4.8)

The components of program state that are shared across threads are Heap Memory and Global Variables. Each thread has its own set of CPU registers and Private Stack (local variables, function calls, and return addresses).

4.9)

A multithreaded solution using only user-level threads does not perform better on a multiprocessor system because the OS only schedules the entire process on one CPU. To utilize multiple processors more efficiently, the system must support kernel-level threads. User-level threads are scheduled and managed entirely in user space without the kernel's involvement and the operating system is unaware of the existence of these threads.

4.7)

One circumstance can be when a thread makes a block system call. In a single threaded process if this happens then the entire process is stalled. Vice versa, in a multithreaded process with kernel threads, the OS can schedule another thread from the same process to execute. Or also when a thread is waiting for I/O, other threads could execute. But generally multithreaded can be better because of responsiveness, resource sharing, economy, and scalability. Responsiveness because let’s say a user wants to click on a button in a single threaded application, he would not be able to do so until the current operation is not completed. In contrast, if the time-consuming operation is performed in a separate thread, the application remains responsive. Resource sharing because thread share the memory and the resources of the process to which they belong by default. Scalability because many benefits of multithreading can be even greater in a multiprocessor architecture, where threads may be running in parallel on different processing cores.