1.

Shared Memory: Threads within the same process share the same memory space, so switching between threads does not require changes to memory mapping, whereas switching between processes requires setting up new memory mappings.
Lower Overhead in Context Switching: Thread switching involves fewer register and stack changes than process switching, as threads share most resources.
No Separate Address Space: Threads operate within the same address space, which eliminates the need for switching to a different address space, reducing the overhead compared to processes that each have their own address space.

2.

Faster Context Switching: ULTs do not require kernel intervention for thread management, which makes context switching between ULTs faster than KLTs.
Customizable Scheduling: ULTs allow for application-specific scheduling policies, giving applications more control over how threads are managed.
Less Kernel Dependency: ULTs do not rely on kernel support for thread management, which can make them more portable across different operating systems.

3.

Jacketing is a technique used to wrap blocking system calls with a layer that checks for conditions that could prevent a user-level thread from blocking the entire process. This layer redirects or handles system calls in a way that prevents a blocking call from affecting other threads in the process.

4.

Since ULTs are managed by the user-space and not by the operating system kernel, the kernel views the process as a single entity. When a ULT makes a blocking system call, the entire process is blocked from the perspective of the OS, which blocks all threads within that process until the call completes.

5.

No, when a process terminates, all threads associated with that process are terminated as well. The operating system manages threads at the process level, so if the main process exists, it results in the termination of all its threads.