[2] http://softpixel.com/~cwright/programming/threads/threads.c.php

[4] https://computing.llnl.gov/tutorials/pthreads/

[5] https://sites.google.com/a/msrg.utoronto.ca/ece297/tas-and-labs/concurrent-connections-with-threads

[6] http://koti.mbnet.fi/niclasw/MutexSemaphore.html

Picture1 : <https://computing.llnl.gov/tutorials/pthreads/>

PICTURE 2: <https://sites.google.com/a/msrg.utoronto.ca/ece297/tas-and-labs/concurrent-connections-with-threads>

PICTURE 3: <http://www.cs.cf.ac.uk/Dave/C/node29.html#SECTION002911000000000000000>

PICTURE 4: <https://sites.google.com/a/msrg.utoronto.ca/ece297/tas-and-labs/concurrent-connections-with-threads>

PICTURE 5: <http://www.cs.cf.ac.uk/Dave/C/node29.html#SECTION002911000000000000000>

PICTURE 6: <http://www.javamex.com/tutorials/threads/how_threads_work.shtml>

PICTURE 7: <http://perfsuite.ncsa.illinois.edu/libpshwpc/>

For this milestone the main task is to modify the server such that it can support multiple clients accessing the server at the same time. Earlier only one client could access the server at a time and for another client to access the server the first client had to disconnect from it. Our team aims to modify our server such that it will be able to support at least 10 simultaneous connections from the client.

PICTURE 1

In order to design such a server, a library POSIX Threads (Pthreads) has to be referred. Using the library, threads can be created and modified, each thread handling a client connected to the server. In this server acts as the shared memory that can be used by many people either sequentially or simultaneously [5].

Thread is an independent set of commands that run at the same time as the server. Using threads as a concurrency mechanism, allows the programmer to divide program into smaller pieces that can be operated and managed independently. Threads allow the different processes of the same program to run simultaneously and efficiently on multiple CPU’s [2].

A process are the set of instructions given by the operating system that are specific to each program code. These instructions are the execution path that make sure that each program runs accordingly. Each instruction is a part of thread, thus making threads independent. Every process has their unique Process ID (PID), code and data (global variables). A process can have multiple threads each having their particular thread ID, stack and program counter [5].

Threads have several advantages over multi - processes.

|  |  |
| --- | --- |
| Factor | Comparison |
| Speed | Being part of the same address space in a process allows threads to:   1. Link fast between threads. 2. Take less time to create. 3. Share everything, making an effective communication. |
| Time |
| Communication |
| Work | Threads carry out less work as they are used for performing small tasks in a process, whereas process undergo heavy loads of work. |

However threads can create problems for the commands ‘storage\_get’ and ‘storage\_set’ happening to the same index at the same time. It is difficult for the process to maintain the thread that is created, accesses the hash table or is terminated first. This can corrupt data stored in the index. Hence access to shared data is synchronized to make sure that consistency is maintained for all the clients. This restricts “race conditions” errors that might occur due to uneven timing [5].

This can be done by using mutex(mutual extension) and semaphore. Mutex is an object that can be locked using a thread at a time. If second thread tries to lock the same mutex, an error will be print. This prevents different threads accessing the same mutex at a similar time. The other way to tackle the overlapping of threads in the same index is using semaphore. Semaphore being exact same to mutex counts in lieu of locks. When the count reaches zero then semaphore blocks the access of a thread. The semaphore will keep the access to that data blocked until another thread in the process increments the count. There are two parameters used in semaphore: ‘uping’ (increases the count) and downing (decreases the count) [2].

In this storage server development our team aims to use mutex for avoiding race conditions. Mutex is secure as it is sequential access for threads to the similar data whereas few threads can access the same data in a semaphore, increasing the risk of storing wrong data [6]. On the other hand mutex is easy to implement compared to semaphore as mutex requires only a boolean function, which when used by first thread on a data is set to true and when another thread tries accessing the same data, finds connections is true and is not allowed to connect until the first thread returns and makes it false, whereas for semaphore tracking every thread’s work on the same data and keeping track of count can be relatively difficult.

Until milestone 3, there were two log files created one for client and another for server, when LOGGING was set to 2. For this milestone two log files will be created per thread, making it easy for us to debug the code. Whenever a new client uses the server, separate log file will be created for that thread. Each log file will differ from its thread ID(unique to every thread).

Whenever a new thread is formed counter is incremented. The threads can be tracked using track ID and counter. Counter tells the number of threads running at the same time. Threads can be implemented using “pthread\_create” function. On every call to this function a new thread is formed with the older thread already running synchronously. Once a function is created, it will pass through suspend() function where the path of thread is decided. If the new thread tries to access the data that is currently being handled by the old thread, the thread will terminate otherwise server allows thread to make fresh changes or access any data in the value of an index. A thread is terminated at the end of “pthread\_create” function operation or earlier through ‘return’ in case of any errors.