

Systems Software Report CA1

TU856

BSc in Computer Science

**Aaron Fu Siang Ann**

**D19126659**

School of Computer Science

TU Dublin – City Campus

**6 November 2021**

**Table of Contents**

*Functionality Checklist* 3

*Feature 1 - System Architecture including makefile* 4

*Feature 2 - Daemon (Setup/ Initialisation/ Management)* 4

*Feature 3 - Daemon (Implementation)* 4

*Feature 4 - Backup Functionality* 5

*Feature 5 - Transfer Functionality* 5

*Feature 6 - Lockdown directories for Backup / Transfer* 5

*Feature 7 – Process management and IPC* 5

*Feature 8 - Logging and Error Logging* 5

*Conclusion* 6

# *Functionality Checklist*

|  |  |  |
| --- | --- | --- |
| ***Feature*** | ***Description*** | ***Implemented*** |
| F1 | System Architecture including makefile | Yes or No |
| F2 | Daemon (Setup/Initialisation/Management) | Yes or No |
| F3 | Daemon (Implementation) | Yes or No |
| F4 | Backup Functionality | Yes or No |
| F5 | Transfer Functionality | Yes or No |
| F6 | Lockdown folder for Backup / Transfer | Yes or No |
| F7 | Process management and IPC | Yes or No |
| F8 | Logging and Error Logging | Yes or No |

Have you included a video demo as part of the assignment: Yes or No

Link to Video: <https://www.youtube.com/watch?v=Lwocznmc6rk>

Declaration

I hereby declare that the work described in this dissertation is, except where otherwise stated, entirely my own work and has not been submitted as an exercise for a degree at this or any other university.

Signed:

Aaron

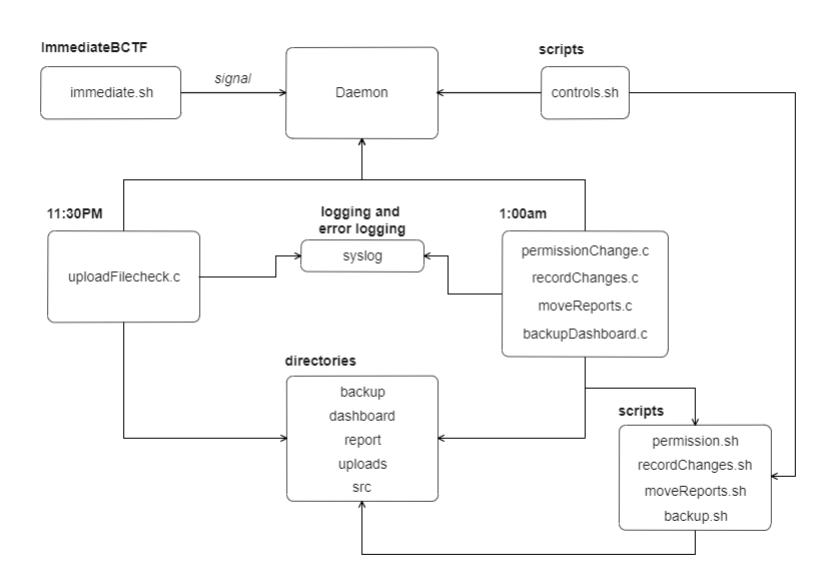
AARON FU SIANG ANN

6 November 2021

# *Feature 1 - System Architecture including makefile*

Separation of Concerns (SoC) facilitates code reuse, separate classes for different data and behaviour. In the daemon permissionChange() demonstrates this principle as when locking and unlocking directory would use the same command but different permission values. To avoid repeating code, permissionChange() takes in an argument to decide whether to lock or unlock directory.

Each functions in the daemon are in a separate file, a C and header file. Each of them is written to be focused only on one task. The programs are compiled to work together in the end. Thus, abiding to the Single Responsibility Principle (SRP).



**Architecture Diagram**

# *Feature 2 - Daemon (Setup/ Initialisation/ Management)*

The daemon is setup to execute tasks at 11:30pm and 1:00am. Before executing all the tasks, auditd watch is put on the uploads directory for documenting purposes. At 11:30pm, the daemon will check the uploads directory if all required xml reports are uploaded. If the required xml reports are not uploaded, it will be logged to the system. At 1:00am, the daemon locks the uploads and the managers will not have access to the directories during this time. Changes to the uploads directory will be documented and stored under the report directory. After this, the reports will be transferred from uploads to the dashboard and the dashboard will be backed up if the transfer is successful. Finally, both uploads and dashboard will be unlocked, granting managers access to the directories again.

Daemon will be initialized on start up. A script is created under **ImmediateBCTF** directory to perform immediate backup and transfer at any time required. This is implemented by having a signal interrupt handler in the daemon. User can just run the script to send an interrupt signal to the daemon. The process id of the daemon is stored in pid.txt for further implementations in the future, makes it easier for other processes to identify the daemon. Another script created called “**controls.sh**” under scripts directory can be used by the managers to start or stop the daemon. Simply call ./controls.sh (action wanted) without the brackets.

