**CPS801- Project**

**Winter 2013**

**PART 1  (2 marks )**

**A)** Write a program (should be run in a UNIX compatible environment) that simulates part of a file system. At the start of the program, you will need to generate 10 files, each consisting of random size between 1 to 40 blocks; where 1 block = 512 bytes.  The content of these files is not important (see the following example).  The files can be either listed in a file called *filelist.txt* or they can be kept in the appropriate data structure, the important part here is that these information will be stored later as the directory entries on the disk by your own file system functions.

For example:

|  |  |  |
| --- | --- | --- |
| File Number | Size (blocks) | Size (bytes) |
| 1 | 20 | 10240 |
| 2 | 10 | 5120 |
| 3 | 25 | 12800 |

Write a program to simulate some of the basic file system functions (excluding disk storage implementation for now) on these files that we assume are placed in an open working directory. Your program should get the operations such as *list (list of all files), create (getting file number and initial size), rename,  delete* and performs the appropriate actions by modifying the above table. Each entry of this list can be considered as the initial file attributes  (without storage information) and will be stored on a text file for further use in the part 2.    

**PART 2 (8 marks)**

**A)** In this part you will write the main part of the program to implement the functions of the file system that are required for disk storage. Assume for each file we can have one or many disk requests containing the file number and the disk operation. The operation can be any of the operations for writing a file on disk (W) , reading a file from disk (R) or deleting a file from disk (D). A file can be requested many times each time with W or R or D operation (see the following table). To produce a group of such disk operation requests, write a  program to generate a random number of requests (in the range of 1 to 5) to each file (shown by its number) for read, write and delete operations with equal probability. Print these requests in another file that we will refer to as *fileoperations.txt* with the following content:

|  |  |
| --- | --- |
| Requested File | Operation |
| 1 | W |
| 2 | W |
| 5 | R |
| 1 | W |
| 4 | D |
| 5 | W |
| 1 | R |

Now write the rest of the program to   keep track of disk space and to implement the file system disk functions. Your program manages a disk space by using appropriate data structure. Your program also should use another data structure/s to maintain the information of the file attributes from part1 and additional information needed for storing the files on disk. Based on the file allocation method that is explained in next paragraph, your program performs the requested operation for each file as listed  in *fileoperations.txt* by allocating disk blocks to that file and updating the related data structure/s.

Assume each block has a size of 512 bytes and only 200KB of a hard disk (400 blocks) is available to allocate to these files.  You will need to implement the three  write( ), read( ) and delete( ) functions for two types of file allocation methods. First allocation method is for contiguous block allocation, where each file is stored in consecutive disk blocks. Second allocation method is linked list block allocation, in which each file can be stored on any disk blocks (random block number in range of 1 to 400). Make any necessary assumptions for different situations and provide appropriate messages in your functions.

The result of your program should be shown during reading each line of file *fileoperations.txt* for each of the allocation method separately. By reading each line of  *fileoperation.txt* your program should use the attribute information of the requested fileand performs the required operation by considering the file allocation method that is used .

For example: the result of your program after reading the first line of *fileoperations.txt* for the contiguous allocation should be similar to the following:

Requested File          Operation                  Start block      length           Status

-----------------           -------------       -------------      ---------         ------------

        1                          W                            0                      20                  Success

And the result of the program after reading this line for the linked allocation of disk space should be similar to the following:

Requested File          Operation              disk blocks                        Status

-----------------           -------------       ---------------------------          ------------

       1                           W                     4,7,40,……….                    Success

You can define the Status with having different values such as Success, FileExists , FileDoesNotExist, AccessConflict , DiskFull and etc.

At the end of output  (we call it *trace*)for each allocation method print a final list of the files including their attribute information and the disk blocks that are allocated to each file. Also  print all the  free disk blocks remained on disk for each allocation method.

Explain any assumptions that you have made.  For example if you allowed the files to grow during your simulation, explain how you implemented the changing of the file sizes in your program and how the write( ) function deals with the file size change.

 (6 marks)

**B)** Your program should be a professional program using multi threading and semaphore wherever they are required. Explain the data structure/s that you have used in your program for each file allocation method.  Also for each allocation method suggest a directory implementation, which is the extension of each directory entry of *filelist.txt*  from part1 that includes storage information for each file allocation method. To do that show a diagram for each directory entry for the used file allocation method and any additional  structure/s that is required to be stored on file/s (disk) to be able to load the current information of directory and files of your  file system.  (2 marks)

**PART 3  Simulation of the Disk with DiskSim**

**(A)** (**Required part 1 mark**)

Install DiskSim (Ver 2.0 can be installed easier) and read its README file.  
Find :

a) Average response time:

b) Average write time:

c) Average read time:

d) Disk seek time average:   
e)  Disk  Rotation latency average:

For default disk type (HP2490\_A or HP\_c2249A) simulation when using ASCII input trace. Use the default input ASCII trace file (i.e., ascii.trace) and the configuration parameter file for HP2490\_A (i.e., ascii.parv). Note that both of these files are included in the DiskSim directories and you **don’t have to write any programs** to find this information.

