Systems Software Report CA1

TU857
BSc in Computer Science

Omair Duadu
C18322011

School of Computer Science
TU Dublin – City Campus

02/11/2021

<u>**Table of Contents**</u>

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/plskpMMflJ4

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:

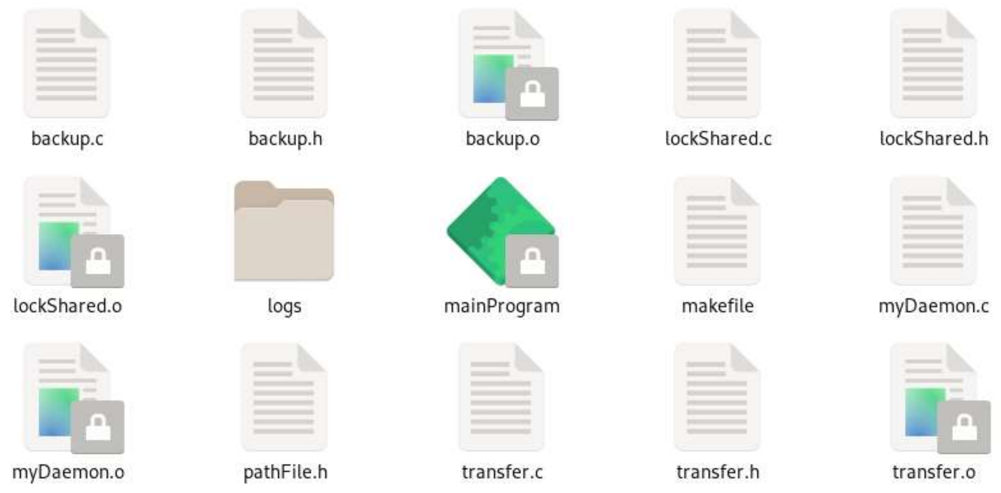
Omair Duadu

02/11/2021

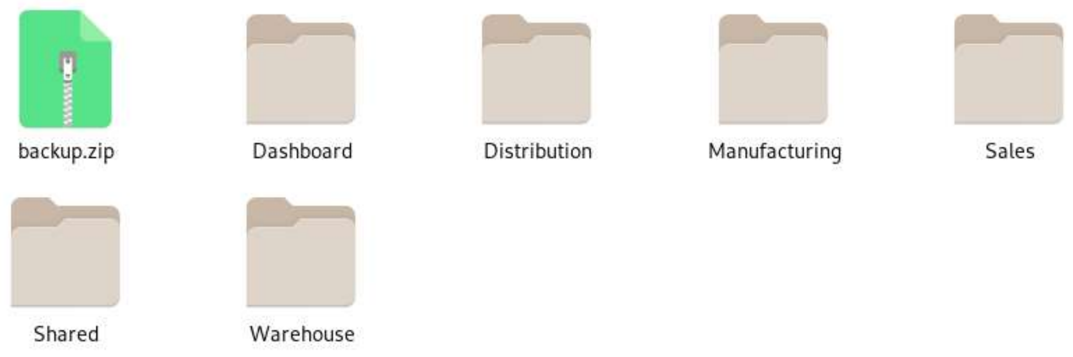
Feature 1 - System Architecture including makefile

The system I am making has 2 core functions which are transferring files from the Shared directory to the Dashboard directory, and then performing a backup of the Dashboard. The daemon is configured to run at 11.30pm every night so that there is minimal impact on the performance because will the transfer/backup is being completed access to these directories will be lost. The operating system I am using is a Debian Linux, which uses systemd to initialise this daemon.

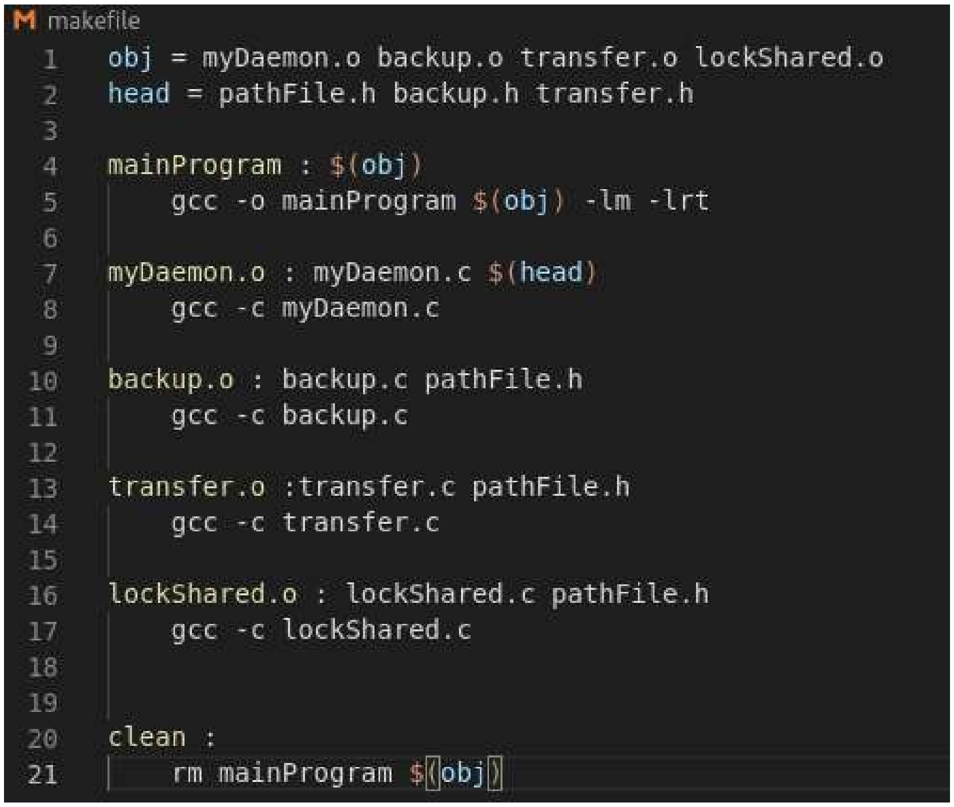
The detailed description of how the daemon works is that it uses a Singleton Pattern, which creates a child process, where an orphan is created and elevated to session leader, this is important so that it loses controlling TTY. From there umask() to gain all permissions required.
The next bit which is highly import is the logic runs in an infinite loop, it contains the lockShared(), transfer(), and backup() functionality.



