Monitoring & Sorting in Multiple User Accounts

From: Sheldon Maschmeyer and Alexi Kent
Student #: 100808353, 100974542
Instructor Michel Barbeau

Date Due: April 7, 2017
Date Submitted: April 7, 2017

1. Introduction

1.1 Context

It is difficult to monitor the activity of multiple users. In the work place, there is no supervisor looking over the shoulder of employees. At home, a parent may wish to monitor a storage folder their children have access to in their user accounts. A simple fix for this is to have a place in the computer where files can be placed without worrying about organization. Thus, letting admins have more time to do their work than worrying about their underlings' work while allowing underlings to not worry as much about organization. Also, admins (i.e. supervisors and parents) are not always at the computer they are monitoring; therefore, an email would be sent to notify them remotely of changes made to the folder(s) being monitored. The user accounts are local ("real") user accounts that can be accessed by logging locally or remotely and transferring files to the folder. If the monitoring program is started by the administrator and continues to run in the background, the folders of the other user accounts will be monitored.

1.2 Problem Statement

This program addresses two issues. The first is the need to monitor a folder across multiple user accounts. The second is to sort files into sub-folders automatically. Monitoring of folders is useful for multiple reasons. If the administrator is a parent, the parent can monitor the activities in their children's folder, such as if they are actively doing homework. The monitoring may also be used if a person is sharing a file with another person and wants to know if files have been changed, accessed, or modified. Administrators may face an additional challenge if they do not spend most of their time in front of the PC they want monitored. Therefore, assuming another user is on the PC, the administrator is notified via e-mail when files have been added to a user's storage folder.

1.3 Result

While the configMonitor has bugs that need to be addressed, and the file organizing should utilize symbolic-links, it is a good proof-of-concept. The program monitors the home/storage folder of user accounts, notifies by email when an event has been detected, and sorts the folder. Therefore, the primary goals were accomplished.

1.4 Outline.

Section 2.1 is the initial concept, where the idea for monitoring and restructuring of user account's home/storage folder originated. Section 2.2 is the program implementation including the use of iNotify to monitor the folder in real-time.

2. Background Information

2.1 Initial Concept

Originally, I setup an ftp (file-transfer-protocol) server, but due to a concern that the shared files were semi-confidential materials between business partners, I created an sftp (secure-ftp over SSH (Secure Shell)) server. One issue was that the administrator could not be notified when a partner accessed or modified files. I wrote a simple shell script that monitored the sftp folders every 60 minutes and emailed him when files were accessed or modified by comparing log files generated using the ls command. It was used for a short time after, however, due to the complexations with added security, requiring security certificates for ease of use through a web-browser, they switched to Dropbox. While the program was limited in scope and poorly executed – comparing log files using 100% system commands – we were hoping to refine the concept and build upon it into something more complex. Our final vision was a user-friendly server that would become a useful and integral part of the OS.

Using other peoples' computers, I noticed that the average users do not organize their files; instead, their files are a mess in both the documents and downloads folders. Few files are sorted into directories and they often have trouble finding things needed like an older document. Using a program that automatically sorts files into subfolders could be enormously beneficial to people who need to sort their files but do not have the time to manually move them. For emails, Google has an auto-sorter that sorts email into categories. The only issue with their approach is that it takes some files out of the primary inbox, which could go un-seen. I have noticed the issue when I receive emails marked important going into an important folder but not my primary inbox. So, the auto-sorter, if used in a practical environment, needs to be customizable, allowing the user to select which folders files with specific extensions or other properties belong. This is another reason for the email notification: while it is nice to auto-detect and sort, alerting the user that the file was in the directory and sorted allows them to find the files later.

