Systems Software Report CA1

DT228
BSc in Computer Science

Eamonn Keogh
C16757629

School of Computer Science
TU Dublin – City Campus

15/03/2020

Table of Contents

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	Reporting (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: please paste link here

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:

_____

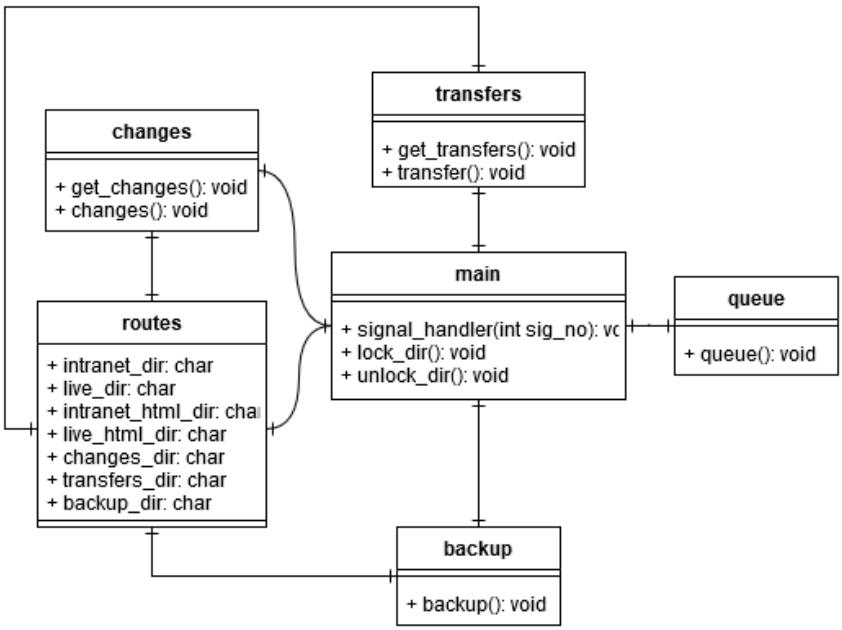
<Student Name>

<Date>

Feature 1 - System Architecture including makefile

Detailed description of the system architecture choices made.
How Separation of Concerns (SoC) and Single Responsibility Principle (SRP) was followed.

Architecture Diagram



For this project it made sense to separate each feature into a separate c file, as the reader can see from the above diagram. Each c file fulfils a single unit of responsibility. This makes the code modular and flexible.

Routes.c

This file contains the directories to the local intranet/live project folders for the web application as well as the hosted html files that get transferred to the apache2 server. It also contains the root admin directory for storing the changes and transfers as text files and the backup directory.

Changes.c

This file contains the logic to create the pipes needed for the daemon to write the file changes to the changes.txt file as well as monitor the modified files within the intranet folder for the web application.

Transfers.c

This file contains the details of how to copy the contents of the local web application html directory to the hosted web app directory of the apache2 server.

Queue.c

This file is responsible for the IPC log that is constantly monitoring any changes made, errors, successes received from the daemon process.

Backup.c

This file creates a new directory and creates a timestamp of when the backup took place as well as copying the contents of the local live web application directory.

Main.c

Contains the driver program and the creation of the daemon process as well as directory locking, unlocking and signal handling.

Feature 2 - Daemon (Setup/ Initialisation/ Management)

Detailed description of the daemon setup:
Startup Script and init process
Daemon control options

The daemon setup follows the following steps:

1. Create the orphan process by calling *fork()*
2. Elevate the orphan process to session leader, to lose controlling TTY by calling *setsid()*
3. Call *umask()* to set the file mode creation mask to 0. This will allow the daemon to read and write files with the permissions/access required
4. Change the current working directory to root by calling *chdir("/").* This will eliminate any issues of running on a mounted drive, that potentially could be removed etc...
5. Close all open file descriptors:
   for (int x = sysconf(_SC_OPEN_MAX); x >= 0; x--) { close(x); }

I also created a systemd service that automatically executes the daemon on server startup:

[Unit]
Description=Daemon that backups and logs users.

[Service]
Type=simple
ExecStart=/usr/bin/main
Restart=on-failure
RestartSec=10
KillMode=process

[Install]
WantedBy=multi-user.target

Feature 3 - Daemon (Implementation)

Detailed description of the process followed to create the background process.

To create the background process one can do it two different ways. One can fork the process just once and then call *setsid()* to ensure that the child process is disconnected from the controlling terminal. Or one can ensure completely that the daemon is running entirely in the background by forking twice and then calling the *setsid()*, *umask()* and closing file descriptors.

Feature 4 - Backup Functionality

Detailed description of the backup implementation

Create a timestamp in the form of Year:Month:Day:Hour:Minute:Second
Copy the local live web application project to the live project directory
Append the created timestamp to the backup directory
Try fork the process
    If forking failed
        Write error to log
    If child process
        Copy the local live directory archive to the backup folder
    If parent process
        Lock and wait for child process to finish
        If successful
            Write to queue that backup was completed and output to server log
        Else
            Write to queue that backup failed and output to server log

Feature 5 - Transfer Functionality

Detailed description of the transfer implementation

Open the transfer file for writing ensure its using the stdout file io
Fork the process to ensure it runs as a different subprocess
If child process
    Find file changes in intranet html directory for the previous 24 hours
If parent process
    Lock file and read data from the std file descriptor which contains the transfer file
    If process locked
        Write transfer success to queue and output success to server log
    If process unlocked
        Write transfer failure to queue and output failure to server log

Feature 6 - Lockdown folder for Backup / Transfer

Detailed description of the lockdown functionality/implementation

To get the locking to work during the backup and transfer I used various user mode permissions. To lock file during the backup process I used *chmod(0777)* during the update and backup stage this allows the user to read write and execute. Then once the update has completed the directory is locked by changing the permissions by using *chmod(0555)* which changes the permissions to allowing read, blocks writing and can execute.

It's important also to first lock the directory so other users can not access this process while the daemon writes the new data to the intranet directory. Next call the *backup(), transfer()* and once these functions are done then unlock the directory so users can make changes again.

Feature 7 - Reporting (IPC)

Detailed description of how child processes communicate success/failure of tack to be completed to parent process etc….

This is achieved through using the *pipe()* command in c which creates a unidirectional data channel that can be used for inter-process communication. This channel overrides the file descriptor with the stdout stdin and a log is created and the messages from a created queue are appened to this log that's constantly running on the server and can be accessed by navigating to /var/log/user.log

Feature 8 - Logging and Error Logging

Detailed description of the error and logging functionality included in the code solution.

The error and logging functionality is handled throughout the code base as checks are made with if conditions and based upon this conditions the log is opened and written to with custom messages based upon the event that occurred. For example, transfer success are the process has exited and the queue has been updated.

Conclusion

Summary of the implementation and achievement