The code files and log folder are all present in one directory, I have applied Separation of Concerns by making different files for all the functions rather than placing them all in one big code file.



All the different directories are placed in the Plant directory like you can see above, this includes the Shared, Dashboard and Backup folders.

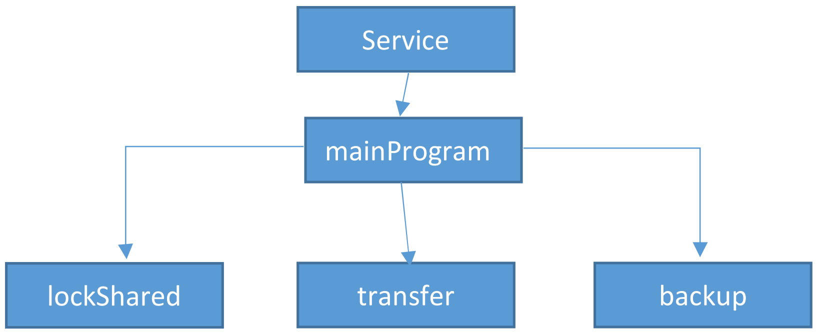


makefile

```
obj = myDaemon.o backup.o transfer.o lockShared.o
head = pathFile.h backup.h transfer.h

mainProgram : $(obj)
	gcc -o mainProgram $(obj) -lm -lrt

myDaemon.o : myDaemon.c $(head)
	gcc -c myDaemon.c

backup.o : backup.c pathFile.h
	gcc -c backup.c

transfer.o :transfer.c pathFile.h
	gcc -c transfer.c

lockShared.o : lockShared.c pathFile.h
	gcc -c lockShared.c


clean :
	rm mainProgram $(obj)
```

My makefile is able to quickly compile and run all the code I need which includes the object & header files with a single command: make!
Make clean also gives me the ability to remove all the object files to recompile

Architecture Diagram.



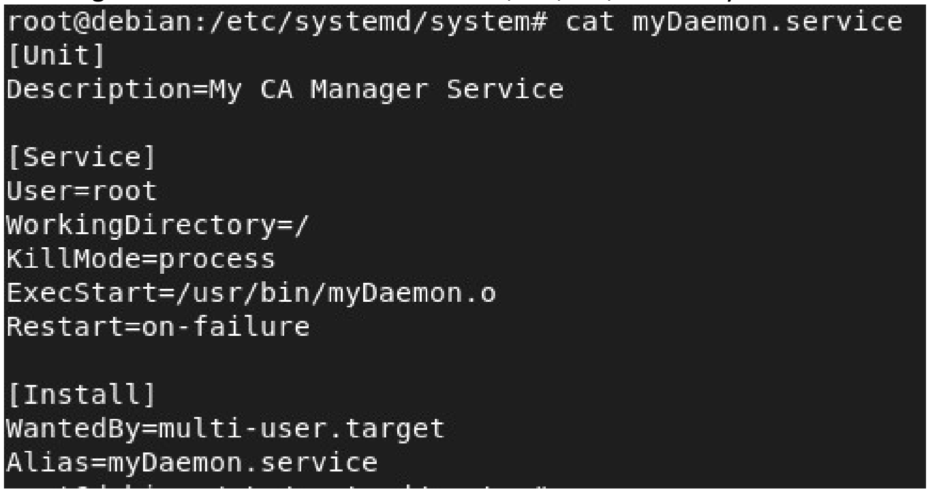
Feature 2 - Daemon (Setup/ Initialisation/ Management)

From the very beginning the first thing in the system is a systemd service called myDaemon.service. Systemd has many great functions and the 5 main ones for my use case are start, stop, status, enable and disable.

Sudo systemctl start/stop myDaemon.service can start/stop the service at will.
Sudo systemctl enable/disable myDaemon.service can start/stop the service at boot time, this essential automates so that even if the system shuts down it will still work the next time it turns on.

