Multithreading in C++

Multithreading is a specialised form of multitasking and multitasking is the feature that allows your computer to run two or more programs concurrently. In general, there are two types of multitasking process based and thread based.
Process based: handles concurrent execution of different programs
Thread based: handles concurrent execution of different pieces of same programs

Creating Threads:
Header file to be include:
#include <pthread.h>

```
#include <iostream>
#include <cstdlib>
```

Creating thread:
pthread _create (thread, attr, start_routine, arg)

Description about various arguments used above:
pthread_create : create a new thread and make it executable. This routine can be called any number of times from anywhere within your code.
attr : used to set thread attributes (can specify any attribute object or NULL for default values)
start_routine : that the thread will execute once it is created. This function should return void *
And void * type argument should be passed in this function.
arg : single argument that may be passed to start_routine (it must be a passed as reference as a pointer cast of type void (void *). NULL may be used if no argument is needed to pass.

The maximum number of threads that may be created by a process is implementation dependent. Once created threads are peers and may create other threads. There is no hierarchy or dependency between threads.

Terminating a thread:
pthread_exit (status) : status will be NULL for default
If main() finishes before the thread it has created, and exits with pthread_exit(), the other thread will continue to execute. Otherwise they will be automatically terminated when main() finishes.

Passing arguments to Threads:
Making a struct for message and sequence no. of client:

```
struct thread_data {
   int  thread_id;
   char *message;
};
```

Start-routine function:

```
void *PrintHello(void *threadarg) {
   struct thread_data *my_data;
   my_data = (struct thread_data *) threadarg;

   cout << "Thread ID : " << my_data->thread_id ;
   cout << " Message : " << my_data->message << endl;

   pthread_exit(NULL);
}
```

Main function;

```
int main () {
   pthread_t threads[NUM_THREADS];
   struct thread_data td[NUM_THREADS];
   int rc;
   int i;

   for( i = 0; i < NUM_THREADS; i++ ) {
      cout <<"main() : creating thread, " << i << endl;
      td[i].thread_id = i;
      td[i].message = "This is message";
      rc = pthread_create(&threads[i], NULL, PrintHello, (void
*)&td[i]);

      if (rc) {
         cout << "Error:unable to create thread," << rc << endl;
         exit(-1);
      }
   }
   pthread_exit(NULL);
}
```

Joining and detaching the thread:

Headers to be include:

```
#include <iostream>
#include <cstdlib>
#include <pthread.h>
#include <unistd.h>
```

pthread_join(threadid, status) : blocks the calling thread until the specified 'threadid' thread terminates.

pthread_detach(threadid) : to declare thread as detachable, or it can never be joined.

When a thread is created, one of its attributes defines whether it is joinable or detached. Only threads that are created as joinable can be joined. If a thread is created as detached, it can never be joined.

Initialize and set thread joinable:
```
pthread_attr_init(&attr);
pthread_attr_setdetachstate(&attr, PTHREAD_CREATE_JOINABLE);
```

Free attribute and wait for the other threads:
```
pthread_attr_destroy(&attr);
```