This text is based on http://www.di.ubi.pt/~operativos/praticos/html/13-sinc.html
Last visit on 15/May/2018.

POSIX Threads

1) Primitives to create and synchronize the conclusion of threads (pthreads.h).

Type for thread handle: pthread_t
```
  int pthread_create(pthread_t  *thread_handle, pthread_attr_t *attr, void *
(*thread_function)(void*), void * arg);
  int pthread_join(pthread_t thread, void **ptr);
```

2) Semaphores

Semaphores are counters to synchronize the use of resources shared between threads. Basic operations on semaphores: increment the counter atomically, wait until the counter is non-null and decrement it atomically.

2.1) Non-Binary semaphores: primitives for mutual exclusion in threads (pthreads.h).

Type for the semaphore used for mutual exclusion: pthread_mutex_t
```
  int pthread_mutex_init(pthread_mutex_t *mutex_lock,const pthread_mutexattr_t *lock_attr);
  int pthread_mutex_lock(pthread_mutex_t *mutex_lock);
  int pthread_mutex_unlock(pthread_mutex_t *mutex_lock);
```

2.2) Non-binary semaphores (POSIX 1003.1b semaphores) <semaphore.h>

Type: sem_t
```
  int sem_init(sem_t *sem, int pshared, unsigned int init_value);
  int sem_wait(sem_t *sem);
  int sem_post(sem_t *sem);
  int sem_getvalue(sem_t *sem, int *sval);
  int sem_destroy(sem_t *sem);
```

2.2.1) Some explanations about non-binary semaphores:

sem_init initializes the semaphore object pointed to by sem.
The count associated with the semaphore is set initially to value.
The pshared argument indicates whether the semaphore is local to the current process (pshared is zero) or is to be shared between several processes (pshared is not zero).
LinuxThreads currently does not support process-shared semaphores, thus sem_init always returns with error ENOSYS if pshared is not zero.

sem_wait suspends the calling thread until the semaphore pointed to by sem has non-zero count.
It then atomically decreases the semaphore count.

sem_post atomically increases the count of the semaphore pointed to by sem.
This function never blocks and can safely be used in asynchronous signal handlers.

sem_getvalue stores in the location pointed to by sval the current count of the semaphore sem.

sem_destroy destroys a semaphore object, freeing the resources it might hold.
No threads should be waiting on the semaphore at the time sem_destroy is called.
In the LinuxThreads implementation, no resources are associated with semaphore objects, thus sem_destroy actually does nothing except checking that no thread is waiting on the semaphore.

2.2.2) Return Value
The sem_wait and sem_getvalue functions always return 0.
All other semaphore functions return 0 on success and -1 on error, in addition to writing an error code in errno.