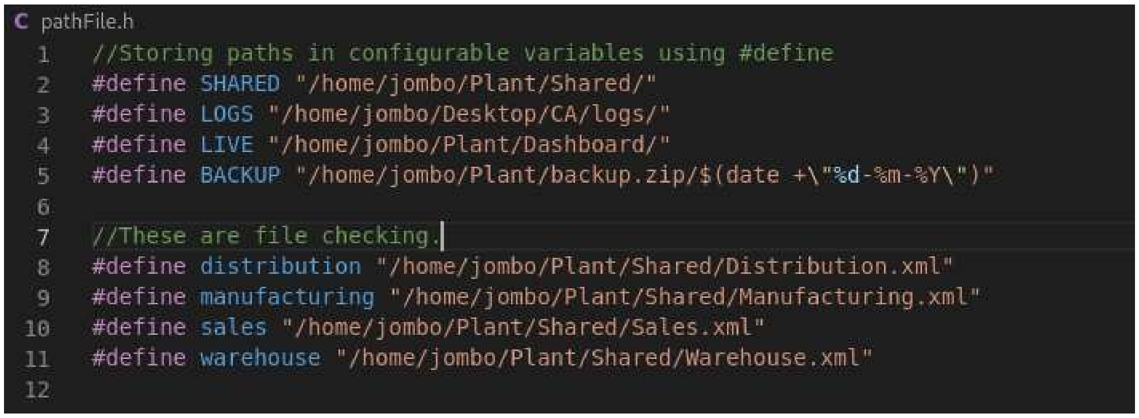
The myDaemon.service and similar service files are stored in the /etc/systemd/system directory. There are also other locations for service files with different purposes.

The target for the service is stored in the /usr/bin/ directory.



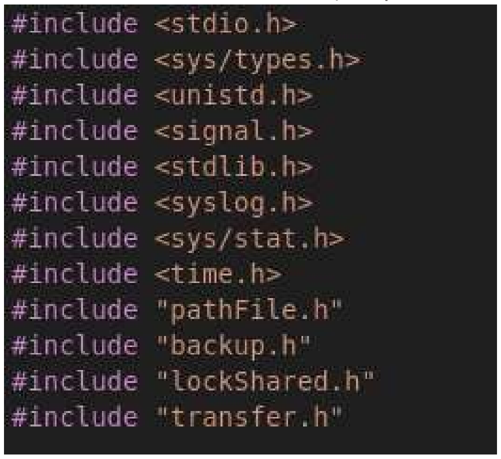
```
root@debian:/etc/systemd/system# cat myDaemon.service
[Unit]
Description=My CA Manager Service

[Service]
User=root
WorkingDirectory=/
KillMode=process
ExecStart=/usr/bin/myDaemon.o
Restart=on-failure

[Install]
WantedBy=multi-user.target
Alias=myDaemon.service
```

Part of my management and initialisation is creating a pathFile.h which holds defined paths which is used all across the different functions for uses.



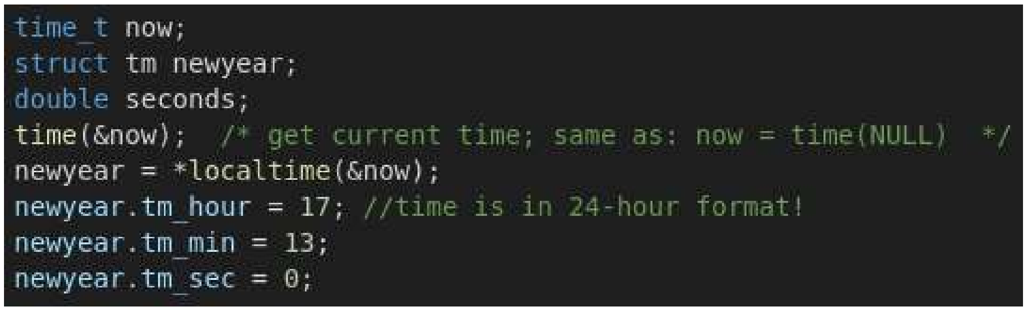
```c
//Storing paths in configurable variables using #define
#define SHARED "/home/jombo/Plant/Shared/"
#define LOGS "/home/jombo/Desktop/CA/logs/"
#define LIVE "/home/jombo/Plant/Dashboard/"
#define BACKUP "/home/jombo/Plant/backup.zip/$(date +\"%d-%m-%Y\")"

//These are file checking.
#define distribution "/home/jombo/Plant/Shared/Distribution.xml"
#define manufacturing "/home/jombo/Plant/Shared/Manufacturing.xml"
#define sales "/home/jombo/Plant/Shared/Sales.xml"
#define warehouse "/home/jombo/Plant/Shared/Warehouse.xml"
```

Feature 3 - Daemon (Implementation)



```
#include <stdio.h>
#include <sys/types.h>
#include <unistd.h>
#include <signal.h>
#include <stdlib.h>
#include <syslog.h>
#include <sys/stat.h>
#include <time.h>
#include "pathFile.h"
#include "backup.h"
#include "lockShared.h"
#include "transfer.h"
```

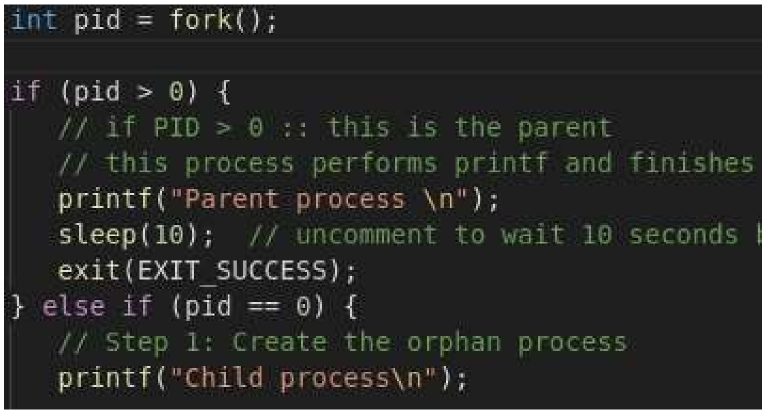
Including all the libraries and header files of the functions needed.



```
time_t now;
struct tm newyear;
double seconds;
time(&now);  /* get current time; same as: now = time(NULL)  */
newyear = *localtime(&now);
newyear.tm_hour = 17; //time is in 24-hour format!
newyear.tm_min = 13;
newyear.tm_sec = 0;
```

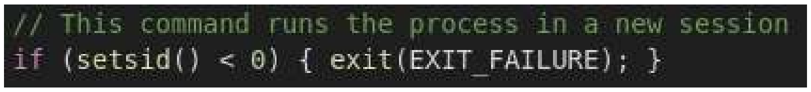
This is how the time is set for when the daemon will run.

