TP4_PBT

Night

March 2023

1 Conditions

Voir mthread_cond.c

On reprend la librairie mthread modifiée du tp3 avec les sémaphores implémentés.

1.1

```
int mthread_cond_init(mthread_cond_t *cond, const mthread_condattr_t *cond_attr)
{
  mthread_log("COND INIT", "Initializing\n");

  if (cond == NULL)
  {
    mthread_log("COND INIT", "Returning EINVAL\n");
    return EINVAL;
  }

  cond->lock = 0;

  __mthread_cond_unchecked_ensure_thread_queue_init(cond);

  mthread_log("COND INIT", "Initialized\n");
  return 0;
}
```

1.2

```
int mthread_cond_wait(mthread_cond_t *cond, mthread_mutex_t *mutex)
{
  mthread_log("COND WAIT", "Waiting\n");
  if (cond == NULL || mutex == NULL)
  {
    mthread_log("COND WAIT", "Arg was NULL\n");
    return EINVAL;
  }

  __mthread_cond_unchecked_ensure_thread_queue_init(cond);

  mthread_spinlock_lock(&cond->lock);

  mthread_virtual_processor_t *vp = mthread_get_vp();

  // Added: saving the previous state, to ensure the rollback is possible if necessa
  volatile struct mthread_s *prev_last = cond->thread_queue->last;
  mthread_tst_t *prev_lock = vp->p;

  // Added: blocking the current thread
  mthread_t self = mthread_self();
  self->status = BLOCKED;
  vp->p = &cond->lock;

  // Added: inserting the current thread at the end of the waiting list for the cond
  mthread_insert_last(self, cond->thread_queue);

  // Added: unlocking the mutex and, in case there was an error, rollback the change
  int err = mthread_mutex_unlock(mutex);
  if (err != 0)
  {
    self->status = RUNNING;
    cond->thread_queue->last = prev_last;
    prev_last->next = NULL;
    vp->p = prev_lock;
    mthread_spinlock_unlock(&cond->lock);
    mthread_log("COND WAIT", "Error unlocking mutex\n");
    return err;
  }

  mthread_spinlock_unlock(&cond->lock);

  mthread_yield();

  // Added: relocking the mutex before exiting the function, returning the result
  mthread_log("COND WAIT", "Waited\n");
  return mthread_mutex_lock(mutex);
}
```

1.3

```
int mthread_cond_signal(mthread_cond_t *cond)
{
  mthread_log("COND SIGNAL", "Signaling\n");
  if (cond == NULL)
  {
    mthread_log("COND SIGNAL", "Returning EINVAL\n");
    return EINVAL;
  }

  __mthread_cond_unchecked_ensure_thread_queue_init(cond);

  mthread_spinlock_lock(&cond->lock);

  // Added: get the first thread to wait, which is also the first we will signal
  struct mthread_s *th = mthread_remove_first(cond->thread_queue);

  // Added: ensure there is at least one waiting thread
  if (th == NULL)
  {
    mthread_spinlock_unlock(&cond->lock);
    mthread_log("COND SIGNAL", "No waiting thread\n");
    return EINVAL;
  }

  mthread_virtual_processor_t *vp = mthread_get_vp();
  th->status = RUNNING;
  mthread_insert_last(th, &(vp->ready_list));

  mthread_spinlock_unlock(&cond->lock);

  mthread_log("COND SIGNAL", "Signaled\n");
  return 0;
}
```

1.4

```
int mthread_cond_broadcast(mthread_cond_t *cond)
{
  mthread_log("COND BROADCAST", "Broadcasting\n");
  if (cond == NULL)
  {
    mthread_log("COND BROADCAST", "Returning EINVAL\n");
    return EINVAL;
  }

  __mthread_cond_unchecked_ensure_thread_queue_init(cond);

  mthread_spinlock_lock(&cond->lock);

  // Added: get all the threads, one after the other, and set them to running.
  mthread_virtual_processor_t *vp = mthread_get_vp();
  struct mthread_s *th = NULL;
  while ((th = mthread_remove_first(cond->thread_queue)) != NULL)
  {
    th->status = RUNNING;
    mthread_insert_last(th, &(vp->ready_list));
  }

  mthread_spinlock_unlock(&cond->lock);

  mthread_log("COND BROADCAST", "Broadcasted\n");
  return 0;
}
```

