Exercise 1

1.1.a)

If the -pthread flag is not used, the following errors occur in compilation.
/tmp/cckjy03h.o: In function `main':
thread.c:(.text+0x68): undefined reference to `pthread_create'
thread.c:(.text+0x91): undefined reference to `pthread_join'
collect2: error: ld returned 1 exit status

Using the -pthread flag will direct the compiler to compile the pthread library so that the code can use functions from it.

Exercise 2

2.1.a) 7 in addition to the parent process (8 total).
2.1.b) 3 in addition to the main thread in process (4 total).

Exercise 3

3.1.a)

Result	Explanation
Thread 1:1 says hi! Thread 1:2 says hi! Thread 1:3 says hi! Thread 2:1 says hi! Thread 2:2 says hi! Thread 2:3 says hi! Thread 3:1 says hi! Thread 3:2 says hi! Thread 3:3 says hi! Thread 4:1 says hi! Thread 4:2 says hi! Thread 4:3 says hi! Thread 5:1 says hi! Thread 5:2 says hi! Thread 5:3 says hi! Main thread says hi!	The result comes because the process (main thread) creates 5 threads sequentially (they begin and complete execution one after the other). The reference to the thread number is passed into the first thread when the value is 1. Every thread increments the value by 1. This is how different threads know their thread numbers. The sequence number within the thread is iterated by the individual threads from 0.

3.1.b)

The reference of the memory location with thread number is passed to threads' running function "thread_function()" casted to a void pointer. The increment operation cannot be done on the void data type.

3.1.c)

The following steps happen in the 15th line

1. The void* is casted to int*.
2. The int* is dereferenced to obtain the value.
3. The int vale is incremented.

3.2)

Result	Explanation
Thread 1:1 says hi! Thread 1:2 says hi!	The blocking nature of the program is relaxed. The threads work asynchronously.

Thread 1:1 says hi! Thread 1:3 says hi! Thread 1:2 says hi! Thread 1:1 says hi! Thread 2:3 says hi! Thread 2:2 says hi! Thread 2:1 says hi! Thread 2:3 says hi! Thread 3:2 says hi! Thread 3:1 says hi! Thread 3:2 says hi! Thread 4:3 says hi! Thread 4:3 says hi!	The number of threads created are still 5. The threads run in parallel. The thread number is

3.2.a)

Commenting out line 12

Output	Explanation
Thread 1:1 says hi! Thread 1:2 says hi! Thread 1:3 says hi! Thread 2:1 says hi! Thread 2:2 says hi! Thread 2:3 says hi! Thread 3:1 says hi! Thread 3:2 says hi! Thread 3:3 says hi! Thread 4:1 says hi! Thread 4:2 says hi! Thread 4:3 says hi! Thread 5:1 says hi! Thread 5:2 says hi! Thread 5:3 says hi! Main thread says hi!	The program behaves as the original except for there is no delay between all the lines printed for a single thread.

Commenting out line 35

Output	Explanation
Thread 1:1 says hi! Thread 1:1 says hi! Thread 1:1 says hi! Thread 1:1 says hi! Thread 1:1 says hi! Thread 1:2 says hi! Thread 1:2 says hi! Thread 1:2 says hi! Thread 1:2 says hi! Thread 1:2 says hi! Thread 1:2 says hi! Thread 1:3 says hi! Thread 1:3 says hi! Thread 1:3 says hi!	The time between thread creation is removed. Therefor all threads are initialized with thread count 1 before at least one thread prints a single line. All 5 threads (with thread numbers 1) will print their internal iterations as 1,2,3. Since all threads are of same thread number, which prints first is not visible.

Thread 1:3 says hi! Thread 1:3 says hi! Main thread says hi!	

Commenting out line 37

Output	Explanation
Thread 1:1 says hi! Thread 1:2 says hi! Thread 1:1 says hi! Thread 1:3 says hi! Thread 1:2 says hi! Thread 1:1 says hi! Thread 2:2 says hi! Thread 2:3 says hi! Thread 2:1 says hi! Thread 2:3 says hi! Thread 3:2 says hi! Thread 3:1 says hi! Thread 3:3 says hi! Main thread says hi! Thread 4:2 says hi!	Since there is no sleep between starting the last thread and the server thread printing it's message, the main thread's message can occur before all the other threads finish printing.

