# 2803ICT Assignment 2

Due Wednesday 7 October at 11.59pm,
Worth 30% of final grade

## Objectives:

The objective of this assignment is to write a multithreaded client server system for multiprocessing. This requires putting into practice what has been taught about Multithreading, IPC, and synchronisation.

## Requirements:

1. The program will consist of a multi-threaded server and single- or multi-threaded client process.

2. The client will query the user for 32-bit integers to be processed and will pass each request to the server to process and will immediately request the user for more input numbers or 'quit' to quit.

3. The server will start up as many threads as there are binary digits × the max number of queries (i.e. 320 threads). The server will take each input number (unsigned long) and create 32 numbers to be factorised from it. Each thread will be responsible for factorising an integer derived from the input number that is rotated right by a different number of bits.

Given an input number K, each thread #X will factorise K rotated right by increasing number of bits. For example, thread #0 will factorise the number K rotated right by 0 bits, thread #1 will factorise K rotated right by 1 bit, thread # 2 will factorise K rotated right by 2 bits etc.

4. The trial division method should be used for integer factorisation.

5. The server must handle up to 10 simultaneous requests without blocking.

6. The client is non-blocking. Up to 10 server responses may be outstanding at any time, if the user makes a request while 10 are outstanding, the client will warn the user that the system is busy.

7. The client will immediately report any responses from the server and in the case of the completion of a response to a query, the time taken for the server to respond to that query.

8. The client and server will communicate using shared memory. The client will write data for the server to a shared 32-bit variable called 'number'. The server will write data for the client to a shared array of 32-bit variables called a 'slot' that is 10 elements long. Each element in the array (slot) will correspond to an individual client query so only a maximum of 10 queries can be outstanding at any time. This means that any subsequent queries will be blocked until one of the 10 outstanding queries completes, at which times its slot can be reused by the server for its response to the new query.

9. Since the client and server use shared memory to communicate a handshaking protocol is required to ensure that the data gets properly transferred. The server and client need to know when data is available to be read and data waiting to be read must not be overwritten by new data until it has been read. For this purpose, some shared variables are needed for signalling the state of data: char clientflag and char serverflag[10] (one for each query response/slot). The protocol operation is:

- Both are initially 0 meaning that there is no new data available

- A client can only write data to ‘number’ for the server while clientflag == 0; the client must set clientflag = 1 to indicate to the server that new data is available for it to read
- The server will only read data from ‘number’ from the client if there is a free slot and if clientflag ==1. It will then write the index of the slot that will be used for the request back to ‘number’ and set clientflag = 0 to indicate that the request has been accepted.
- A server can only write data to slot x for the client while serverflag[x] == 0; the server must set serverflag[x] = 1 to indicate to the client that new data is available for it to read.
- The client will only read data from slot x if serverflag[x] ==1 and will set serverflag[x] = 0 to indicate that the data has been read from slot x.

10. The server will not buffer factors but each thread will pass any factors as they are found one by one back to the client. Since the server may be processing multiple requests, each time a factor is found it should be written to the correct slot so the client can identify which request it belongs to. The slot used by the server for responding to its request will be identified to the client at the time the request is accepted by the server through the shared ‘number’ variable.

11. Since many threads will be trying to write factors to the appropriate slot for the client simultaneously you will need to synchronise the thread’s access to the shared memory slots so that no factors are lost. You will need to write a semaphore class using pthread mutexes and condition variables to used for controlling access to the shared memory so that data is not lost.

12. While not processing a user request or there has been no server response for 500 milliseconds, the client should display a progress update messages for each outstanding request (repeating every 500ms until there is a server response or new user request). The repeated progress message should be displayed in a single row of text. The message should be in a format similar to: > Progress: Query 1: X% Query2: Y% Query3: Z%

If you want you can use little horizontal progress bars i.e.
> Progress: Q1:50% ▒▒▒▒▒▒▒▒_____| Q2:30% ▒▒▒▒_____| Q3: 80% ▒▒▒▒▒▒▒▒▒▒▒▒▒▒__|

13. When the server has finished factorising all the variations of an input number for a query it will return an appropriate message to the client so that it can calculate the correct response time and alert the user that all the factors have been found.

**Notes:**

A. if your threads are too fast you may want to use a 10 millisecond delay in your loops
B. To be able to run shared memory and semaphores using **Cygwin**, we need to do steps below only one time to configure the **cygserver**

1- run Cygwin as administrator

2- type (cygserver-config) then accepts by typing (yes)

3- start the service by typing (cygrunsrv -S cygserver)

## Submission:

• Source code (include all makefiles, project files, project subdirectories; except the object files)
• Software documentation must be submitted according to the sample documentation that is available on learning@griffith. Same as for assignment 1.

## Marking Scheme:

| | |
|---|---|
| 1. Multithreaded server implementation | 10 |
| 3. Correct factorisation | 10 |
| 5. Non blocking server | 10 |
| 6. Non blocking client | 10 |
| 8. Shared memory use as specified | 5 |
| 9. Client – server synchronisation | 5 |
| 10. Non buffered server output | 5 |
| 11a. Semaphore implementation (non-busy waiting) | 5 |
| 11b. Correct server thread synchronisation | 5 |
| 12. Progress reporting | 10 |
| 13. Results display | 10 |
| Documentation | 10 |
| Code Style – Completeness – Robust - Quality | 5 |