**Documentaion of TCP/IP XML Logger**

***BASIC DESIGN FLOW***

As this is the TCP/IP logger I have designed the Logger by keeping in mind that this logger is intended to run on Servers. So, I started my designing with my **serve** class which has the definition in serve.h file and the declaration is in serve.cpp file. As this is the server class it receives input command from the client side after doing the TCP/IP 3 way hand shake. It is running on the localaddress “127.0.0.1” and the port number is “50000”. The reason I chose those two are because as that task was to run on local machine and port is the private port according to the RFC documents. Sever accepts commands from the nc and initiates the logger accordingly using threads.

Logger resides inside logger class which has the declaration and definition in logger.h and logger.cpp respectively. Logger reads the proc directory file for the assiged pid( process id ) and fetch out the asked details and writes them onto the .xml file.

Encryption and Decryption is also happening in this given task for which I gave declaration and definition in for encryption and description in enc\_dec.h and enc\_dec.cpp file. The input file for encryption and decryption is the current file, means todays date file. It reads the file and encrypt decrypt accordingly.

Whenever logger starts it creates the new file for the new day but if the logger runs again for the same day, in that case it deletes the old data and stores the new entries. For example, a person starts logger on 16/Jun/2017 at 10:00 am , in that case logger will create fresh files , stores all the fresh entries in the file and after sometime user stops that logger lets say at 11:00 am. And again after some time on the same day he/she runs it again like at 1:00 pm on the same day, in this case logger will not create the new file it just deletes the previous entries and stores the new entries.

Same goes for Encyription and Decryption files.

***NAMING CONVENTION FOR FILES***

All files generated by this logger resides inside the FILE directory, the file contins name in format-

**Log file** – *log\_dd/mm/yyyy.xml*

**Enryption file** - *ENClog\_dd/mm/yyyy.csv*

**Decryption file** - *DEClog\_dd/mm/yyy.xml*

***INSIDE LOOK***

In this section I am going to give my explanation on per classes basis -

***Header.h*** – This file contains all the standard header files which I have used in this task. I tried to follow concept of Abstraction as I could. Apart from header files it contains all the constant values which I have used in whole process. The reason I did this is because of portability like for example, what if anyone wants this logger to run on minute basis instead of hourly basis , instead of running through all the files , the person can directly change the desired setting from this file.

There are few things which I have used for error checking in my this program and a macro defined as ERROR. This macro has the default arguments which are

*funcname* – Function name in which error has occured

*line* - Line number where error has occured

*errnum* - Error number set by errno variable and I used to standard function which converts this

errno number to the standard string.

***enc\_dec.h/enc\_dec.cpp*** – These files have there dedicated purpose for Encryption and Decryption.

These file contains below described variables

ofname – contains the old file name this is the input file which we are reading and using for encryption and decryption as a input

*nfname* – stores new file name which we are creating after encryption and decryption

*ofd* - File descriptor used by standard linux system calls for doing operations on old file.

*nfd* - File descriptor used by sys calls for doing opeation on new file.

Methods described in these files are

*Close()* - This method closed all the open files in our case ofd and nfd.

*Encrypt()* - This method is dedicated to the Encryption operation. It receives the log file name as the input and creates the new file.

*Decrypt()* - This method is dedicated to the Decryption operation. It receives name of the file as an argument and generates one.

**logger.h/logger.cpp** – This file contains all the operations we are doing in logger file. Created the file genearated the entry in logger and stores it in the logger file. All operations are happening through threads. The varaible which are the part of this class are-

*buffer* – used to store data which we are fetching out from /proc directory.

*rawtime, timeinfo* – variables used to fetch out the current time and date info from the system.

*fp* – stores the file name which we have generated in logger one argument constuctor.

*fd* – fd is the file descriptor to store opeations on file.

*timetosleep* – this the variable which is used to make thread to sleep for the desired time, in our case 1 hour.

Methods used by this class -

*logger(std::string &)* - This is the one argument constructor which I have used to generate the file name and opening it through out the operation using fd. The argument stores the name of new file.

*MEMLogger()* - As name indicates it fetch out the details from the file and writes them onto the fd which is the file descriptor of the new file.

*CPULogger()* - This stores all the information related to cpu inside the same file I have opened earlier.

*End()* - This function will close the file which the logger was using for write operation.

*MEMThread()* - This is the firend type function which I have described below

*CPUThread()* - This is also the friend type function which I would be explaning in a moment

Apart from the class this file also contains some global variables, they are

*mut\_lock* – it is a mutex variable which I have initiated statically and it helped in syncronization pupose.

*write\_lock* – It is also the mutex variable initialized statically , this gives extra level of syncronization to the all the write operations happening on file.

