POSIX Functions

Creating a thread:

int pthread_create(pthread_t *restrict thread, const pthread_attr_t *restrict attr, void *(*start_routine)(void*), void *restrict arg);

Creates a new thread and starts execution of the function **start_routine** with the given argument **arg**. The thread's ID is stored in the location pointed to by **thread**. Optional attributes can be specified using **attr**.

Joining a thread:

int pthread_join(pthread_t thread, void **value_ptr);

Waits for the thread specified by **thread** to terminate. If the thread has already terminated, **pthread_join** returns immediately. If **value_ptr** is not **NULL**, the exit status of the terminated thread will be stored in the location pointed to by **value_ptr**.

Exiting a thread:

void pthread_exit(void *value_ptr);

Terminates the calling thread and returns a value specified by **value_ptr**. The resources associated with the thread are released by the system. If **pthread_exit** is called from the main thread, it will terminate the entire process.

Creating a mutex:

int pthread_mutex_init(pthread_mutex_t *restrict mutex, const pthread_mutexattr_t *restrict attr);

Initializes a mutex pointed to by mutex with attributes specified by attr. If attr is NULL, default attributes are used.

Locking a mutex:

int pthread_mutex_lock(pthread_mutex_t *mutex);

Locks the mutex pointed to by mutex. If the mutex is already locked by another thread, the calling thread will block until it can acquire the lock.

int pthread_mutex_trylock(pthread_mutex_t *mutex);

Tries to lock the mutex pointed to by mutex. If the mutex is currently unlocked, the calling thread acquires the lock and returns immediately with a return value of 0 (success). If the mutex is already locked by another thread, the function returns immediately with a return value of **EBUSY** to indicate that the lock is not acquired.

Unlocking a mutex:

int pthread_mutex_unlock(pthread_mutex_t *mutex);

Unlocks the mutex pointed to by mutex. If there are threads waiting to acquire the lock, one of them will be unblocked and granted the lock.

Creating/Destroying condition:

int pthread_cond_init(pthread_cond_t *restrict cond, const pthread_condattr_t *restrict attr);

Initializes a condition variable pointed to by **cond** with attributes specified by **attr**. If **attr** is **NULL**, default attributes are used.

pthread_cond_t cond = PTHREAD_COND_INITIALIZER;

Static initializer for a condition variable. This macro initializes a condition variable statically with default attributes.

int pthread_cond_destroy(pthread_cond_t *cond);

Destroys the condition variable specified by **cond**. It should not be in use (no threads are waiting on it) when this function is called.

Waiting on condition:

int pthread_cond_wait(pthread_cond_t *restrict cond, pthread_mutex_t *restrict mutex);

Atomically unlocks the mutex specified by **mutex** and waits on the condition variable specified by **cond**. The mutex must be locked by the calling thread. Upon successful return, the mutex will be locked by the calling thread.

int pthread_cond_timedwait(pthread_cond_t *restrict cond, pthread_mutex_t *restrict mutex, const struct timespec *restrict abstime);

Similar to **pthread_cond_wait**, but this function waits until the absolute time specified by **abstime** is reached. If the condition variable is signaled before the specified time, the function will return. If the specified time elapses, the function returns with a timeout error.

Waking thread based on condition:

int pthread_cond_signal(pthread_cond_t *cond);

Wakes up one thread waiting on the condition variable specified by **cond**, if there are any waiting threads. If multiple threads are waiting, which thread gets woken up is not specified.

int pthread_cond_broadcast(pthread_cond_t *cond);

Wakes up all threads waiting on the condition variable specified by **cond**. All waiting threads are awakened and will compete for the mutex lock associated with the condition variable.

Examples

```
pthread_mutex_t mutex = PTHREAD_MUTEX_INITIALIZER;
// Attempt to acquire the lock
if (pthread_mutex_trylock(&mutex) == 0) {
    // Lock acquired successfully
    // Perform operations while holding the lock
    // Release the lock
    pthread_mutex_unlock(&mutex);
} else {
    // Failed to acquire the lock
    // Another thread holds the lock
    // Handle the case where the lock is not acquired
}
```

Skeletons

Listing 1: Skeleton Thread Program

```
#include <pthread.h>

/*
 * The function to be executed by the thread should take a
 * void* parameter and return a void* exit status code.
 */
void *thread_function(void *arg)
{
  // Cast the parameter into what is needed.
  int *incoming = (int *)arg;

  // Do whatever is necessary using *incoming as the argument.

  // The thread terminates when this function returns.
  return NULL;
}

int main(void)
{
  pthread_t thread_ID;
  void      *exit_status;
  int        value;

  // Put something meaningful into value.
  value = 42;

  // Create the thread, passing &value for the argument.
  pthread_create(&thread_ID, NULL, thread_function, &value);

  // The main program continues while the thread executes.

  // Wait for the thread to terminate.
  pthread_join(thread_ID, &exit_status);

  // Only the main thread is running now.
  return 0;
}
```

Listing 2: Mutex Example

```c
#include <pthread.h>
#include <unistd.h>

pthread_mutex_t lock;
int shared_data;
  // Often shared data is more complex than just an int.

void *thread_function(void *arg)
{
  int i;

  for (i = 0; i < 1024*1024; ++i) {
    // Access the shared data here.
    pthread_mutex_lock(&lock);
    shared_data++;
    pthread_mutex_unlock(&lock);
  }
  return NULL;
}

int main(void)
{
  pthread_t thread_ID;
  void      *exit_status;
  int        i;

  // Initialize the mutex before trying to use it.
  pthread_mutex_init(&lock, NULL);

  pthread_create(&thread_ID, NULL, thread_function, NULL);

  // Try to use the shared data.
  for (i = 0; i < 10; ++i) {
    sleep(1);
    pthread_mutex_lock(&lock);
    printf("\rShared integer's value = %d\n", shared_data);
    pthread_mutex_unlock(&lock);
  }
  printf("\n");

  pthread_join(thread_ID, &exit_status);

  // Clean up the mutex when we are finished with it.
  pthread_mutex_destroy(&lock);
  return 0;
}
```

Listing 3: Condition Variable Example

```
#include <pthread.h>
#include <unistd.h>

pthread_cond_t  is_zero;
pthread_mutex_t mutex;    // Condition variables needs a mutex.
int shared_data = 32767; // Or some other large number.

void *thread_function(void *arg)
{
  // Imagine doing something useful.
  while (shared_data > 0) {
    // The other thread sees the shared data consistently.
    pthread_mutex_lock(&mutex);
    --shared_data;
    pthread_mutex_unlock(&mutex);
  }

  // Signal the condition.
  pthread_cond_signal(&is_zero);
  return NULL;
}

int main(void)
{
  pthread_t thread_ID;
  void      *exit_status;
  int        i;

  pthread_cond_init(&is_zero, NULL);
  pthread_mutex_init(&mutex, NULL);

  pthread_create(&thread_ID, NULL, thread_function, NULL);

  // Wait for the shared data to reach zero.
  pthread_mutex_lock(&mutex);
  while (shared_data != 0)
    pthread_cond_wait(&is_zero, &mutex);
  pthread_mutex_unlock(&mutex);

  pthread_join(thread_ID, &exit_status);

  pthread_mutex_destroy(&mutex);
  pthread_cond_destroy(&is_zero);
  return 0;
}
```