Also by looking at ascii.trace (in Ver 2.0 it is trace.ascii)

f) Explain what kind of allocation method is used to store each file in ascii.trace :

**Hint:** You can download [DiskSim Ver 2.0](http://www.scs.ryerson.ca/%7Eaabhari/CPS801/disksim2.0.tar.gz)  and  also for more information see its  [pdf document](http://www.scs.ryerson.ca/%7Eaabhari/CPS801/disksim2.0-doc.pdf)    
For Ver 2.0 the command for performing the simulation with ASCII input trace is:  
 ../src/disksim par.ascii out.ascii ascii trace.ascii 0  
You can report the requested statistics for either HP2490 or HP2249 disk types. Note that to get the disk average response time you should see "IOdriver  Response time average" in generated output file . Also to get disk average read and write times use "IOdriver Non-Critical Read Response" and "IOdriver Non-Critical Write Response" statistics. For the seek and rotation times see Disk Statistics.

**(B) (Bonus Part  1.5 marks)**

In this part DiskSim will be used to simulate the disk for the file system program that you have created. Increase the number of the files to 1000 files and disk capacity to 40,000 blocks because you should increase *fileoperations.txt* to be able to produce a long input trace for your file system that will generate the input trace for DiskSim. You will  do the file contiguous allocation method of as you used in part 2 with preloading the 1000 files from part 1. Each of these files will get equal  probability of read and write (ignore the delete operation if it makes your program complicated) operation requests. Similar to what you did in  part 2 each file and the request to that will be appeared randomly in the increased  *fileoperations.txt* thatshould have at least 2500 request lines*.* Thenuse contiguous file allocation method and run your file system program and by reading each line of the increased *fileoperations.txt* produce the output trace with the same format as the DiskSim ASCII input trace (*i.e.,* ascii.trace ). Note that in the output trace there is only read and write (*i.e.,* disk operations)  that are shown as a bit flag in each request line. File number and status are not written in this trace too. Each line of this trace has the following contents:

Request arrival time (a sequential number that is a random number between 10 to 100ms more than previous request), Device number (storage component accessed by request, assume there is only one device with number of 1), Starting block number, Request size in number of blocks and finally  read or write flag (1 for read and 0 for write).

Use the trace that you have created above as the input to DiskSim and find the following information:

a) Average response time:

b) Average write time:

c) Average read time:

d) Disk seek time average:

e) Disk rotation time average:

Simulate at least two different disk types (for example HP2490\_A or HP\_c2249A) and show the above data in the appropriate graphs and compare the simulated disk types.

Hint:  You can save and run  [this program](http://www.scs.ryerson.ca/%7Eaabhari/CPS801/801project_part3.out) which is the in executable form  to see the sample of trace that your program should produce for this part. 

**(C)  (Bonus Part 1.5 marks)**

In this part you will implement a file by modified linked list allocation method and you will test it with DiskSim. Use all the parameters as above and generate an input trace for DiskSim.  For writing a file use the following steps:

1- Find a random free disk block and  use contiguous block allocation method to store the  file until storing the entire file or encountering an occupied block.      
2- By encountering an occupied block  go to step 1 and continue  for the remaining blocks of the file.  
  
Simulate at least two different disk types and compare the resulted statistics for this method and contiguous allocation method as explained in Part 4 by using the appropriate graphs.      
 **PART 4**

**Distributed File System (Bonus Part, 3 marks)**  
In this part by using rpc and connecting part 1 and part2 chane your program to function a *distributed file system*. You can assume there are different workstations and in each of them threre is one open directory with the remotely access functions that managing the files which are stored on    the harddisk of that specific workstation.  For implementation of this program you can make any assumption about the structure of distributed file system. For testing the file system that you have created  in part 1 and 2 of the project should work with the same functionality in two workstations with the rpc generated programs of client and server.    
 **Submission:**

* You are required to email and hand in the hard copy of your answers and source code of the programs  for each part on the submission days.
* You will also demonstrate your program for each part in the computer lab to the TA.
* For Part 1 the due date is  March  27. If you submit it earlier on March 20, there will be 0.5 bonus mark added to your mark on this part.
* For Part 2 and Part3 (A) due date is  April  3 .
* For all bonus parts the due date in the last lab on April 10.

GOOD LUCK!!!!