Threads Review

4.1, 4.2, 4.3, 4.10, 4.11, 4.13, 4.17

1. Draw the process memory layout for a process with exactly one thread. Be sure to include text, data, shared libraries (libc.so), the heap, and the stack.

2. Draw the process memory layout for a process with exactly two threads. Be sure to include text, data, shared libraries (libc.so), the heap, and the stack. Be sure to indicate which parts of the memory space are shared between threads.

3. Draw the PCB or task_struct for each thread in 2 above. Be sure to indicate which portions of the task_struct are shared and which are private to each thread. You don't need to draw the entire task_struct. Refer to the notes to see the parts of the task_struct we said were relevant.

4. How is a userspace thread different than a kernel space thread? What are the advantages of using a kernel space thread?

5. Why is creating a thread faster than creating a process?

6. When might you choose to create a new process rather than create a new thread?

7. What are the hazards associated with using threads?

8. Define race condition. Using a C statement such as x++, explain how incorrect data can end up in x if two threads run x++ simultaneously. Show assembly instructions for full credit.

9. How big is a long double? Why does using a long double (rather than a char or int) make it more likely to create a race condition?

10. Define atomic.

12. Define safety.

12. Look at the source code for strok() and strtok_r() linked on the class schedule web page. Explain why strtok() is not thread-safe and why strtok_r() is thread safe. Be sure to discuss which data is shared among threads and where that data is stored.

13. What is the difference between a multicore CPU and a dual CPU system?

14. What challenges are involved with mulithreaded programming?