PROCESS MANAGEMENT



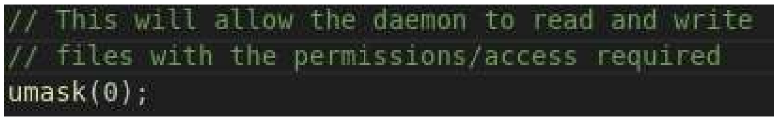
```
int pid = fork();

if (pid > 0) {
   // if PID > 0 :: this is the parent
   // this process performs printf and finishes
   printf("Parent process \n");
   sleep(10);  // uncomment to wait 10 seconds
   exit(EXIT_SUCCESS);
} else if (pid == 0) {
   // Step 1: Create the orphan process
   printf("Child process\n");
```

From the parent process create a child the orphan process.



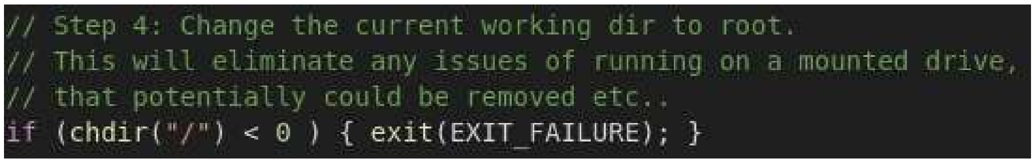
```
// This command runs the process in a new session
if (setsid() < 0) { exit(EXIT_FAILURE); }
```

Make the orphan the new session leader



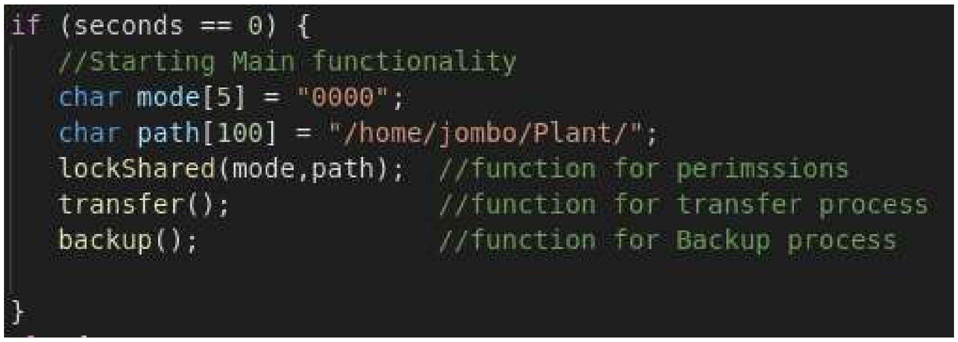
```
// This will allow the daemon to read and write
// files with the permissions/access required
umask(0);
```

Give the process all permissions that it requires



```
// Step 4: Change the current working dir to root.
// This will eliminate any issues of running on a mounted drive,
// that potentially could be removed etc..
if (chdir("/") < 0 ) { exit(EXIT_FAILURE); }
```

Go to the root directory to prevent problems.



```
if (seconds == 0) {
   //Starting Main functionality
   char mode[5] = "0000";
   char path[100] = "/home/jombo/Plant/";
   lockShared(mode,path);  //function for perimssions
   transfer();             //function for transfer process
   backup();               //function for Backup process

}
```

The last part of the Daemon is where it is ready to call the different functions.
There is other functionality in the daemon which is covered elsewhere.

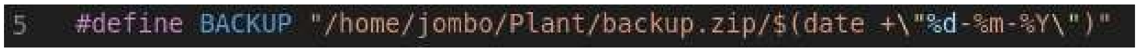
Feature 4 - Backup Functionality

Detailed description of the backup implementation

The backup function is called only after lockShared() which changes the permissions, this is covered in more detail in the *Feature 6*
The backup file code contains all the libraries and header files that the program needs are included.

All system commands are made through system() calls.
The backup has all the necessary logging through syslog.

