COMP 2710: Project 2 – Phase 3: Implementation

auDiskTool - A Tool for Monitoring Disk Devices

Implementation Portion **(60 points)**: turned in via Canvas

No collaboration between students. Students should NOT share any project code with each other. Collaborations in any form will be treated as a serious violation of the University's academic integrity code.

Rating

- Design difficulty: 2/5
- Implementation difficulty: 3/5
- Time required: 3/5
- Fun: 2/5

Goals of Project 2:

- To improve your software design skills
- To design and implement a non-trivial object-oriented program.
- To develop a tool to monitor disk performance in Linux.
- To add this programming experience in your resume.
- To develop a reasonably user-friendly application.

Goals of Project 2 - Phase 3:

- To use namespace.
- To learn the make utility program
- To perform separate compilation.
- To create a single compressed (.tar.gz) file

1. Overview

1.1 (45 points) Implement your auDiskTool in C++

1.1.1 Namespaces. Namespaces allow to group entities (e.g., classes, objects, and functions) under a name. With namespaces in place, the global scope in your program can be divided in "sub-scopes", each one with its own name. In this implementation, you must create one namespace.

1.1.2 Separate Compilation **[3]*.*** You must use separate compilation and create a makefile for this project.

What is Make?

Make is a program that looks for a file called "makefile" or "Makefile", within the makefile are variables and things called dependencies. There are many things you can do with makefiles, if all you've ever done with makefiles is compile C or C++ then you are missing out. Pretty much anything that needs to be compiled (postscript, java, Fortran), can utilize makefiles.

Format of Makefiles -- Variables

First, lets talk about the simpler of the two ideas in makefiles, variables. Variable definitions are in the following format:

```
VARNAME = Value
```

So lets say I want to use a variable to set what compiler i'm going to use. This is helpful b/c you may want to switch from cc to gcc or to g++. We would have the following line in our makefile

```
CC = g++
```

This assigns the variable CC to the string "gcc". To expand variables, use the following form:

```
${VARNAME}
```

So to expand our CC variable we would say:

```
${CC}
```

Format of Makefiles -- Dependencies

[3] http://oucsace.cs.ohiou.edu/~bhumphre/makefile.html

Dependencies are the heart of makefiles. Without them nothing would work.
Dependencies have the following form:

dependecy1: dependencyA dependencyB ... dependencyN

```
    command for dependency1
```

Check out the following links for more information on makefiles:
http://oucsace.cs.ohiou.edu/~bhumphre/makefile.html

1.1.3 Reuse well-written classes. You are welcome to reuse well-written classes from earlier COMP 2710 projects (but indicate where they came from).

1.1.4 No Global variables. You may not use global variables or global functions – all your data/operations must be contained within your objects.

1.1.5 Usability concerns and error-checking. Your program's output does not necessarily need to match the style of the sample output, but the contents should be understandable and no functionality should be lost. You should appropriately prompt your user and assume that they only have basic knowledge of the system.

You should provide enough error-checking that a moderately informed user will not crash your program. This should be discovered through your unit-testing of functions.

1.1.6 Comments. Follow the comment standard posted on the web or some alternate, approved standard.

1.2. (15 points) Test Results: After developing auDiskTool, actually try all of your test system cases. Actually show the results of your testing (a copy and paste from your program output is fine – don't stress too much about formatting as long as the results are clear). You should have test results for every test case you described. If your system doesn't behave as intended, you should note this.
Note: Driver output will substitute for this phase.

2. Building your Linux disk monitoring tool - auDiskTool

In this project, you will build a Linux tool called – auDiskTool – to monitor disk performance (i.e., I/O transfer rates) in a Linux system. Your auDiskTool can output a file containing a list of reports that help system administrators in configuring the Linux system to achieve good disk I/O performance.

The monitoring reports created and maintained by your auDiskTool offers statistical information on I/O transfer rates. Our long term goal is building a tool to monitor both processor and disk performance of Linux systems. However, the short term goal to be achieved in this project is to generate report of disk activities. After you complete this project, you may extend your auDiskTool to monitor processor performance in addition to disk performance.

3. Requirements

3.1. Statistical Information

Each report item recorded in an output file contains statistics on a per disk or partition basis. auDiskTool allows users to choose a particular disk or partition to monitor. If no disk nor partition is chosen by the users, then auDiskTool monitors all disks used by the Linux system.

Each report item may show the following information:
Device:
This column provides the disk or partition name (e.g., sda). If your Linux machine has new kernels, the device name listed in the /dev directory is displayed. If your Linux kernel is 2.4, the format of the names is devm-n, where m is the major number of the device, and n a distinctive number. If your Linux kernel is 2.2 (this is an uncommon case), the name will be displayed as hdiskn.