3.2.b.i) lines 35 and 37

Result	Explanation
Thread 1:1 says hi! Thread 1:1 says hi! Thread 1:1 says hi! Main thread says hi! Thread 1:1 says hi! Thread 1:1 says hi!	The time between the thread creation is zero. Therefor all 5 threads get created with the same id 1. The individual threads print the first Hi message. There is no delay between the end of thread creation and the message of the main thread. Therefor it gets print asynchronously. The exit() is reached in the main thread without any delay. None of the other threads make it to the second iteration before that. Therefor only 5 1.1 messages are displayed.

3.2.b.ii) lines 12 and 35

Result	Explanation
Thread 1:1 says hi! Thread 1:2 says hi! Thread 1:3 says hi! Thread 1:1 says hi! Thread 2:2 says hi! Thread 2:3 says hi! Thread 1:1 says hi! Thread 3:2 says hi! Thread 3:3 says hi! Thread 4:1 says hi! Thread 4:2 says hi!	Results change. But the last line printed is always by the main thread. Individual threads print asynchronously. The results change because of this (asynchronous running gives rise to race conditions).

Thread 4:3 says hi! Thread 5:1 says hi! Thread 5:2 says hi! Thread 5:3 says hi! Main thread says hi!	

3.2.b.iii) line 12 and 37

Result	Explanation
Thread 1:1 says hi! Thread 1:2 says hi! Thread 1:3 says hi! Thread 2:1 says hi! Thread 2:2 says hi! Thread 2:3 says hi! Thread 3:1 says hi! Thread 3:2 says hi! Thread 3:3 says hi! Thread 4:1 says hi! Thread 4:2 says hi! Thread 4:3 says hi! Thread 5:1 says hi! Thread 5:2 says hi! Thread 5:3 says hi! Main thread says hi!	**37 is already commented and therefor we cannot increase it.** The code was tested while changing the line 35 from sleep(1) to sleep(5) There is no difference in the output. The output will change only if the time for creating a single thread and executing it is more than 1s. But the computer takes way less time than this.

3.2.c.i)

Result	Explanation
Thread 1:1 says hi! Thread 1:2 says hi! Thread 1:1 says hi! Thread 1:2 says hi! Thread 1:3 says hi! Thread 2:1 says hi! Thread 2:2 says hi! Thread 2:3 says hi! Thread 1:1 says hi! Thread 3:2 says hi! Thread 3:3 says hi! Thread 1:3 says hi! Thread 1:1 says hi! Main thread says hi! Thread 5:2 says hi! Thread 5:3 says hi!	The threads are asynchronous because they are not connected by join() blocking. More than one thread may start with the same thread count variable. The thread's count might change while they are in the middle of their internal iterations. Therefor, 5.2 can occur without 5.1. The main thread may finish printing its message before the other threads complete since there is only 1s delay between the thread creation and main thread's printing.

3.2.c.ii)

Yes. Now consistently the main thread's message comes in the last line.

3.3.a)

"you use sleep() statements instead of join calls to get the desired output of a multi-threaded program."

Sleep can be used to mimic almost anything that can be achieved by join if we have an estimate for the portions of code to run (including the time needed to start new threads, perform IO operations etc:). It is impossible to estimate these time periods with 100% certainty. Reducing uncertainty itself requires introducing higher sleep() times.
But this is a huge waste of time. The same output can be achieved in a short time with join.

Exercise 4

4.1) ex4.c

4.2)

confd is a pointer to the connection file descriptor (an integer).
Assume that confd was pointing to a location in stack. Then the thread should make a copy of the file descriptor value before the next server loop iteration overwrites the stack memory location. This can give rise to race conditions (probably the server loop will iterate before the overhead of initiating a new thread is taken care of.)