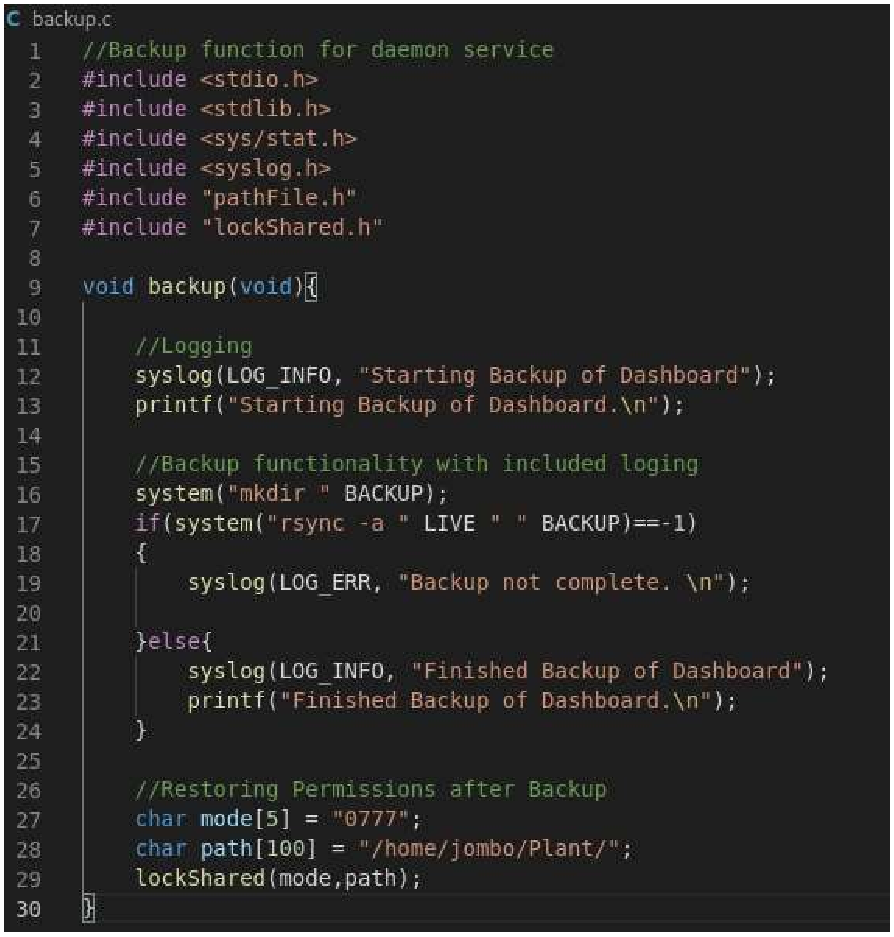
A new directory is created for every day in the backup.zip folder, using the mkdir command and BACKUP variable.



```
#define BACKUP "/home/jombo/Plant/backup.zip/$(date +\"%d-%m-%Y\")"
```

The backup is made by performing an “rsync -a” of the Dashboard directory to the backup location. The logging is created for a failure and success of the function.

After the backup the permissions are restored for all users.



```c
backup.c
//Backup function for daemon service
#include <stdio.h>
#include <stdlib.h>
#include <sys/stat.h>
#include <syslog.h>
#include "pathFile.h"
#include "lockShared.h"

void backup(void){

    //Logging
    syslog(LOG_INFO, "Starting Backup of Dashboard");
    printf("Starting Backup of Dashboard.\n");

    //Backup functionality with included loging
    system("mkdir " BACKUP);
    if(system("rsync -a " LIVE " " BACKUP)==-1)
    {
        syslog(LOG_ERR, "Backup not complete. \n");

    }else{
        syslog(LOG_INFO, "Finished Backup of Dashboard");
        printf("Finished Backup of Dashboard.\n");
    }

    //Restoring Permissions after Backup
    char mode[5] = "0777";
    char path[100] = "/home/jombo/Plant/";
    lockShared(mode,path);
}
```

Feature 5 - Transfer Functionality

Detailed description of the transfer implementation

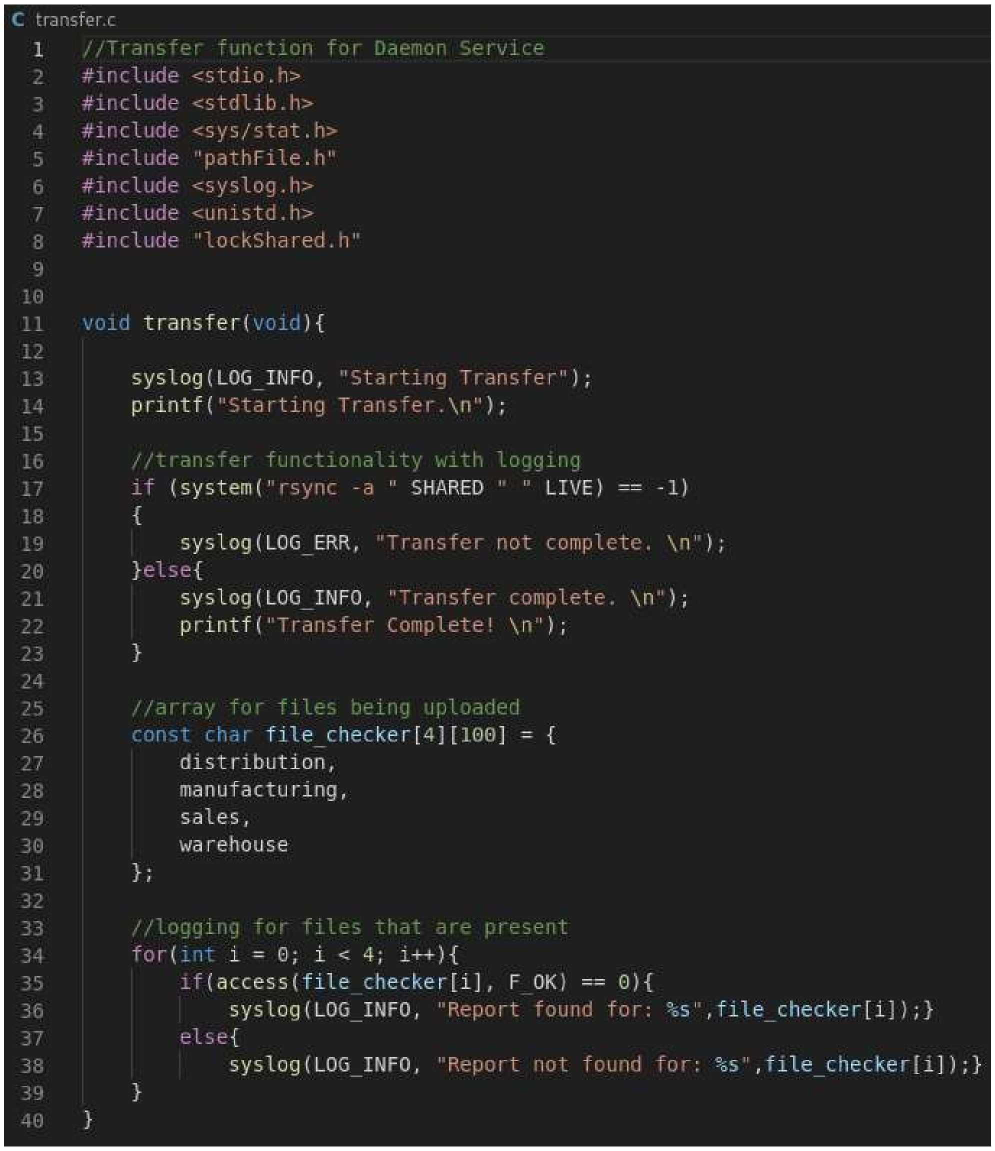
The backup function is called only after lockShared() which changes the permissions, this is covered in more detail in the *Feature 6*

