

3420ICT Assignment 1 – Option B

s2951339 Thomas McDonald

Contents

3420ICT Assignment 1 – Option B	1
1. Problem Statement	1
2. User Requirements	2
3. Software Requirements	2
4. Function Descriptions	4
5. High Level Design	5
.....	5
6. Requirement Acceptance Tests	6
7. User Instructions	12
Compiling the Server/Client	12
Running the Server	12
Running the Client	12
Commands.....	12

1. Problem Statement

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.

2. User Requirements

The following outlines the user requirements for the program:

- The user should be able to input a number above 0, and have the factorised results returned back, along with the factorised results of each rotation of that number up to 32 bits.
- If the user inputs a 0 it will activate the servers test mode and the server will return 0-29 to the client using 3 sets of threads.
- If the user inputs 'q' it will close the client and server and clean any shared memory.

3. Software Requirements

The following outlines the software requirements for the program:

1. The program will consist of a multithreaded server and single 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 'q' to quit.
3. The server will take each input number (unsigned long) and will start up either the number of specified threads if given (see Req.17) or as many threads as there are binary digits (i.e. 32 threads). 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 input number is K, each thread #X will factorise K rotated right by (K-1) bits. For example say K and it has N significant bits, then 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. Rotating an integer K by B bits = $(K \gg B) | (K \ll 32 - B)$. **CLARIFICATION: $C = K \ll (32 - B)$; Rotated = $(K \gg B) | C$**
4. The trial division method should be used for integer factorisation.
5. The server is must handle up to 10 simultaneous requests without blocking.
6. The client is non-blocking. A 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. Each factorising thread will report its progress as a percentage in increments not larger than 5% as it proceeds to the server primary thread. The server will use the individual thread progress values to calculate an overall progress % for each request being processed and pass this back to the client in increments not larger than 5%. This will require a second shared array **char progress[10]** that is 10 elements long but no handshaking protocol or synchronisation since it does not matter if the client misses some values or reads them multiple times.
 13. 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 that does not scroll up (see example lab 3). 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 ie
> Progress: Q1:50% ██████████ | Q2:30% ████████ | Q3: 80% ██████████ |
A sample session with only a single query may look like this (the example is contrived)....
 - Enter numbers to factor:
12345678912345
factors: 13 17
Progress: Query 1: 30% ████████ |
factors: 333 1234
Query complete for 12345678912345
 14. 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.
 15. The system will have a test mode that will be activated by the user entering 0 as the number to be factored. This will be passed to the server which will launch 3 sets of 10 threads each, each set will simulate one of three user requests. Each thread in each set will return to the client 10 numbers starting with the thread # times 10 and incrementing by 1. For example thread #0 will return numbers 0-9, thread #1 will return numbers 10-19, thread #2 returns numbers 20-29. Progress output will be disabled during the test and it will only run if the server is not processing any outstanding requests instead a warning will be issued to the user.
 16. If the client is terminated by the user entering 'q' or by typing Ctrl-C (cygwin only) it will first cleanly shutdown the server and then print out a nice termination message.
 17. The server will have a dispatch queue driven thread pool architecture where the thread pool size will be configured at startup by means of a command line argument

18. The source code shall be portable so that it can be compiled and run on Unix and Visual Studio (use can use the built-in win32 semaphore).

4. Function Descriptions

Client

double time_diff(struct timeval x, struct timeval y)
// calculates the difference between two given timesA

int kbhit()
//Keypress detection

void nonblock(int state)
// Allow non blocking reading

void displayProgress(Query x[], int len, int width)
//Display progress is given a query percentage and it will print out that percentage out in a progress bar