2.2 Program implementation

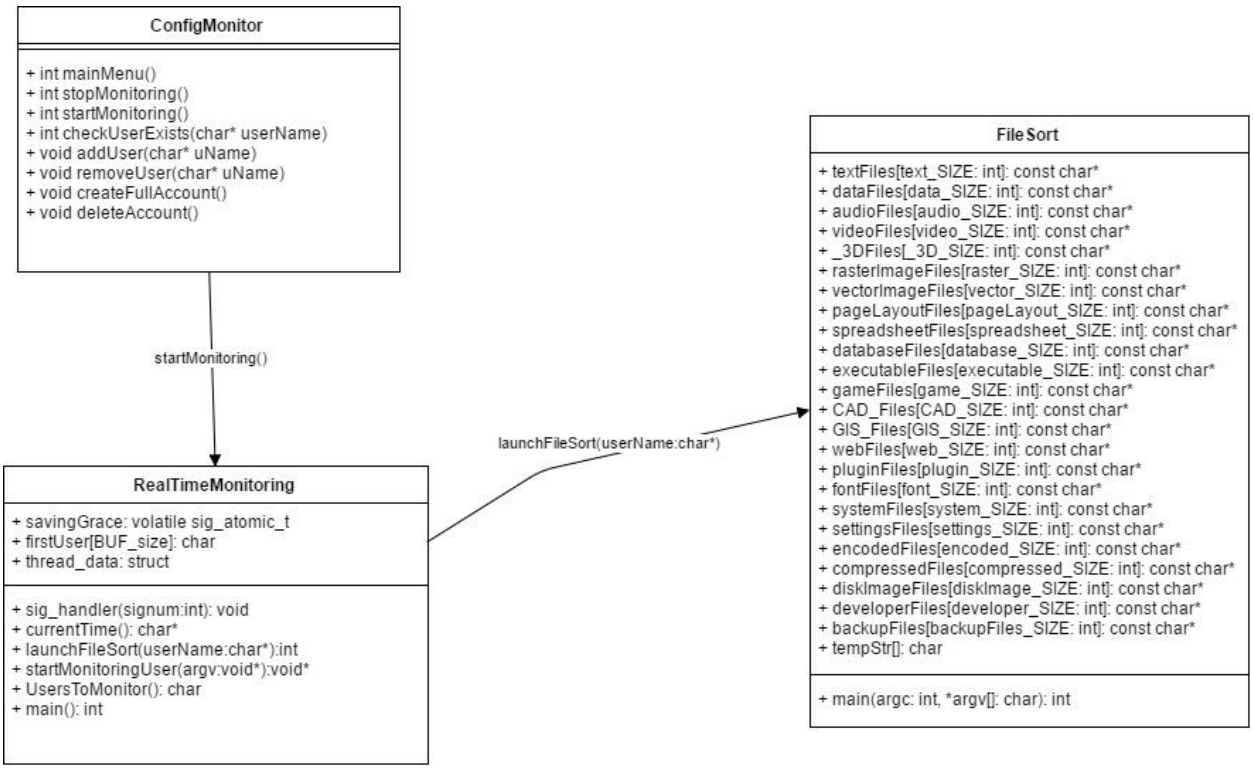
The monitoring part of the program is designed to monitor the home/storage directory of users for file changes, and then sort them into folders. The real-time-monitoring utilizes the iNotify kernel subsystem first available in kernel version 2.6.13 (Ubuntu version 4.10+). The real-time monitoring does not recursively check through all the directories in a folder for changes, but rather, only monitors the root directory. Since files are being moved into sub-directories, recursively monitoring sub-directories would cause more issues than it would solve. In the future, the sub-directories would only contain symbolic-links to the real files in the root storage directory, making the need to monitor the sub-directories a non-issue. Each user's home/storage directory is being monitored in its own thread. Each thread is passed thread-data array which contains the userName, and sleep/pause between events timer. Since there are no shared modifiable variables between the threads, a mutex is not required. As the sorting is currently extensively using system commands, it is being forked and run in its own processes to not interfere with the rest of the program. While fanotify is newer than iNotify, both coexist on