The backup file code contains all the libraries and header files that the program needs are included.

All system commands are made through system() calls.
The backup has all the necessary logging through syslog.

The transfer functionality is very similar to the backup, except that the contents of Shared are moved to the Dashboard, and no new directory is created.

Part of transfer() is that it checks for if the managers have uploaded a new file the day, and this is done by making an array of path variables and comparing that to what is found in the Shared directory.


transfer.c
```c
//Transfer function for Daemon Service
#include <stdio.h>
#include <stdlib.h>
#include <sys/stat.h>
#include "pathFile.h"
#include <syslog.h>
#include <unistd.h>
#include "lockShared.h"


void transfer(void){

    syslog(LOG_INFO, "Starting Transfer");
    printf("Starting Transfer.\n");

    //transfer functionality with logging
    if (system("rsync -a " SHARED " " LIVE) == -1)
    {
        syslog(LOG_ERR, "Transfer not complete. \n");
    }else{
        syslog(LOG_INFO, "Transfer complete. \n");
        printf("Transfer Complete! \n");
    }

    //array for files being uploaded
    const char file_checker[4][100] = {
        distribution,
        manufacturing,
        sales,
        warehouse
    };

    //logging for files that are present
    for(int i = 0; i < 4; i++){
        if(access(file_checker[i], F_OK) == 0){
            syslog(LOG_INFO, "Report found for: %s",file_checker[i]);}
        else{
            syslog(LOG_INFO, "Report not found for: %s",file_checker[i]);}
    }
}
```

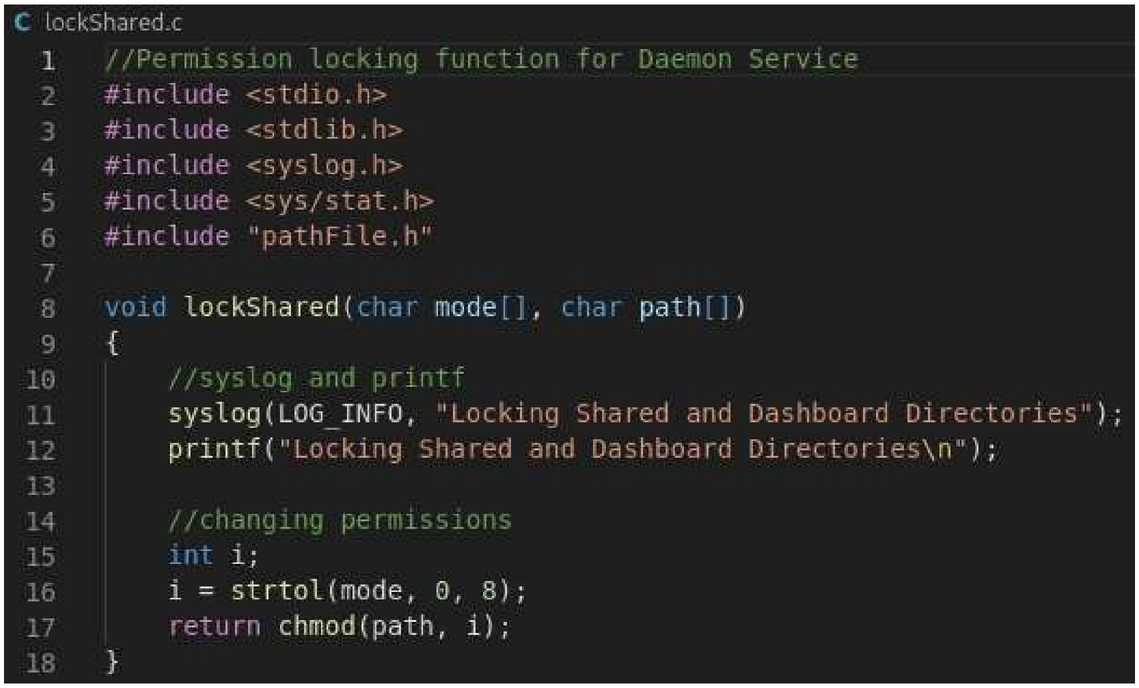
Feature 6 - Lockdown directories for Backup / Transfer

Detailed description of the lockdown functionality/implementation
The lockdown of permissions is the first function that is triggered in the main program, and also triggered in the backup function.