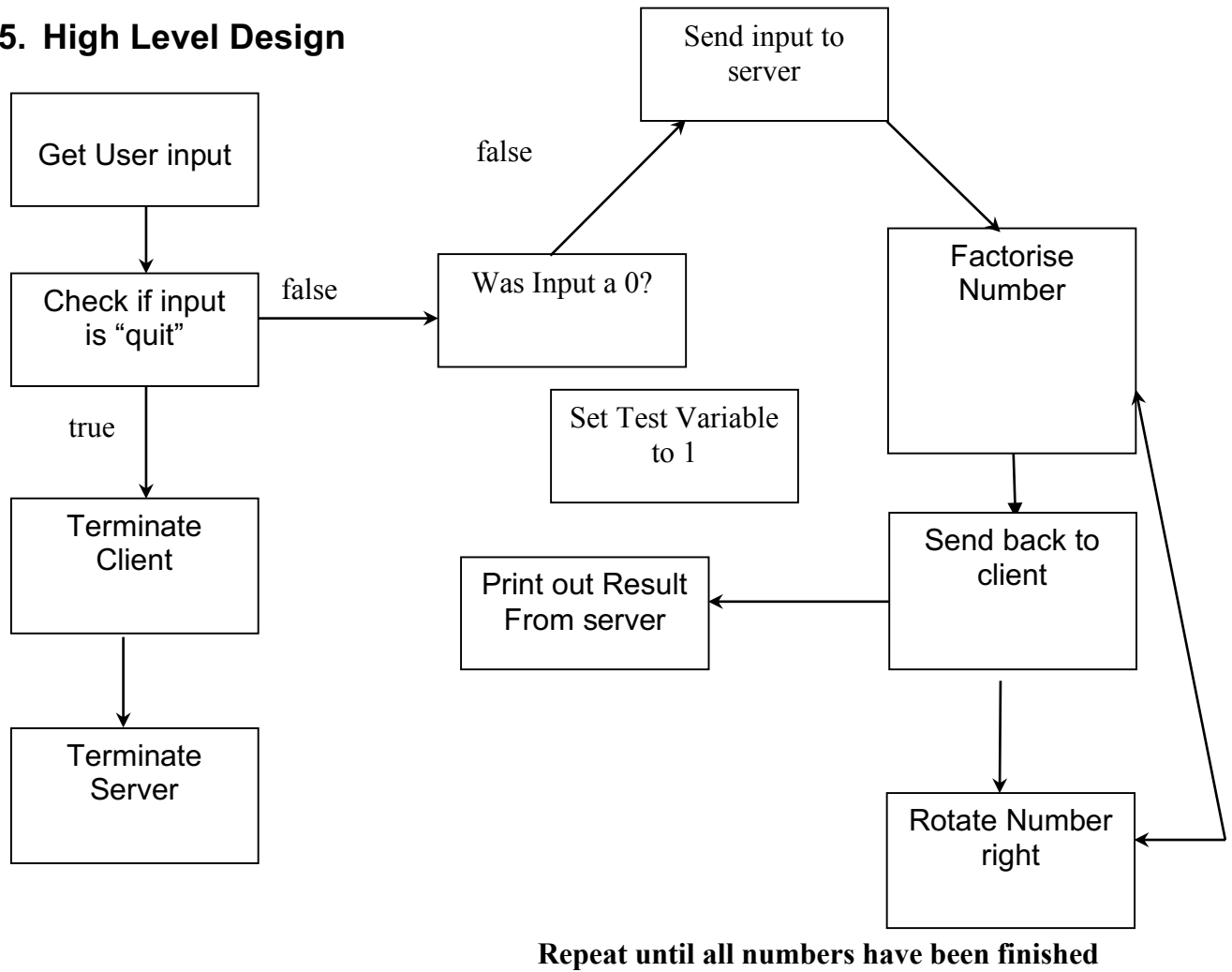
void cleanup()
//When called this will delete all shared memory for the given key

Server

Void threadable(void * input)

This function handles most of the manipulation of the data, it factorizes it and assigns it back to the shared memory for the client to read from, it will also call sem_wait, sem_signal so multiple threads don't access the shared memory at the exact same time.