1.5

```
int mthread_cond_destroy(mthread_cond_t *cond)
{
  mthread_log("COND DESTROY", "Destroying\n");

  if (cond == NULL)
  {
    mthread_log("COND DESTROY", "Returning EINVAL\n");
    return EINVAL;
  }

  mthread_spinlock_lock(&cond->lock);
  if (cond->thread_queue != NULL && cond->thread_queue->first != NULL)
  {
    mthread_spinlock_unlock(&cond->lock);
    mthread_log("COND DESTROY", "Returning EBUSY\n");
    return EBUSY;
  }

  // The queue is free but not it's members: they may be used elsewhere,
  // for another conditions for example
  if (cond->thread_queue != NULL)
  {
    free(cond->thread_queue);
    cond->thread_queue = NULL;
  }

  mthread_spinlock_unlock(&cond->lock);

  mthread_log("COND DESTROY", "Destroying\n");
  return 0;
}
```

1.6

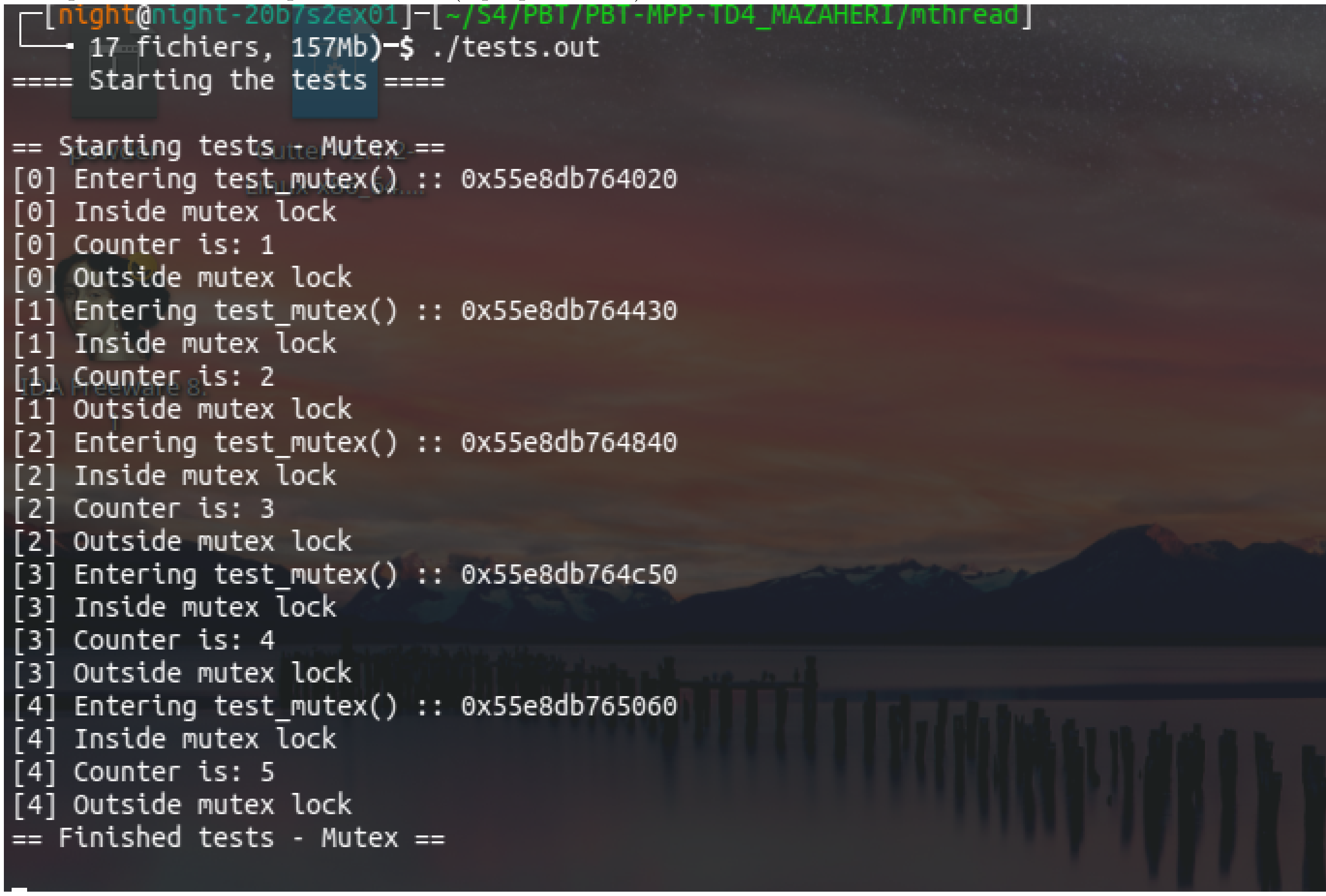
Voir test.c
Ici , on test la fonction signal:

```
mthread_cond_t cond_signal = MTHREAD_COND_INITIALIZER;
mthread_mutex_t mutex_signal = MTHREAD_MUTEX_INITIALIZER;
void *test_cond_signal(void *arg)
{
  const long thread_num = (long)arg;
  fprintf(stderr, "[%ld] Entering test_cond_signal() :: %p\n", thread_num, mthread_s

  if (thread_num == 0)
  {
    sleep(10);
    fprintf(stderr, "[%ld] Starting signaling\n", thread_num);
    for (int k = 1; k < NB_THREADS; k++)
    {
      mthread_cond_signal(&cond_signal);
      sleep(1);
    }
    fprintf(stderr, "[%ld] Finished signaling\n", thread_num);
  }
  else
  {
    mthread_mutex_lock(&mutex_signal);
    fprintf(stderr, "[%ld] Waiting for cond signal\n", thread_num);
    mthread_cond_wait(&cond_signal, &mutex_signal);
    fprintf(stderr, "[%ld] Got signaled\n", thread_num);
    mthread_mutex_unlock(&mutex_signal);
  }

  return NULL;
}
```