2 parameters need to be passed to the function which are mode[] for the permission level and path[] for the path where the permissions will be changed.

Chmod is the command for changing permission levels.

Chmod 0000 closes all permissions to all users except root, Chmod777 then will reopen all permissions.

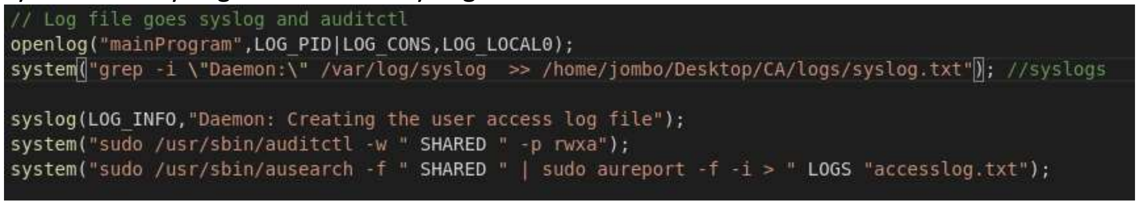


lockShared.c

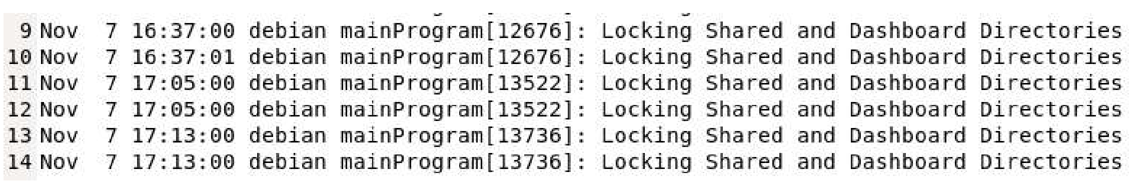
```c
//Permission locking function for Daemon Service
#include <stdio.h>
#include <stdlib.h>
#include <syslog.h>
#include <sys/stat.h>
#include "pathFile.h"

void lockShared(char mode[], char path[])
{
    //syslog and printf
    syslog(LOG_INFO, "Locking Shared and Dashboard Directories");
    printf("Locking Shared and Dashboard Directories\n");

    //changing permissions
    int i;
    i = strtol(mode, 0, 8);
    return chmod(path, i);
}
```

Feature 7 – Process management and IPC

Detailed description of how child processes communicate success/failure of tack to be completed to parent process etc….
I have achieved process management by being able to fork process and be able to observe the result of the process wherever it acts throughout the system.
This is achieved through applying abundant logging through syslog which is a very efficient system. The syslog was saved to syslog.txt.



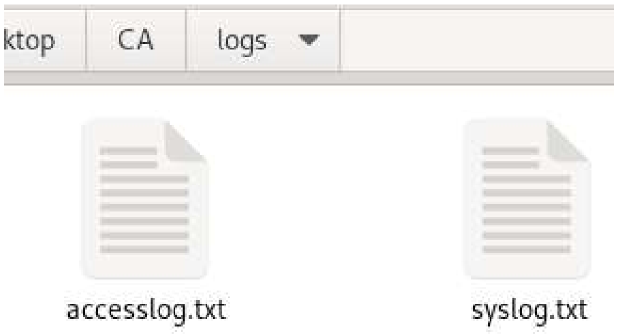
```c
// Log file goes syslog and auditctl
openlog("mainProgram",LOG_PID|LOG_CONS,LOG_LOCAL0);
system("grep -i \"Daemon:\" /var/log/syslog  >> /home/jombo/Desktop/CA/logs/syslog.txt"); //syslogs

syslog(LOG_INFO,"Daemon: Creating the user access log file");
system("sudo /usr/sbin/auditctl -w " SHARED " -p rwxa");
system("sudo /usr/sbin/ausearch -f " SHARED " | sudo aureport -f -i > " LOGS "accesslog.txt");
```



```
Nov  7 16:37:00 debian mainProgram[12676]: Locking Shared and Dashboard Directories
Nov  7 16:37:01 debian mainProgram[12676]: Locking Shared and Dashboard Directories
Nov  7 17:05:00 debian mainProgram[13522]: Locking Shared and Dashboard Directories
Nov  7 17:05:00 debian mainProgram[13522]: Locking Shared and Dashboard Directories
Nov  7 17:13:00 debian mainProgram[13736]: Locking Shared and Dashboard Directories
Nov  7 17:13:00 debian mainProgram[13736]: Locking Shared and Dashboard Directories
```

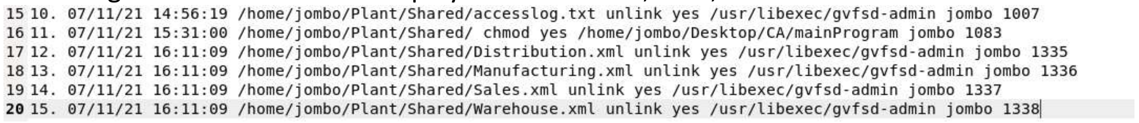
Feature 8 - Logging and Error Logging

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



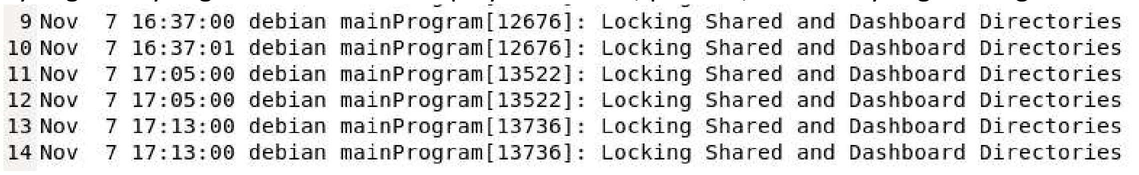
The logging is stored in one directory /logs/ accesslog.txt holds the user access logs, syslog.txt holds the syslog details.