5. High Level Design







6. Requirement Acceptance Tests

Software Requirement No	Test	Implemented (Full /Partial/ None)	Test Results (Pass/ Fail)	Comments (for partial implementation, failed test results or extra)
1	The program will consist of a multithreaded server and single threaded client process.	FULL	PASS	
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 'q' to quit.	FULL	PASS	
3	The server will take each input number (unsigned long) and will start up either the number of specified threads if given (see Req.17) or as many threads as there are binary digits (i.e. 32 threads). 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 input number is K, each thread #X will factorise K rotated right by (K-1) bits. For example say K and it has N significant bits, then 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. Rotating an integer K by B bits = $(K \gg B) \mid (K \ll (32 - B))$. CLARIFICATION: $C = K \ll (32 - B)$; Rotated = $(K \gg B) \mid C$	FULL	PASS	
4	The trial division method should be used for integer factorisation.	FULL	PASS	
5	The server is must handle up to 10 simultaneous requests without blocking.	FULL	PASS	

Software Requirement No	Test	Implemented (Full /Partial/ None)	Test Results (Pass/ Fail)	Comments (for partial implementation, failed test results or extra)
6	The client is non-blocking. A 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.	FULL	PASS	
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.	FULL	PASS	
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.	FULL	PASS	
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:	FULL	PASS	

Software Requirement No	Test	Implemented (Full /Partial/ None)	Test Results (Pass/ Fail)	Comments (for partial implementation, failed test results or extra)
	<ul style="list-style-type: none"> ○ 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 			
15	<p>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.</p>	FULL	PASS	

Software Requirement No	Test	Implemented (Full /Partial/ None)	Test Results (Pass/ Fail)	Comments (for partial implementation, failed test results or extra)
16	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.	FULL	PASS	
17	Each factorising thread will report its progress as a percentage in increments not larger than 5% as it proceeds to the server primary thread. The server will use the individual thread progress values to calculate an overall progress % for each request being processed and pass this back to the client in increments not larger than 5%. This will require a second shared array char progress[10] that is 10 elements long but no handshaking protocol or synchronisation since it does not matter if the client misses some values or reads them multiple times.	FULL	PASS	
	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 that does not scroll up (see example lab 3). 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 ie > Progress: Q1:50%  Q2:30%  Q3: 80% 	FULL	PASS	

Software Requirement No	Test	Implemented (Full /Partial/ None)	Test Results (Pass/ Fail)	Comments (for partial implementation, failed test results or extra)
	A sample session with only a single query may look like this (the example is contrived)....			
	Enter numbers to factor: 12345678912345 factors: 13 17 Progress: Query 1: 30%  _____ factors: 333 1234 Query complete for 12345678912345	FULL	PASS	
	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.	FULL	PASS	
	The system will have a test mode that will be activated by the user entering 0 as the number to be factored. This will be passed to the server which will launch 3 sets of 10 threads each, each set will simulate one of three user requests. Each thread in each set will return to the client 10 numbers starting with the thread # times 10 and incrementing by 1. For example thread #0 will return numbers 0-9, thread #1 will return numbers 10-19, thread #2 returns numbers 20-29. Progress output will be disabled during the test and it will only run if the server is not processing any outstanding requests instead a warning will be issued to the user.	FULL	PASS	
	If the client is terminated by the user entering 'q' or by typing Ctrl-C (cygwin only) it will first cleanly shutdown the server and then print out a nice termination message.	FULL	PASS	

7. User Instructions

Compiling the Server/Client

Linux:

To compile the software simply use the provided Makefile for each using the “make” command in a linux terminal.

Windows:

Compile using visual studio

Running the Server

The server program does not require user input to run properly, It is optional to add how many threads you would like to run per thread.

If at any point you need to stop the server simply use **CTRL^C**.

Running the Client

To start the client you need to start the executable.

If at any point you need to stop the client you need to input the quit command, or simply use **CTRL^C**.

Commands

The user can input the commands:

q – Will quit the program, this is has the same affect as **CTRL ^ C**

quit - Will quit the program, this is has the same affect as **CTRL ^ C**

A number can be inputted into the program, it will then be sent to the server to perform the factorisation and rotations on.

If the user inputs a 0 it will activate the servers test mode, which will block all user input and count from 0-29 using different threads.