the latest Linux kernel since fanotify lacks some features of iNotify and vice versa. It is 'easier' with fanotify to recursively search sub-directories of a folder being monitored and allows for larger and more directories to be watched. For instance, iNotify, by default, is limited to 128 instances and 8192 watches per user. However, fanotify cannot check for deletes, renames or moves. The current Ubuntu distro Nautilus file explorer does not delete files but rather moves them to another location, the trash folder, and creates links. Therefore, iNotify checking for a delete event is pointless since they would be identified as move events. iNotify is a replacement for dNotify, which is now depreciated. Also, fanotify is a root-only process. I have learned that it is better to run the monitoring program as a regular user than through sudo [Morgado, 2013]. When moving files and folders, using the root user changes file permissions and creates, frankly, a mess as you can't delete some files from the trash bin. Therefore, since iNotify is a user-accessible kernel module, and all user storage folders have read-write permissions by the same group, it is preferred. The template code and description for how to use iNotify is from an article by Martin Streicher on IBM developerWorks website [Streicher, 2008]. This was the foundation which was used to build the monitoring system. It was extensively modified and changed. An infinite loop was made with a signal handler, passing the signal to the threads, to break out of the loop. To interrupt the monitoring when it is waiting for an event, an empty file is created called "stop.thread" and immediately deleted, breaking out of the loop without performing any further tasks and thereby properly removing the watch from the folder and closing the file descriptor. The initial template was looking for specific events and performed a different action based on the event detected. It uses a watch descriptor pointer that takes in a file descriptor pointer, the directory location to watch and what to watch: `wd = inotify_add_watch(fd, fullDir,IN_ALL_EVENTS)`. For the purposes of this program, a catch-all (IN_ALL_EVENTS) was used for simplicity. After an event is detected by iNotify, it checks the root storage directory for files that need to be moved, ignoring directories: `while(ep = readdir(dp)) {if(ep->d_type == DT_REG) {strcat(filesList, ep->d_name); strcat(filesList, "\n");}}`

. If they exist, it sends and email using the external program Mutt as a formatted string:

```
if(strcmp(filesList, "\n") != 0) {printf("File(s) in a directory were created or
modified.\n"); strcpy(email, "echo \"Notice: Files created or modified by ");
strcat(email, userName);strcat(email, " ");strcat(email, currentTime());strcat(email, "
\n");strcat(email, filesList);strcat(email, " \" | mutt -s \"User: ");strcat(email,
userName);strcat(email, " created or modified files on PC ");strcat(email,"\"
comp3000sheldon@gmail.com");system(email);launchFilesort(userName);}
```

It then launches the Filesorting program as a forked process that is described later in this document. Before the loop goes through another iteration, I used fsync(fd) to flush/sync the file descriptor before it reads and waits for an event again.



The list of users being monitored is stored in a persistent text file. The configMonitor edits the MonitorList.txt file, adding and removing users to monitor. The RealTimeMonitoring class reads the file, each user is sent to startMonitoringUser and starts in its own thread. When the computer is booted and the admin is logged in, the program starts automatically and monitors users that are on the list. Each class is like an independent module of the program that could run in isolation. The Real-time-monitoring can be adapted and modified, removing a function, to be used with or without FileSort. The restructuring, auto-organizing of the root folder into subfolders can be run manually without RealTimeMonitoring. This flexibility allows for adaptability in the future. The ConfigMonitor passes a SIGTERM to the RealTimeMonitoring class which is caught by the sighandler (catches SIGTERM and SIGINT) for a proper shutdown of monitoring program. Because the RealTimeMonitoring is launched on boot and not necessarily started by the configMonitor, it is passed to the RealTimeMonitoring through “sudo killall -s SIGTERM -e RealTimeMonitor”. The -e is for exact name.

3. Result

The real-time monitoring worked without errors. Sorting files into folders worked but could be further optimized. The ConfigMonitor utility did not perform all the tasks it was designed to adequately. On demoing, the ConfigMonitor failed to create the storage folder in the home directory. Safety features such as ensuring that the administrative user with access to the monitoring program could not be deleted only worked if the user checked for is hard-coded. To make the program portable, it needs to check for the admin user by seeing if the user name is the same as the user logged in. This did not work as expected, destroying snapshot OS images.

4. Evaluation

The limitation of the sorting algorithm being used is that it has a lot of overhead requiring the extensive use of system commands, some which fail because the file with that extension does not exist. The algorithm, in the future, could be more efficient and faster by detecting the files that need to be sorted, determining file extensions, and then performing the operation required.

5. Conclusion

5.1 Summary

This program is a very good proof of concept but needs lots of work to make it real-world useful and bug free.

5.2 Relevance

The relevance to the course is that the monitoring part of the program utilizes the kernel feature iNotify and may be used by parents or admins to monitor the folder of other user accounts. It also has practical applications for file sharing with notifications either between users on a local machine or remotely.

5.3 Future Work

The email notification should be improved to notify the admin of the sub-directory the files were sorted into.

A limited usability study consisting of three people was conducted in the computer science lounge after the program demonstration. The following questions were asked: How easy is it to use the program, how intuitive is the configMonitor setup interface to use, what improvements would you like to see, any bugs you have noticed, and is the program deployable or needs improvements. From the feedback, the following future improvements were recommended. Bugs such as the “storage” folder not being automatically created need to be fixed in the configuration of user accounts. Adding and deleting user accounts also needs to be smoother.