Blk_read:
This column reports the total number of reads.

Blk_read/s:
This column indicates the number of reads from the disk per second.

KB_read/s:
This column indicates the amount of data blocks read from the disk per second (i.e., measured in Kilobytes per second). **Note:** we assume the size of sector is 1 KB.

Blk_wrtn:
This column reports the total number of blocks written.

KB_wrtn/s:
This column indicates the number of data blocks written to the disk per second. (i.e., measured in Kilobytes per second.)

Blk_wrtn/s:
This column indicates the number of data blocks written to the disk per second.

3.2. Configure Time Interval and Count

Your auDiskTool can set an interval parameter, which specifies the amount of time measured in seconds between each disk report. Each disk report contains statistics collected during the interval since the previous report is recorded. Your auDiskTool also can decide the number of reports to be collected in an output file. If the interval parameter and the count parameter are not specified, auDiskTool may use default values. For example, the default value of the interval parameters is 1 second; the default value of the count parameter is 10.

3.3. Specify the File Name of an Output Report

Your auDiskTool should allow users to specify the file name of an output report. The file name may be fully specified with a path. If no path is provided, then the working directory will be the current directory where the new report file is created. If the output file exists, then new report items will be appended at the end of the existing file.
Note 1: If users specify a file name that does exist, auDiskTool must inform users that an output report file with the same name exists and reported items will be added to the existing file.
Note 2: If the report file name is not specified, then "report.adt" will be a used as a default output file name.

3.4. Specify what statistical data to be reported

In section 3.1, we list 7 statistical data items. Your tool should allow users to decide what data items to be included in a report. A configuration file (see Section 3.5 blow) stores default values for these decisions.

3.5. A Configuration File – audisktool.conf

All the default parameters (e.g., time interval, count, output file name) are stored in a configuration file. This configuration file is loaded into main memory when auDiskTool starts its execution. The configuration file name is "audisktool.conf". The format of the configuration file is:

```
Interval, count, print_blk_read, print_blk_read/s,
print_kb_read/s, print_blk_write, print_blk_write/s,
print_kb_write/s
```

The values of print_blk_read, print_blk_read/s, print_kb_read/s, print_blk_write, print_blk_write/s, print_kb_write/s can be either '1' or '0'. '1' means that the value will be reported; '0' means the value is ignored in the report.

For example, suppose we have the following configuration file:

```
5 10 1 1 1 0 0 0
```

The above file indicates that Interval is 5 seconds, count is 10, report values of blk_read, blk_read/s, kb_read/s. Do not include the values of blk_write, blk_write/s, kb_write/s in the report.

You do not need to submit this configuration file via Canvas; the TA will use the configuration file downloaded below to test your implementation.

http://www.eng.auburn.edu/users/xzq0001/courses/comp2710/audisktool.conf

3.6. Display the report
Users are allows to open the report and display monitoring records inside audisktool. If the report file does not exist or cannot be opened, audisktool must show a warning message.

3.7. System Quit
This should safely terminate the audiskTool. If any parameter (e.g., time interval and count) is updated, the system parameters must be saved back to the configuration file called "audisktool.conf".

4. Retrieve Disk Statistics

The Linux (version 2.4.20 and above) operating system offers extensive disk statistics to measure disk activities. You can use the following command to check the version of your Linux:

```
$uname -r
```

Linux Version 2.6 and above:

The Linux version of machines you remotely access in the lab is 2.6.32. Let us assume you are working on these machines. The disk statistical information can be found in `/proc/diskstats`

You can use the following command to display this file:

```
$cat /proc/diskstats
```

Example 1: Below is an example to show the format of the above file:

```
   3    0   sda 446216 784926 9550688 4382310 424847 312726 5922052
19310380 0 3376340 23705160
   3    1   sda1 2 0 4 24 0 0 0 0 0 24 24
```

You also can use the following command to display the information related to disks in the `/proc/diskstats` file.

```
$grep 'sda ' /proc/diskstats
```

Note that `grep` is a utility program for searching plain-text data sets for lines matching a regular expression (e.g., 'sda' in our case).

5. Format of "/proc/diskstats"

In example 1 shown on page 4, you can find each row has 14 items. The first three items are the major and minor device numbers, and device name. For example, given the following row:

```
   3    1   sda1 2 0 4 24 0 0 0 0 0 24 24
```

The major device number is 3, and minor device number is 1, and device name is sda1.

The 11 fields listed after the device name are statistics data of the device whose major/minor device numbers as well as name are shown in the first three fields. All these 11 fields except field 9 are cumulative since boot. Note that field 9 goes to zero as I/Os complete; all others only increase. These fields are unsigned long numbers.

The 11 fields are explained below:

Field 1: # of reads completed. This is the total number of reads completed successfully.

