tseek + Trotation + Ttransmit

where Tseek     - is a linear function of the number of tracks that
                  the head must cross in order to get from its
                  current track Tinitial to the track Ttarget which
                  contains the sector it is to read or write.
      Trotation - some constant time
      Ttransmit - some constant time
```

You are to write a C++ program using object-oriented design that will test the following disk scheduling policy:

Shortest Seek Time First (SSTF) - This policy uses a priority queue of I/O requests determined by the distance between the target track and the track on which the read head is currently positioned: priority = abs(Tr - Tcurrent). The disk head will always be moved the shortest distance to the next I/O request thereby spending less time seeking and more time transmitting data.

Notes:

1. Implement your general priority queue using a max heap stored as a vector. I have provided you with a header file for the priority queue called "MaxHeap.h", and you must implement the definitions for all of the functions listed in a file called "MaxHeap.cpp". Do not change any of the function names or data types.
2. To implement the special details of the priority queue used for disk scheduling, such as updating all of the priorities, you will need to subclass "MaxHeap.h" with a new class called "DiskSchedulingHeap.h" for example. Add any specialized functions and/or data to this class.
3. I have provided you with a file "Node.h" containing an abstract class with three pure virtual functions that must be overloaded. These are *comparator* to compare the current node with the one passed in, *changeKey* to change the key of the current node and *outputNode* to output the contents of the current node. You must subclass "Node.h" to hold the contents of the specific node for disk scheduling and implement the pure virtual functions. If you want to be adventurous, make *comparator* an overload of the > operator.
4. Initialize your request queue from the data in the file "init.dat". This will give the effect

of having several I/O requests queued up. Also, initialize each of the node's waiting time to 0. Next you are to read a line of data from the file “diskio.dat”, queue up the request using the disk scheduling policy, and then process a request. Proceed in this manner until the request queue is empty and the file is at EOF. You will encounter the EOF first.

5. Assume that the read/write head starts at track 0.
6. The “init.dat” file requests’ id numbers will be in the 0-47 range. Further, this file could be empty.
7. The files can contain multiple requests with the same id.
8. Assuming that the disk head is on track 100 and needs to go to 105, then the tracks crossed is 5.
9. The wait time is simply the number of requests the current request had to wait before being processed. The calculation of wait time does not include tracks crossed.
10. Do not take the sector number into account when determining priority. If two requests have the same priority, take the request that was queued up first.
11. All of your output will be displayed to the screen for this assignment.
12. Do NOT use structs in this assignment.

Warning: This is not a last minute assignment. You should begin working on this assignment today so that you have plenty of time to iron out any problems you may encounter.

This assignment will also hone your skills in object-oriented design. I encourage you to come and talk to me about the design of your program. Part of your grade will be allocated to how well you designed your program.

What to Complete for Part 1

- Implement the complete class MaxHeap to perform all of the standard heap and priority queue operations.
- Subclass the Node class to hold the data needed for an integer priority queue and implement the virtual functions. You can add any other functions that you might need.
- Write a driver to test your implementations. The driver should do the following:
    - Read the following integers one at a time from a file and insert them into MaxHeap: <5, 3, 17, 10, 84, 19, 6, 22, 9>, then display the heap to ensure that the result is indeed a max heap.
    - Display the result of calling the function **`heapMaximum`**.
    - Display the heap after calling the function **`heapExtract`** on the heap.
    - Display the heap after calling the function **`insert`** with the value 20.

What to Submit for Part 1

- Submit an electronic copy of your project by 9:40am on the day that it is due. Name your project “02PUNETMaxHeap”, replacing PUNET with your PU Net ID (i.e. khoj0332).
- Submit a hard copy of the files starting with the file containing main, followed by the other classes where the header file of a class is always just before the cpp file.
- A summary of the time that you spent working on this assignment, and what slowed you down the most. Submit this document electronically as a Google Document called “02PUNetPart1” for example “02khoj0332Part1”. Create the Google document and share it with me at ShereenKhoja@gmail.com.

What to Complete for Part 2

Your program must use the data found in the files “init.dat” and “diskio.dat” and report on the following:

- The total number of requests that were processed.
- The total number of tracks the read/write head had to move to process all of the I/O requests.
- The average number of tracks the read/write head had to move for each I/O request.
- The maximum number of requests that any of the I/O requests had to wait before being scheduled. Include the identification number of the I/O request that had the longest wait.
- Show what happened at each step of the disk scheduling policy. Headings should be as follows:

