There are two types of semaphores : Binary Semaphores and Counting Semaphores

* + Binary Semaphores : They can only be either 0 or 1. They are also known as mutex locks, as the locks can provide mutual exclusion. All the processes can share the same mutex semaphore that is initialized to 1. Then, a process has to wait until the lock becomes 0. Then, the process can make the mutex semaphore 1 and start its critical section. When it completes its critical section, it can reset the value of mutex semaphore to 0 and some other process can enter its critical section.
  + Counting Semaphores : They can have any value and are not restricted over a certain domain. They can be used to control access a resource that has a limitation on the number of simultaneous accesses. The semaphore can be initialized to the number of instances of the resource. Whenever a process wants to use that resource, it checks if the number of remaining instances is more than zero, i.e., the process has an instance available. Then, the process can enter its critical section thereby decreasing the value of the counting semaphore by 1. After the process is over with the use of the instance of the resource, it can leave the critical section thereby adding 1 to the number of available instances of the resource.

#include <semaphore.h>

int sem\_init(sem\_t \*sem, int pshared, unsigned int value);

So pshared is a boolean value: in practice meaningful values passed to it are false (0) and true(1), though any non-0 value will be treated as true. If you pass it 0 you will get a semaphore that can be accessed by other threads in the same process -- essentially an in-process lock. You can use this as a mutex, or you can use it more generally for the resource-counting properties of a semaphore. Arguably if pthreads supported a semaphore API you wouldn't need this feature of sem\_init, but semaphores in Unix precede pthreads by quite a bit of time.

The pshared argument indicates whether this semaphore is to be shared between the threads of a process, or between processes.

If pshared has the value 0, then the semaphore is shared between the threads of a process, and should be located at some address that is visible to all threads (e.g., a global variable, or a variable allocated dynamically on the heap).

If pshared is nonzero, then the semaphore is shared between processes, and should be located in a region of shared memory (see shm\_open(3), mmap(2), and shmget(2)). (Since a child created by fork(2) inherits its parent's memory mappings, it can also access the semaphore.) Any process that can access the shared memory region can operate on the semaphore using sem\_post(3), sem\_wait(3), etc.