Field 2: # of reads merged.
Field 6: # of writes merged. Reads and writes which are adjacent to each other may be merged for efficiency. Thus two 4K reads may become one 8K read before it is ultimately handed to the disk, and so it will be counted (and queued) as only one I/O. This field lets you know how often this was done.

Field 3: # of sectors read. This is the total number of sectors read successfully.

Field 4: # of milliseconds spent reading. This is the total number of milliseconds spent by all reads (as measured from __make_request() to end_that_request_last()).

Field 5: # of writes completed. This is the total number of writes completed successfully.

Field 7: # of sectors written. This is the total number of sectors written successfully.

Field 8: # of milliseconds spent writing. This is the total number of milliseconds spent by all writes (as measured from __make_request() to end_that_request_last()).

Field 9: # of I/Os currently in progress. The only field that should go to zero. Incremented as requests are given to appropriate struct request_queue and decremented as they finish.

Field 10: # of milliseconds spent doing I/Os. This field increases so long as field 9 is nonzero.

Field 11: weighted # of milliseconds spent doing I/Os. This field is incremented at each I/O start, I/O completion, I/O merge, or read of these stats by the number of I/Os in progress (field 9) times the number of milliseconds spent doing I/O since the last update of this field. This can provide an easy measure of both I/O completion time and the backlog that may be accumulating.

6. Programming Environment

Write a short program in C++. Compile and run it using the g++ compiler on a Linux box (either in Shop 3, computer labs in Shelby, your home Linux machine, a Linux box on a virtual machine, or using an emulator like Cygwin).

7. Deliverables

7.1 No hardcopy is required for the final submissions.

7.2 Please create and submit a single compressed (.tar.gz) file named "<username>_project2.tar.gz" (for example, mine might read "xzq0001_project2.tar.gz") through the Blackboard system online. Next section shows you how to compress files in Linux.

- **Makefile** (this should make your project)
- **Multiple CPP files** (The source code of class implementations)
- **Multiple header files**
- **Project2_results.pdf** (This file includes all process information, especially the results of testing)

Note: You will lose points (at least 5 points and up to 10 points) if you do not submit a single compressed file and name your compressed file in the format described in this section.

7.3 Create your compressed file:

To create a compressed tar.gz file from multiple files or/and folders, we need to run the tar command as follows.

```
tar -czf <username>_project2.tar.gz <folder_2710_project2_folder>
```

where `<folder_2710_project2_folder>` is a folder that contains Makefile, header file, and other source code of your project;
`<username>_project2.tar.gz` is the single compressed file to be submitted via BLACKBOARD. For example, my single compressed file to be submitted can be created using the following command:

```
tar -czf xzq0001_project2.tar.gz ./comp2710/project2
```

where `./comp2710/project2` is a folder that contains files for my project 2.

8. Grading Criteria

8.1 (45 points) Implementation

- (2 points) submitted file names
- (3 points) Use comments to provide a heading at the top of your code containing your name, Auburn Userid, and filename. Please describe any help or sources that you used (as per the syllabus).
- (3 points) namespace
- (5 points) makefile
- (5 points) multiple source code files.
- (15 points) report class, configuration class, tool class, and commandline class
- (2 points) print help information
- (5 points) the main() function
- (5 points) coding style

8.2 (15 points) Test Results

- (6 points) Run
- (3 points) Set
- (3 points) Print
- (2 points) Save
- (1 points) Help

9. Late Submission Penalty

- Twenty percent (20%) penalty per day for late submission. For example, an assignment submitted after the deadline but up to 1 day (24 hours) late can achieve a maximum of 80% of points allocated for the assignment. An assignment submitted after the deadline but up to 2 days (48 hours) late can achieve a maximum of 60% of points allocated for the assignment.
- Assignment submitted more than 3 days (72 hours) after the deadline will not be graded.

10. Rebuttal period

- You will be given a period of 72 hours to read and respond to the comments and grades of your homework or project assignment. The TA may use this opportunity to address any concern and question you have. The TA also may ask for additional information from you regarding your homework or project.

11. Hints

- Start early, you have a good deal of time but you may need it to debug your

program. Although the following timeline is not mandated, it is a suggestion of milestone:
    - 1/4 time: Finish process planning. Implement infrastructure and be able to load the files.
    - 2/4 time: Implement the basic classes as well as vectors for lists of branch staff employees, clients, and accounts sort.
    - 3/4 time: Implement system login, administration management, client/account management
    - 4/4 time: Complete interfaces, appropriate testing information, complete testing and finish final documentation.
- If you bring your design documents by early, I will give you comments and help point you in the right direction on this project.
- Note that this teller terminal system is not secure, because passwords are saved in plaintext in a file. In a future project, all the passwords will be encrypted.