Global Methods -

*MEMThread()* – This method is used by the thread which the server is going to start. It runs on the infinite loop and starts the MEMLogger() to make an entry in the file when the wait time comes. As this is used to initiate the class method that’s why this is a friend to *logger* class.

*CPUThread()* - Same as the above but used to run the CPULogger() routine from the class.

So, what is syncronization and why I needed that in first place ? Here is the answer, to avoid race conditions. As, we are creating only one object and two threads are working on the same time (assume) on the same file. What if a person starts both MEMLogger and CPULogger at the same time , meaning Logger will make 2 entries one for MEMORY and one for CPU at the same time on the same file. So this is the race condition right? To avoid overwritten of data or to avoid chaos I used mutexes.

***serve.h/serve.cpp* -** This file contains the follwing variables -

*server\_fd* – stores file descriptor value returned by the socket sys call.

*new\_socket* – stores file descriptor value returned by accept sys call.

*valread* – stores number of bytes read by read sys call from the client(nc) .

*address* – its a struct varable used to store the port , address information which we need to bind the socket and accept the connection for that address.

*buffer* – to store the string in our case the commands sent through client side.

*th* – its a thread array, used for creating the threads, in our case 2 threads.

*cur\_work\_file* – this is the variable which we pass along with the logger class object to retrieve the file name

*cpu\_log\_check* – checks if CPU logger is running or not.

*mem\_log\_check* – checks if MEM logger is running or not.

*log\_close\_check* – checks if the serve class has called any file close operations for logger object, in our case it happens in case of ENCRYPT and SHUDOWN.

*m\_repeat* – varaible used for our unit test case.

Apart from the above variables this class also contains some predefined strings which I am using for the messages send through our server to client(nc).

Methods described in this class -

*CheckCase(int &)* - This method is called from our unit test case class and used to store the result in the method argument to check if the things are working right or not.

*Sock()* - This is the most important method in this program, it does the following things -

i. Create socket using sock system call.

ii. inet\_pton to convert the address in network address form.

iii. Setsockopt , it is used to forcefully get the port , if it is alrady in use.

iv. Bind binds all the information on the address specified in address struct variable.

v. Listen indicates that the server is ready for the listen

vi. Accept accpets the connection from the client side.

And after that goes on infinite loop which works untill person sends the SHUTDOWN command or asks for Encryption(ENCRYPT command ).

Commands listen by the server-

**START\_MEM\_LOGGER** – starts the memory logger from logger class by creating thread, if logger is already running , intimate the user the same. Flow is defined below

*START\_MEM\_LOGGER -> Creates thread -> MEMThread -> MEMLogger*.

**STOP\_MEM\_LOGGER** – stops the memory logger, detach all the resources initiated by thread , if the logger is already stopped , intimate the same to user.

**START\_CPU\_LOGGER** – same as **START\_MEM\_LOGGER** but in this case it starts the CPU logger, if in case is logger is already running, intimate user the same. Flow is

*START\_CPU\_LOGGER -> Create thread -> CPUThread -> CPULogger*

**STOP\_CPU\_LOGGER** – stops the memory logger if the logger is already stopped , initimate same to the user.

**ENCRYPT** – Encrypt will close if any of the logger is running, close the logger file. And use the file name and done the encryption.

*Case 1* – If loggers are up and running running

*STOP\_MEM\_LOGGER -> STOP\_CPU\_LOGGER -> End() -> Encrypt() -> Stop the process*

*Case 2* – if no loggers are running

*Encrypt() -> Stop the process*

*End()* - method from logger class.

*Encrypt()* - method from logger class.

The reason why I stop the process( in our case logger program ) as all the resources are closed , as we are working on the current file, so we need it first close for further work, so it does not make any sense to run logger in background as it would not be doing anything.

**DECRYPT** – it does not require to stop the process as it is going to work on Encrypted file not on the logger file.

**SHUTDOWN** – It will shutdown all the resources, in our case thread, close the file, in our case logger file, and sends the intimation to user that process has been shuted down.

***TestCase.h***– This file contains the test case unit which I have used our server. These is only test case as of now which checks that if the value is not zero inside TestStraightAll and assert if something wrong occured.

Test case I am generating its on serve class object.

***main.cpp* –** main starts the test case and out server and gives us the warnign message if something bad happened.

***TOOLS AND EXTRA LIBRARIES***

*CppUnit* – This is the tool which I have used for unit test case.

*Pthread* - Library which I have used to run threads.

***DOWNLOAD LINK FOR CPPUnit***

*sudo apt-get install libcppunit-doc libcppunit-dev*