Exemple de la sortie des premiers tests (laptop 4 coeurs):



```
┌─[night@night-20b7s2ex01]─[~/S4/PBT/PBT-MPP-TD4_MAZAHERI/mthread]
└──╼ 17 fichiers, 157Mb)─$ ./tests.out
==== Starting the tests ====

== Starting tests - Mutex ==
[0] Entering test_mutex() :: 0x55e8db764020
[0] Inside mutex lock
[0] Counter is: 1
[0] Outside mutex lock
[1] Entering test_mutex() :: 0x55e8db764430
[1] Inside mutex lock
[1] Counter is: 2
[1] Outside mutex lock
[2] Entering test_mutex() :: 0x55e8db764840
[2] Inside mutex lock
[2] Counter is: 3
[2] Outside mutex lock
[3] Entering test_mutex() :: 0x55e8db764c50
[3] Inside mutex lock
[3] Counter is: 4
[3] Outside mutex lock
[4] Entering test_mutex() :: 0x55e8db765060
[4] Inside mutex lock
[4] Counter is: 5
[4] Outside mutex lock
== Finished tests - Mutex ==
```

2 Les clés posix

2.1 q7

```
int mthread_key_create(mthread_key_t *__key, void (*__destr_function)(void *))
{
    return pthread_key_create(__key, __destr_function);
}
```

2.2 q8

```
int mthread_key_delete(mthread_key_t __key)
{
    return pthread_key_delete(__key);
}
```

2.3 q9

2.4 q10

2.5 q11