Accesslog.txt looks like this its displays actions taken, user, and time.



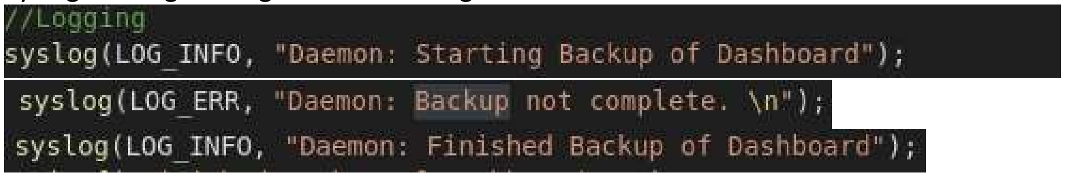
```
10. 07/11/21 14:56:19 /home/jombo/Plant/Shared/accesslog.txt unlink yes /usr/libexec/gvfsd-admin jombo 1007
11. 07/11/21 15:31:00 /home/jombo/Plant/Shared/ chmod yes /home/jombo/Desktop/CA/mainProgram jombo 1083
12. 07/11/21 16:11:09 /home/jombo/Plant/Shared/Distribution.xml unlink yes /usr/libexec/gvfsd-admin jombo 1335
13. 07/11/21 16:11:09 /home/jombo/Plant/Shared/Manufacturing.xml unlink yes /usr/libexec/gvfsd-admin jombo 1336
14. 07/11/21 16:11:09 /home/jombo/Plant/Shared/Sales.xml unlink yes /usr/libexec/gvfsd-admin jombo 1337
15. 07/11/21 16:11:09 /home/jombo/Plant/Shared/Warehouse.xml unlink yes /usr/libexec/gvfsd-admin jombo 1338
```

Syslogs.txt syslog looks like this it displays time user, process, and the syslog message



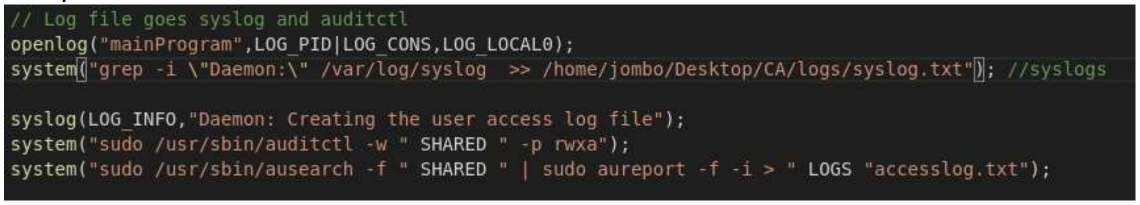
```
Nov  7 16:37:00 debian mainProgram[12676]: Locking Shared and Dashboard Directories
Nov  7 16:37:01 debian mainProgram[12676]: Locking Shared and Dashboard Directories
Nov  7 17:05:00 debian mainProgram[13522]: Locking Shared and Dashboard Directories
Nov  7 17:05:00 debian mainProgram[13522]: Locking Shared and Dashboard Directories
Nov  7 17:13:00 debian mainProgram[13736]: Locking Shared and Dashboard Directories
Nov  7 17:13:00 debian mainProgram[13736]: Locking Shared and Dashboard Directories
```

Syslog messages are generated through functions



```
//Logging
syslog(LOG_INFO, "Daemon: Starting Backup of Dashboard");
syslog(LOG_ERR, "Daemon: Backup not complete. \n");
syslog(LOG_INFO, "Daemon: Finished Backup of Dashboard");
```

Syslog is opened through the openlog.
/usr/sbin/auditctl records user actions into accesslog.txt from where they can be viewed easily



```
// Log file goes syslog and auditctl
openlog("mainProgram",LOG_PID|LOG_CONS,LOG_LOCAL0);
system("grep -i \"Daemon:\" /var/log/syslog  >> /home/jombo/Desktop/CA/logs/syslog.txt"); //syslogs

syslog(LOG_INFO,"Daemon: Creating the user access log file");
system("sudo /usr/sbin/auditctl -w " SHARED " -p rwxa");
system("sudo /usr/sbin/ausearch -f " SHARED " | sudo aureport -f -i > " LOGS "accesslog.txt");
```

Conclusion

Summary of the implementation and achievement
The Daemon is built like a lego set, each part which is created is interconnected with other pieces.
The main daemon has been added to the service files, giving many amazing functionality which is provided by systemd.
When the daemon it forks a process, it also has a countdown function which sets when the transfer and backup will occur. The forked orphan process will perform the permission locking, backup and transfer.
Logging is also present through logging for user actions by using auditd and syslogs which are activated throughout the functions.