Modern operating systems support multithreading.
- Process is divided into threads that can run concurrently.
- Thread:
  - schedulable unit of work
  - executes sequentially and is interruptible
- Process is a collection of one or more threads.

Benefits:
- Responsiveness (one thread may be blocked but other can be still runnable).
- Resource sharing (threads share the memory and resources of the process to which they belong).
- Economy (easy context switch between threads).
- Utilization of multiprocessor architectures.

Multithreading programming is putting pressure on programmer.

Challenges include:
- dividing activities
- balance
- data splitting
- data dependency
- testing and debugging
Multithreaded Processes
- A thread library provides the programmer API for creating and managing threads.
- User threads: thread management done by user-level threads library, not by operating system
- Kernel threads: supported by all contemporary operating systems:
  - POSIX threads (called pthreads) can be provided as either a user or kernel-level library.
- Multithreading models – define relationships between user and kernel threads:
  - many-to-one
  - one-to-one
  - many-to-many

Many-to-One Model
- Many user-level threads mapped to single kernel thread.
- Used on systems that do not support kernel threads.

One-to-One Model
- Each user-level thread maps to kernel thread.
- Used by many operating systems.

Many-to-Many Model
- Allows many user level threads to be mapped to a smaller or equal number of kernel threads.
### PThread Library Calls

- **int pthread_create (pthread_t *thread, const pthread_attr_t *attr, void *(*start_routine) (void*), void *arg);**
  - the thread is created executing start_routine with arg as its sole argument
  - upon successful completion, pthread_create() stores the ID of the created thread in the location referenced by thread.
  - attr is normally NULL

- **void pthread_exit (void *value_ptr);**
  - value_ptr is made available to any successful join with the terminating thread

- **int pthread_join (pthread_t thread, void **value_ptr);**
  - waits for the thread specified by thread to terminate; if that thread has already terminated, then pthread_join() returns immediately

- **int pthread_self (void)**
  - returns the ID of the calling thread

### Linux (Unnamed) Semaphore Library Calls

- **int sem_init(sem_t *sem, int pshared, unsigned int value);**
  - initializes the unnamed semaphore at the address pointed to by sem; the value argument specifies the initial value for the semaphore; if pshared has the value 0, then the semaphore is shared only between the threads of a process

- **int sem_wait(sem_t *sem);**
  - performs wait operation on semaphore at the address pointed to by sem;

- **int sem_post(sem_t *sem);**
  - performs signal operation on semaphore at the address pointed to by sem;