

Systems Software Report CA1

TU857

BSc in Computer Science

**Conor Farrell**

**C19704439**

School of Computer Science

TU Dublin – City Campus

**19/03/2023**

**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* 6

*Feature 8 - Logging and Error Logging* 6

*Conclusion* 6

# *Functionality Checklist*

|  |  |  |
| --- | --- | --- |
| ***Feature*** | ***Description*** | ***Implemented*** |
| F1 | System Architecture including makefile | Yes |
| F2 | Daemon (Setup/Initialisation/Management) | Yes |
| F3 | Daemon (Implementation) | Yes |
| F4 | Backup Functionality | Yes |
| F5 | Transfer Functionality | Yes |
| F6 | Lockdown folder for Backup / Transfer | Yes |
| F7 | Process management and IPC | Yes |
| F8 | Logging and Error Logging | Yes |

Have you included a video demo as part of the assignment: Yes

Link to Video: please paste link here

https://www.youtube.com/watch?v=5AIRxG\_D8xI

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:

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Conor Farrell

19/03/2023

# *Feature 1 - System Architecture including makefile*

The daemon adheres to the Separation of Concerns (SoC) and Single Responsibility Principle (SRP). One such example of this is:

Backup\_dashboard(): This function adheres to SRP and is responsible for backing up the contents of the source directory to the destination directory. While it does handle multiple tasks such as creating a new directory, opening the source directory, iterating through its entries, and copying the files, all these tasks are related to the overall goal of backing up the directory.

The main functionalities of the Daemon all follow this example whereby the actions are separated into their own files/functions.

The project destination folder layout is as follows:  
Text

Description automatically generated

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

The project contains an “init.sh” file. When this is run it will use make to build the daemon executable which will in turn be configured to run as a startup service.

Once the daemon has started it will print its PID which can be used alongside the “kill -2” command to perform a manual backup. The PID of the daemon can also be retrieved using the “ps aux” command.

# *Feature 3 - Daemon (Implementation)*

The code uses the fork() system call to create child processes in the background. The child processes are then used to carry out various tasks such as logging and message queue processing while the parent process continues to run the main daemon code.

It also employs the daemon pattern, in which a child process is created and the parent process is terminated. The child process is then promoted to session leader and runs as an orphan process in the background. This is done to ensure that the process is not linked to any controlling terminal and that it can run without user input.

Signal handling is also used in the code to manage background processes. When the process is terminated, the signal(SIGINT, sig handler) function is used to handle the SIGINT signal.

# *Feature 4 - Backup Functionality*

The code loops through all of the files in the source directory and copies them to the destination directory using the fopen(), fread(), and fwrite() functions.

The copied files are organized in a directory named after the current date.

It begins by opening the message queue and dispatching a message indicating that the backup\_dashboard function has been launched. The memory for the file directories is then allocated, and the current date is obtained. It uses the mkdir() function to make the destination directory and constructs the source and destination file paths for each file in the source directory.

For each regular file in the source directory, the code opens the source and destination files using the fopen() function, reads the contents of the source file using the fread() function and writes the contents to the destination file using the fwrite() function. After copying each file, it logs the file name to the syslog and to the terminal.

# *Feature 5 - Transfer Functionality*

The function first establishes the source and target directories. Then, using pathconf, it specifies variables for the source and destination file paths, as well as the maximum file path length. If the maximum file path length cannot be determined, a default number is used. It then uses opendir to open the source directory and readdir to loop through all files in the directory. It skips the "." and ".." directories and uses a for loop to see if the current file name fits any XML files in the array.

If the current file name matches an XML file in the array, it uses snprintf to build the source and destination file paths and stat to get file information. If the file is a regular file, it is renamed and moved to the target directory. If the move fails, an error notice is logged and a message is queued. If the transfer is successful, a message is logged and a success message is sent to the message queue.

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

To modify the permissions of the directories, the lock\_directories() function calls the chmod() system call. If an error occurs while changing permissions, an error message is printed to the console and a message is sent to the "/daemon\_queue" message queue showing that the action was unsuccessful. If the permissions are successfully changed, a message showing that the action was successful is sent to the same message queue. Finally, the console displays a success notification.

# *Feature 7 – Process management and IPC*

This program creates a message queue and monitors it for communications. When a message is received, it examines to see if the words "successful" or "unsuccessful" are present. If it does, it displays the message to the console and saves it to a log file, along with a timestamp and the tag "DAEMON".

The code first configures the message queue's attributes, such as the max number of messages and their size. It then uses mq\_open() to construct the message queue with the specified name and attributes.

It gets messages from the queue using mq\_receive() within the do-while loop, with a maximum size of 1024 bytes. If the message includes the words "successful" or "unsuccessful," it opens the log file, inserts a timestamp, writes the message to the log file with a "DAEMON" tag at the beginning, and then closes the file.

Finally, it uses mq\_close() and mq\_unlink() to close and remove the message queue.().

# *Feature 8 - Logging and Error Logging*

Logging and error logging have been implemented using the syslog function. The syslog function is used to send messages to the system logger, which then writes them to a log file.

In the message\_queue() function, logging is implemented using both printf statements and writing to a log file. When a message is received from the message queue, the function checks if it contains the strings "successful" or "unsuccessful". If it does, the message is printed to the console and written to a log file with a timestamp and "DAEMON" tag at the start.

# *Conclusion*

The daemon uses the chmod function to prevent unauthorized access to two directories, gathers reports from a shared directory, and moves them to a reporting directory. It also creates a message queue and logs its activity, sending messages to the queue showing whether or not each job was successful. The daemon also ensures that the necessary directories have the correct permissions and logs any relevant information to a log file.

Overall, the daemon automates the process of collecting and organizing reports, saving time and reducing the risk of human error.