A web interface backend was suggested instead of, or in addition too, sending emails to notifying the admin that files were added to the storage directory. The web interface could allow configuring of user accounts remotely. The benefit of using a simple email to notify, like the text pagers in the 1990s, is that the person subscribed to the email notifications may check their email on their mobiles or other devices without installing special software.

Another suggestion was, instead of sorting based on file extension, use the hex signature of the file to determine the file type. For instance, the hex signature of an executable in windows is “4D, 5A” [Notenboom, 2011]. This way, if a file does not have a file extension, or is masquerading as another file, it can attempt to properly detect the file. This method of detection is most commonly used to detect executables masquerading as other file types in an email attachment, for example. This method of detection can utilize either a file signature table or database of known file signatures. This method of detection is also flawed since a file might contain multiple signatures, for example, zip, jar, odt, ods, odp, xlsx, docx, pptx, vsdx and apk. In these cases, they might all be detected as zip files since they contain the signature for a zip file, but also, in theory, might have a unique signature [Hector]. For instance, docx and xlsx for Office 2007 both have the signature 50 4B 03 04 14 00 06 00, while the zip signature is 50 4B 03 04 [“File Signature Database”, 2017]. Therefore, the ideal approach would be to combine file extension recognition with signature detection for a ‘sanity check’.

As discussed during the program demonstration, instead of moving files into the subdirectories, we could create a symbolic link to the file, a shortcut, in the subdirectories. That way, the files are not physically moved which can cause issues if the file was created by an editor or linked too by other programs. Also, the monitoring program currently detects when a file is accessed or modified using iNotify, therefore a symbolic link would allow for more robust monitoring where the admin can be notified when files are added, accessed, modified or moved with unique notifications (i.e. Files were Accessed by {user} on {date}). The email notification should be improved to notify the admin of the sub-directory the files were sorted in.

The monitoring is of use to only the administrator and for the other users in sorting and organizing their files automatically; however, the email notifications could have an option to also notify the users when files in their storage folder are accessed or modified.

Currently, an integer is used to sleep/pause the iNotify event monitoring, set to 10 seconds or real-time. In practice, the configMonitor should allow the admin to determine the length of time before detecting an event. When transferring large amounts of files, the notification and sorting could be set to 30 or 60 minutes, but should be at the discretion of the administrator. If using symbolic-links to files in sub-directories, this would mostly affect the number and frequency of email notifications.

6. Contributions of Team Members

A. **Sheldon Maschmeyer**:

The RealTimeMonitor part of the program which monitors for a change in the /home/storage directory of users on the monitoring list configured through configMonitoring, sending an email and launching the sorting.

The configMonitoring utility which performs user management and prepares the program for real-time monitoring.

B. **Alexi Kent**:

The FileSort part of the program. FileSort has, for each file category, a constant character pointer array that holds the most common file extensions for their respective category [“Common File Extensions”, 2017]. The code then organizes the files into subdirectories that are appropriate for the file extensions. FileSort also ensures that the subdirectories are made inside of the user’s storage folder.

References:

“Common File Extensions.” Accessed. 7 Apr. 2017. <https://fileinfo.com/filetypes/common>.

“File Signature Database.” Accessed. 7 Apr. 2017. <https://www.filesignatures.net/>.

Morgado, Aleksander. “Filesystem Monitoring in the Linux Kernel.” 26 Nov. 2013. Accessed. 7 Apr. 2017. <http://www.lanedo.com/filesystem-monitoring-linux-kernel/>.

Hector, Guo. “How to Detect File Type through HTML5.” *hectorguo.com*. Accessed. 7 Apr. 2017. <https://hectorguo.com/en/file-signature-check/>.

Notenboom, Leo A. “How Do I Figure out What Kind of File I Have - without the File Extension?” *ask-leo.com*. 2011. Accessed. 7 Apr. 2017. <http://ask-leo.com/how_do_i_figure_out_what_kind_of_file_i_have_without_the_file_extension.html>.

Streicher, Martin. “Monitor File System Activity with Inotify.” 16 Sept. 2008. Accessed. 7 Apr. 2017. <http://www.ibm.com/developerworks/library/l-ubuntu-inotify/index.html>.