Please fill in each section of this documentation file with the information which we will need to mark your coursework.

**Full Name:** Barnabas Forgo

**University Username:** psybf

**Computer Science Username:** bxf03u

We will try to compile your program by opening the project files, replacing the DoNotChangeThese.h file and recompiling everything.

**Are your projects in one Visual Studio Solution or multiple:**

One solution.

**What are the project names for the different components:**

The part A Client:

clientA

The part B/C Server:

server

The part C Server:

clientC

**Are there any special instructions we need in order to get your programs to run?**

Just click start, the project is set up so that Visual Studio starts the server first, and then the two clients. If that doesn’t work (I never had any issues) then start the server first and then each client.

**What problems do you currently know about with your programs?**

No known problems.

In the general comments section for each requirement, please make comments about what you did for that requirement, the names of the files which you changed and any special comments which you think we should know about.

**Requirement A1:**

General comments:

For creating the threads I made a function pointer array to create them programmatically (compared to an ugly switch for example).

**Requirement A2:**

General comments:

Only these three methods are responsible for handling critical sections. I think their names are self-explanatory.

void IncrementResourceAndEnterCriticalSection(int resourceId)

void EnterResourceCriticalSection(int resourceId)

void LeaveResourceCriticalSection(int resourceId)

These functions block the thread until the resource is available.

Please comment on the method that you used to protect your critical sections and why you believe that it was the most appropriate and/or efficient method:

I used CRITICAL\_SECTION objects, because in labs I learned that these provide the fastest locking and they provide ownership information. This is crucial in my solution as without this, the threads would just lock themselves. The way I avoid deadlocks is that each resource is acquired in the same order (ie. 1, 2, 3,… 6), getting rid of cycles. For this I annotated each TestCase with the resources they use (in comments) so that when other TestCases call them I can easily identify which resources I need to acquire beforehand in that thread.

**Requirement A3:**

General comments:

Please comment on whether you needed to protect your counts from interference between threads and what method you used to do this if so.

A global volatile unsigned int array was used to count resource usage. Initially it was incremented using the interlocked API, but then I realised that I could just move the incrementing in the critical section, changing to a simple pre-increment, which is faster.

**Requirement A4:**

General comments:

Please comment on whether you used PostMessage or SendMessage and briefly say why.

For communicating with the server I just used PostMessage to put the data on the message queue. This way the worker thread doesn’t get blocked even if the receiver end is busy and the data doesn’t get lost since the data on the queue gets dealt with sooner or later.

**Requirement B1:**

General comments:

For this I just copied code from lab/lecture code. I ended up handling the WM\_PAINT message on my own, because I forgot that it was provided, my bad. Counters are simple global int arrays.

**Requirement B2:**

General comments:

I utilised copied code here as well. In the message handler, I just check whether the message ID is in the expected region and change the counters as described in the requirement.

**Resource protection in the server:**

Please comment on whether you needed to protect your resources from interference. If so, please explain under what circumstances you need to protect them and why. If not, please explain why you believe that this is unnecessary.

No protection was necessary as the message loop runs on a single thread. Even socket messages get translated to windows messages.

**Requirement C1:**

General comments:

If you launch the program there will be 6 processes in total: 5 workers and the one creating the shared memory, waiting for the workers to die and printing the resource counts in the end. Note that no new terminals are opened for the new processes, they inherit the parent’s. Command line arguments were used to tell which one is which (copied from labs as well).

Please explain how you share the resource usage counts between the processes:

A simple file mapped, shared struct was created containing the resource counts and mutexes.

Please explain any changes that you had to make to the entry protocols for your critical sections in your clients:

The CRITICAL\_SECTIONs had to be replaced with mutex objects, since CRITICAL\_SECTIONs can’t be shared between processes. The reason why I chose mutexes is that these are the only objects that work in an inter-process fashion and have the ownership information that is crucial to my solution as mentioned before. First I was afraid to use it, because a mutex is a “heavy-weight” object; it has the biggest overhead among the synchronisation methods provided by windows. Only after I switched to mutexes did I realise that they might even be faster than CRITICAL\_SECTIONs. After examining the “issue” I realised that mutexes don’t contain debug information, as opposed to CRITICAL\_SECTIONs (in debug mode). When I ran the client outside of in release mode CRITICAL\_SECTIONs were faster as expected. Other than this I kept the interface of the functions that manage critical sections and increment the resource counts, so the TestCases wouldn’t have to be changed.

**Requirement C2:**

General comments:

These functions handle the creation and closure of the connection with the server.

int InitialiseServerConnection()

int CloseServerConnection()

Only one connection is opened and it is kept until the process is done, to minimise the overhead of creating one every time a message needs to be sent.

Please explain the structure of your messages that you pass from the clients to the server and of any messages which are passed back again.

The client follows a send and forget fashion, that is, the socket message is sent and the process returns immediately. The reason why I chose to do this is that it is unlikely that a packet will be lost locally. Furthermore I think that performance is more important in this task than the data being correct on the server and waiting for a response from the server would have blocked the process. If the same task had to be done over different machines I would have sent a confirmation of some sort.

Please explain the structure of your messages that you pass from the clients to the server and of any messages which are passed back again.

The data is sent in one struct, in the same fashion as the lab example. The struct contains the processID and the resource counts in an array. There is no data sent back from the server.

Please explain any additional critical sections which you now need (if any) and how they have been protected:

In the client there are no new critical sections. The connections don’t need protection, because each client process has its own connection.

There are no new critical sections in the server either, as even though these socket messages are received in separate threads, they just get translated to windows messages identical to the ones sent from the initial client. Then these messages are dealt with on a single thread so there can be no interference.