

Systems Software Report CA1

TU856

BSc in Computer Science

**Steven Aherne**

**C18742005**

School of Computer Science

TU Dublin – City Campus

**7th November 2021**

**Table of Contents**

*Functionality Checklist* 3

*Feature 1 - System Architecture including makefile* 4

*Feature 2 - Daemon (Setup/ Initialisation/ Management)* 5

*Feature 3 - Daemon (Implementation)* 5

*Feature 4 - Backup Functionality* 6

*Feature 5 - Transfer Functionality* 6

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

*Feature 7 – Process management and IPC* 7

*Feature 8 - Logging and Error Logging* 7

*Conclusion* 8

# *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: https://youtu.be/0a6aAwJo8vQ

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:

Steven Aherne

Steven Aherne

7/11/2021

# *Feature 1 - System Architecture including makefile*

Diagram

Description automatically generated

The architecture of this project has been separated out into various C files where each C file performs a single function and is concerned only about this function in order to abide by the Separation of Concerns design principle and the Single Responsibility Principle. Rather than having one large C file that included all the code, time was spent modularising the code so that each C file would perform one single responsibility but perform it extremely well. This makes for a project that is understandable, easily extensible, and modular in fashion. A brief description of each C file and its function is given below:

* manufacturingDaemon.c

This file holds the main function of the code. It is responsible for creating the orphan child process which is then converted to the daemon process through the 5-step process outlined in feature 2. It is also responsible for handling signals, locking directories, and unlocking directories.

* locs.c

This file is responsible for storing the constant values needed for the project. The constant values needed are the locations for the reporting, upload, and backup directories as well as the text file location that stores a log of all transfers and changes made to each directory.

* queue.c

The queue file is responsible for controlling the IPC that takes place within the project. It monitors any changes made, any errors that occur, or any successful functions then logging these events.

* changes.c

The changes file is responsible for creating the pipes that enable the daemon process to write any file changes to the changes.txt file while also monitoring any modifications to the upload directory.

* backup.c

The function of this file is to create a new subdirectory named using a timestamp within the backup directory then copying the contents of the reporting directory to this new subdirectory.

* transfers.c

The responsibility of this file is to transfer any created or modified files from the upload directory to the reporting directory.

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

The systemd service “*myDaemon.service*” is set up to automatically execute the daemon on system boot. This can be used in conjunction with the systemctl commands in order to produce the following results:

* Start and stop the daemon process
* Restart the daemon process
* Disable and enable the daemon process on startup
* Check the status of the process

The daemon can also be tasked with backing up and transferring files at any time the user pleases. This can be done by finding the daemons PID and sending a SIGUSR1 interrupt to the daemon. The daemon runs a backup and transfer when it encounters a SIGUSR1 interrupt.

The *locs.c* file holds all constant variables which in this case is the directory names needed for the daemon process to access while other header files contain functions that deal with the backup, transfer, changes, and queue functions.

# *Feature 3 - Daemon (Implementation)*

A 5-step process was used when setting up the Daemon:

1. To begin, the fork command was used to create a child process which initially has a parent process. This child process will be made an orphan in later steps to produce the daemon process.
2. Step 2 uses the setsid command to make the child process created in step 1 run in a new session as a session group leader and a process group leader. This enables the child process to lose its controlling TTY or terminal and become an orphan process.
3. Step 3 involves using umask to set the file creation mode to 0 allowing the process to read and write files with the permissions it requires.
4. Step 4 uses chdir to change the processes current working directory to root, eliminating any possibility of issues arising due to the process running on a mounted drive.
5. Finally, the close command was used in conjunction with a loop to release any IO connections that may have been inherited from the parent process. Sysconf is also used in conjunction with a for loop during this step to determine the maximum value of file descriptors and close them subsequently.

After following these 5 steps the daemon process is completely decoupled from its parent process and no longer has a controlling terminal making it run as a background process. Error checking is implemented throughout this 5-step process to ensure the daemon is correctly set up as a background process while a signal handler function is used to handle any signal interrupts that are received.

# *Feature 4 - Backup Functionality*

A timestamp is first created with the format “YYYY-MM-DD-HH-MM-SS” and is concatenated to the backup directory to form a new subdirectory within the backup directory. The fork and execute commands are then used to create a new process that will copy the contents of the current upload directory to the new backup subdirectory. A new subdirectory is created each time a backup occurs which gives users the options to choose from multiple backups should they need to.

While this occurs, a lock is placed on the reporting directory while the parent process waits until the copying process is complete. If an error occurs with the forking process an error will be logged. Similarly, a syslog entry will be written on completion of the backup to indicate whether it was a success or failure.

# *Feature 5 - Transfer Functionality*

The *transfer.txt* is first opened for writing while the fork command is used to execute the process as a new subprocess. This subprocess finds any changes that have occurred to the upload directory over the last 24 hours. These files are then copied over to the reporting directory. The parent process uses the wait command to wait for this process to finish before logging the succession of failure of the transfer process.

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

The lock and unlock directory functions use varying permission modes to restrict and open directories. The lock function uses the 0555 permission with the chmod command to restrict writing to the upload directory while backups and transfers are occurring but allows reading and execution of files in the directory. The unlock directory function uses the 0777 permission with the chmod function to allow reading, writing, and execution of files in the directory once backups and transfers have concluded.

The daemon flow of operation ensures the upload directory is locked before any backup or transferring of files begins to prevent users from writing or changing files while these functions are occurring. Once the backup and transfer of files has completed, the unlock function is called to allow files once again to be added, removed, or edited as pleased.

# *Feature 7 – Process management and IPC*

Interprocess Communication is achieved using the pipe command which forms a one-way data channel between processes. That is one process can write through the pipe while another process can read from it. It also allows for child processes to communicate with their parent process. Pipe is used in this daemon process to log messages that are created from a message queue and direct them to the *user.log* file which can then be read outlining what is occurring in the program, any errors that occur, or any changes to files.

The daemon process also makes extensive use of the fork and exec family of functions throughout to perform activities away from the main process. Examples include, copying files from the upload directory to the reporting directory, finding files that have been edited or created, and backing up files from the reporting directory to the backup directory.

# *Feature 8 - Logging and Error Logging*

Error checking is included throughout the daemon process and its associated C files. If statements are used to check for appropriate conditions throughout the process. Successes and failures cause the syslog to be opened and for the appropriate log message to be appended to it. The syslog command is used to track various events such as logging when tracking has begun on given directories, when directories become locked and unlocked, and when backups and transfers begin. Errors that are logged include errors when transferring and backing up files, signal interrupts being received, failure of the fork and exec commands, and failure to lock or unlock directories. The syslog and user.log can be checked for the various events described above.

The project also logs any changes to directory files and any transfers that take place in the *changes.txt* and *transfers.txt* files respectively in the logs directory.

# *Conclusion*

While the project was a challenge to get all the required functions working together smoothly, it was enjoyable to work on a project that modelled a real-world problem. This was a great opportunity to perform how one should when in a production environment and many of the challenges that arose were familiar to those that came up during the third-year work placement module.

The project was challenging in what it asked for but fair in the amount of work and effort required to come up with a working solution. It also really helped with learning the various topics in the Systems Software module such as creating and running daemon processes, IPC, usage of the fork and exec commands and the logging of errors. It is a great achievement to look back on the project now that it is complete and know that it is achievable to be able to complete work on a real-world solution.