* Multiple threads divide the stack into different sections
  + Heap division depends on OS
* Sharing of the text section depends on what is in each thread
* Data section will be shared
* User threads and kernel threads:
  + Many-to-one
    - Multiple user threads in a process cannot be executed on different processors in parallel since a single kernel thread can only execute one user thread at a time
  + Many-to-many
    - User threads scheduled by the user scheduler, kernel threads scheduled by the kernel scheduler (which can also specify the number of kernel threads
  + Two-level model
    - Allows for a combination of many-to-one and one-to-one (many-to-many but with some threads being able to run one-to-one)
  + One-to-one
    - Scheduling or user threads is not required, just kernel scheduling
    - More concurrent threads running at once
* Threads can be cancelled when they hit a cancellation point (one example point is when it opens a file). These points are safe times for a thread to end
  + Signal handlers are used to notify when these events happen (or any event)
* Worker-pool: pool of threads that take up a request and returns to the pool when it is finished (not destroyed)
  + Faster than making a new thread
* Thread-specific data is stored in the heap (such as a transaction ID for a service transaction)
* Light-weight threads act as a middleman between user and kernel threads
* Scheduler activations: calls from kernel to process to notify a process user-level scheduler of an event
* In one-to-one threads, each thread could run on its own processor in parallel
  + Schedule activation is not required