```
ID#    R/W     Sector    Track    Tracks Crossed    Wait Time
---    ---     ------    -----    --------------    ---------
```

What to Submit for Part 2

- Submit an electronic copy of your project by 9:40am on the day that it is due. Name your project “02PUNETDiskScheduling”, replacing PUNET with your PU Net ID (i.e. khoj0332).
- Submit a hard copy of the files starting with the file containing main, followed by the other classes where the header file of a class is always just before the cpp file. All the files must be submitted, even if they haven’t changed from part 1.
- A summary of the time that you spent working on this assignment, and what slowed you down the most. Submit this document electronically as a Google Document called “02PUNetPart2” for example “02khoj0332Part2”. Create the Google document and share it with me at ShereenKhoja@gmail.com.

```
//*************************************************************************
 // File name:          Node.h
 // Author:             Shereen Khoja
 // Date:               02/16/2009
 // Class:              CS380
 // Assignment:         Disk Scheduling
 // Purpose:            This is the header file for the abstract Node class.
 //************************************************************************

 #ifndef NODE_H
 #define NODE_H

 class Node
 {
 public:

   // Function to compare the current Node with the passed in Node
   // It will return true if the current Node is larger than the passed in
   // Node, and false otherwise

   virtual bool comparator(Node*) = 0;


   // Function to change the key of the Node to the passed in argument
   // The argument is a void* since the key of the Node is dependent upon
   // the implementation

   virtual void changeKey(void*) = 0;


   // Function to output the contents of the Node. This will be called
   // by the function displayHeap in the class MaxHeap

   virtual void outputNode() = 0;
 };
 #endif
```

```
//***********************************************************************
 // File name:          MaxHeap.h
 // Author:             Shereen Khoja
 // Date:               02/16/2009
 // Class:              CS380
 // Assignment:         Disk Scheduling
 // Purpose:            This is the header file for the Max Heap class.
 //**********************************************************************
 #ifndef MAXHEAP_H
 #define MAXHEAP_H

 #include "Node.h"
 #include <vector>

 using namespace std;

 class MaxHeap
 {
 protected:

   // The heap stored as a vector of Node*'s

   vector<Node*> heapArray;

 public:

   // Constructor that places a null pointer at element 0 of the vector
   // This is so that the root of the heap will be located at node 1
   // and you can use the algorithms in the book

   MaxHeap()
   {
     Node* pTemp = NULL;
     heapArray.push_back(pTemp);
   }

   // Destructor for MaxHeap that will delete all of the Node*'s
   // You will NOT have to delete this pointers in your driver

   ~MaxHeap()
   {
     for(int i = 0; i < (int) heapArray.size(); i++)
     {
       delete heapArray[i];
     }
     heapArray.clear();
   }


   // Function to insert a new node into the heap and maintain
   // the heap property

   void insert(Node*);
```

```
  // Function that will display the heap. Note that this function will call
  // the Node display function to display the heap to either a file or the
  // screen as specified in Node

  void displayHeap();


  // Function to build a heap from the current vector

  void buildHeap();


  // Function to maintain the heap property starting from the given index

  void heapify(int i);


  // Function to remove the root of the heap and return it to the calling
  // function

  Node* heapExtract();


  // Function to return the root of the heap but NOT remove it from the heap

  Node* heapMaximum();
};

#endif
```