2803ICT – Assignment 2

Ellis Rourke

Contents

1. Problem Statement

(In this section you write a few sentences about what needed to be done for the assigment)

The goal of this project is to create a client server program using C. The program uses shared memory to allow a client and server to communicate. The server accepts inputs from the client and used multithreading to calculate the numbers factors..

Further to this; the server also creates 32 threads and calculates the factors of all of the number represented by converting the origional number to binary and shifting the bits (essentiall multiplying)

2. User Requirements

- User should be able to pass values to the server from the client using shared memory
- User should be able to pass multiple values at once using command line arguments
- Server should correctly factorise number
- Server should pass values to the user (client) as they are found without interrupting factorisation or overwriting values in shared memory
- Server should check if shared memory is in use before writing to it
- If shared memory is in use, wait until free

3. Software 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:

    a. Both are initially 0 meaning that there is no new data available
    b. 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
    c. 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.
    d. 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.
    e. 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

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.

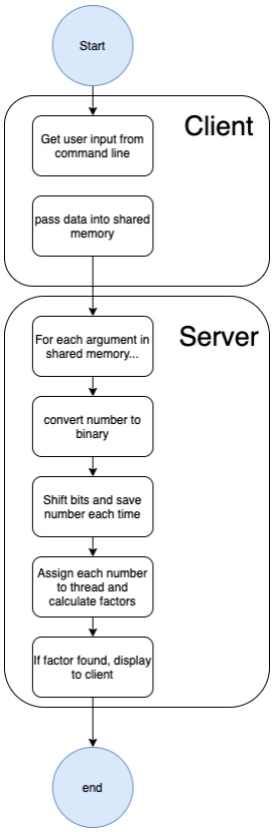
11. Since many threads will be trying to write factors to the appropriate slot for the client simultaneously you will need to synchronise the thread’s access to the shared memory slots so that no factors are lost. You will need to write a semaphore class using pthread mutexes and condition variables to use 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%

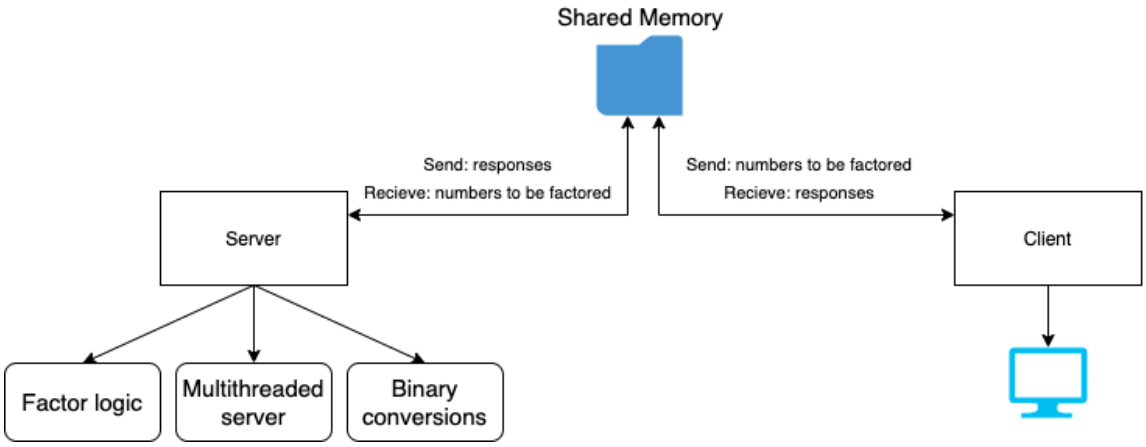
13. When the server has finished factorising all the variations of an input number for a query it will return an appropriate message to the client so that it can calculate the correct response time and alert the user that all the factors have been found.

Software Design

High Level Design – Logical Block Diagram



High Level Design – Structure Chart



List of all functions in the software.

For each function in the list the following information is provided:

- a brief description of what it does (1 or 2 sentences);
- a list of the input parameters, and their data types, and what they are used for;
- a list of any side effects caused by the function (ie change global or member variables, changes data passed by reference from calling function etc)
- a description of the function's return value

Name	Description	Parameters	Return
convertToBinary	Takes integer value and returns binary representation as an intger array where each index represents 1 bit of a 32 bit number	Int number	Int binary[32]
BinaryToDecimal	Sister function to convertToBinary… takes as input an array of integers representing a binary number and converts to its binary representation	Int number[]	Int value
arrayShift	Takes as input a 32 bit binary representation and circularly shifts the bits	Int array[]	Int array[]
Factorise	Multithreading function, takes as input a struct of data relevant to that thread, calculates all factors of input number and puts them in shared memory.	Void* arguments	Void*

14. Requirement Acceptance Tests

Software Requirement No	Test	Implemented (Full /Partial/ None)	Test Results (Pass/ Fail)	Comments (for partial implementation or failed test results)
1	Program has multithreaded server and single threaded client	Full	Pass	
2	Client will query user for 32 bit integers	Partial	Pass	Client accepts integers from command line
3	Client starts as many threads as needed	Full	Pass	
4	Trial Division used	Full	Pass	
5	Server must handle 10 requests	Full	Pass	
6	Client is non blocking	None	Fail	Command line arguments passed
7	Client immediatley reports responses	Full	Pass	
9	Handshaking protocol	Partial	Pass	Server waits to send data to client if shared memory variable is in use
10	Non buffering	Partial	Pass	Data is not passed into slots, rather just passed to a single variable for results being passed back to the client
11	Semaphore	None	Fail	
12	Progress Displayed	None	Fail	
13	Time and status displayed when complete	Full	Pass	

15. Detailed Software Testing

No	Test	Expected Results	Actual Results
1	No arguments passed to client	Client alerts user to supply arguments	As expected
2	Too many arguments passed	Client only accepts valid amount	Seg fault
3	0 is attempted to be factored	Client does not allow division by 0	As expected
4	Shared memory not connected correctly	Client or server allerts user and exits	As expected
5	Multiple threads trying to pass values to shared memory at one time	Prints in order	Some values missing

16. User Instructions

(Here you state what a user needs to do to run and use your program(s), this also includes and special instructures on how to compile the program(s) as necessary)

Compilation instructions:

Client:

Gcc client.c -o client x y z ….
./client

(Where x y z are n optional arguments) *must supply atlease 1

Server:

Gcc server.c -o server
./server