1. Output after running the program. (replacing exit with return)

Parent sees i = 0

Parent sees i = 1

Child sees i = 1

Parent sees i = 2

Child sees i = 2

Child sees i = 0

1. Output after running the program. (replacing exit with return, including the header file pthread.h and commenting out the while(1) loop which made program go into infinite wait state)

Parent says a: 1

id: 1 a: 2 b: 1

id: 2 a: 3 b: 1

Thread 1 and 2 complete

1. In multi-threaded applications, mapping m user-level threads to n kernel-level threads can prove to be a good idea as the context switching becomes fast owing to the fact that system calls are now avoided. Also, it gets best of user and kernel level implementations as many short lived user level threads are mapped to a big pool.

m>>n best option because thread switching does not require kernel mode privileges, which saves overhead.

m>n reasonable option as scheduling can be application specific.

m=n not a good option as one process occurs on one processor, which is not multi-threading in actuality.

m<n reasonable option as not much overhead is created.

m<<n not a good option as many system calls block threads.