# *Feature 3 - Daemon (Implementation)*

To create a background process, fork is called first. The parent process will exit while the child process runs setsid(). This allows the child process, otherwise known as an orphan process to a session leader to lose controlling TTY. The command lets the process run in a new session. Then fork is called again to check the process is not a session leader, umask is called after to set the file mode creation mask to 0. Next, the currently working directory is changed to root to eliminate any issues of running on a mounted drive. Close all open file descriptors and keep the process running with an infinite loop.

# *Feature 4 - Backup Functionality*

The backup functionality was implemented by calling a script. Firstly, the backup method was created in a separate file for code readability and named as backupDashboard(). This method only be called if the transfer function executes without any issues. The reason for this is because if for instance, the transfer files failed then there will be no files in the dashboard to backup and this would result in an error when executing backupDashboard(). In the separate file itself, it forks a child process to run a script using execl. Script uses the command tar to perform the backup and files are compressed into gzip. The parent process then waits for the child to finish executing the script before returning the exit status of the child process. If execl is successful, the next line will not be executed in the child process but if execl is unsuccessful, the next line exit(EXIT\_FAILURE) status will be returned to the parent process.

# *Feature 5 - Transfer Functionality*

The transfer functionality was implemented by calling a script. Firstly, the transfer method was created in a separate file for code readability and named as moveReports(). This method will be called by the daemon when time reaches 1am. In the separate file itself, it forks a child process to run a script using execl. Script uses the command mv to perform the transfer. The parent process then waits for the child to finish executing the script before returning the exit status of the child process. If execl is successful, the next line will not be executed in the child process but if execl is unsuccessful, the next line exit(EXIT\_FAILURE) status will be returned to the parent process.

# *Feature 6 - Lockdown directories for Backup / Transfer*

The lockdown directories functionality was implemented by calling a script. Firstly, the lockdown method was created in a separate file for code readability and named as permissionChange(int num). This method will be the first to be called by the daemon when time reaches 1am. It takes an integer argument to distinguish whether to lock or unlock the directories. Before performing other functions, it first locks the directories with 700 which grant access only to the owner in case anything happens. Both directories group are set to the departments. In the separate file itself, it forks a child process to run a script using execl. Script uses the command chmod to perform the lockdown. The parent process then waits for the child to finish executing the script before returning the exit status of the child process. If execl is successful, the next line will not be executed in the child process but if execl is unsuccessful, the next line exit(EXIT\_FAILURE) status will be returned to the parent process. After performing all the necessary functions in the daemon, this method will be called once again with 770 being passed in as the argument. Giving all access to the department group only.

# *Feature 7 – Process management and IPC*

Most of the functions implemented in daemon uses execl to execute scripts created to perform certain function. But the problem with execl is that it replaces the current program. To tackle this, first fork a child process to perform execl. If it executes without issues, this will communicate a success message to parent represented with an integer 0. However, if execl fails, the child process will exit with exit(EXIT\_FAILURE) and this will be communicated to the parent process as failure. Another problem with fork is that the processes will not wait for each other to complete and executes the rest of the code. In order to solve this, a waitpid() is included in the parent process so that it will wait for the child process to finish executing before performing its own executions. In other words, parent process will wait for the child process to finish and retrieve the exit status of it. This then allows the daemon to log if the function execution is successful or not.

# *Feature 8 - Logging and Error Logging*

For logging, two types of function are used. For most of it, syslog() is utilized for logging if the functions in daemon executed successfully or not. These logs are stored in /var/log/daemon.log. To find logs specifically for this daemon, “cat daemon.log | grep Manufacturing logger” in the terminal will display all logs related such as if a function in daemon failed or succeeded. On the other hand, another function used for logging is auditd. For auditd to work, a watch was first placed on the “uploads” directory in the command line. This was implemented to keep track of changes to the “uploads” directory. If any changes were made to this directory, auditd will log all the changes. The changes are documented into a txt file which can be found under the report directory, named report.txt.

# *Conclusion*

In conclusion, the daemon runs when system boots and can be controlled with a script implemented under scripts directory called “**controls.sh**”. Managers can simply call ./controls.sh (action wanted) to start, stop or restart the daemon without the brackets. The daemon can perform transfer and backup at anytime with the script under **ImmediateBCTF** directory. However, backup will not execute if the transfer failed. For example, the uploads directory does not have any files which can cause an error because there are no files to transfer, and it would not make sense to execute backup with an empty dashboard. At 11:30pm, daemon will check for files required in uploads the directory. This will be logged in **daemon.log** if the files were uploaded on time or not. At 1:00am, uploads and dashboard directory will be locked, files will be transferred from uploads to dashboard and a backup will be executed on the dashboard directory. Backups are stored under backup directory in the form of .gzip extension, compressed. Finally, both directories are unlocked, and managers can access them again.

Most functions utilize **exec** to run scripts which requires forking of a child process. The parent process will wait for the child process to exit, at the same time acquiring the exit status of the child process. This status will then be used for logging success or failure of the function.