UTA

Tracing/Replay for Semaphores and Locks

- PV-sequence & Lock/Unlock-sequence
- Trace and Replay

UTA

Shared Objects

Let P be a concurrent program that uses shared variables, semaphores and locks.

The outcome of executing P with a given input depends on the order in which shared objects in P are accessed.

P's shared objects are its shared variables, semaphores, and locks.

PV-sequence

A SYN-sequence for a semaphore is referred to as a PV-sequence, which consists of a sequence of events of the following types:

- completion of a P operation
- completion of a V operation
- start of a P operation that is never completed
- start of a V operation that is never completed

PV-sequence (cont'd)

An event in a PV-sequence is encoded by the identifier (ID) of the thread that executes the P or V operation.

The order in which threads complete P and V operations is not necessarily the same as the order in which these P and V operations start.

PV-sequence (cont'd)

Let s be a binary semaphore initialized with 0.

Thread 1	Thread 2
...	...
s.P();	s.V();
...	...
s.V();	s.P();
...	...
s.V();	s.P();
...	...
s.V()	s.V()
...	...

2, 1, 1, 2, 1, 2, 1, 2

Lock/Unlock-sequence

A SYN-sequence for a lock is referred to as a Lock/Unlock-sequence, which consists of a sequence of events of the following types:

- completion of a lock operation
- completion of a unlock operation
- start of a lock operation that is never completed
- start of a unlock operation that is never completed

Lock/Unlock-sequence (cont'd)

An event in a Lock/Unlock-sequence is encoded by the identified (ID) of the thread that executed the Lock or Unlock operation.

The order in which threads complete Lock and Unlock operations is not necessarily the same as the order in which these Lock and Unlock operations start.

Lock/Unlock-sequence (cont'd)

Let l be a mutex lock.

Thread 1	
...	
l.lock();	(1)
x = 1;	(2)
l.unlock();	(3)
...	

Thread 2	
...	
l.lock();	(1)
x = 2;	(2)
l.unlock();	(3)
...	

1, 1, 2, 2

UTA

SYN-sequence for a program

A SYN-sequence for a concurrent program is a collection of RW-sequence, PV-sequence, and Lock/Unlock-sequence, where there is one sequence for each shared variable, semaphore, and lock in the program.

UTA

Replay

Assume that shared variables are always safely accessed within critical sections.

If we replay PV-sequence or Lock/Unlock-sequence, then the RW-sequence of each shared variable will be replayed.

UTA

P & V methods

```
P() {
   if (replay) {
      control.requestPermit (ID);
   }
   // enter the CS
   if (replay) {
      control.releasePermit();
   }

   /* body of P () */

   if (trace) {
      control.traceCompleteP (ID);
   }
   // leaving the CS
}
```

```
V() {
   if (replay) {
      control.requestPermit(ID);
   }
   // enter the CS
   if (replay) {
      control.releasePermit();
   }

   /* body of P () */

   if (trace) {
      control.traceCompleteV (ID);
   }
   // leave the CS
}
```


UTA

Lock & Unlock methods

Lock & Unlock methods for a mutex lock will be implemented in a way that is similar to P & V methods.

UTA

class control

Each semaphore or lock is associated with a control object.

In trace mode, the control object collects and records the sequence of synchronization events.

In replay mode, the control object inputs the SYN-sequence of the semaphore or mutex lock and handles the calls to requestPermit and releasePermit.

UTA

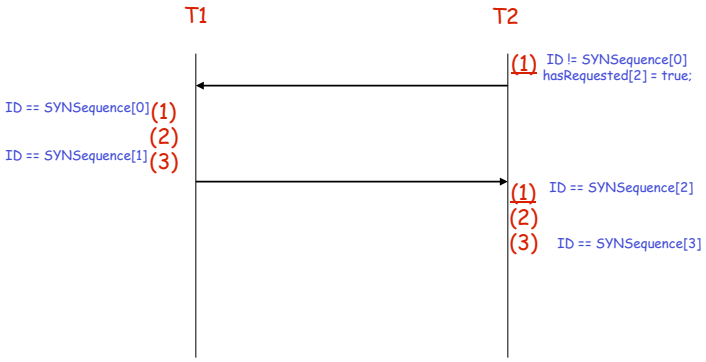
class control (cont'd)

```
public class control {
        private Vector SYNsequence;        // PV-sequence or Lock/Unlock-sequence; sequence of IDs
        private BinarySemaphore [] threads;          // all semaphores initialized to 0
        boolean [] hasRequested;             // hasRequested[i] is true if Ti is delayed in requestPermit
        int index = 0;                       // index into SYNsequence
        MutexLock mutex;

        public control () {
            // initialization
        }
        public void requestPermit (int ID) {
            mutex.lock ();
            if (ID != SYNsequence[index]) {
                hasRequested[ID] = true;
                mutex.unlock ();
                threads[ID].P ();
                hasRequested[ID] = false;
            }
            else mutex.unlock ();
        }
        public void releasePermit () {
            mutex.lock ();
            ++ index;
            if (index < SYNSequence.size ()) {
                if (hasRequested[SYNsequence[index]]) {
                    threads[SYNsequence[index]].V();
                }
            }
            mutex.unlock ();
        }

        public void traceCompleteP (int ID) { ... } // record integer ID
        public void traceCompleteV (int ID) { ... } // record integer ID
}
```

UTA

Example

Consider the example lock/unlock sequence discussed earlier. What happens if T2 attempts the lock operation first during replay?



1, 1, 2, 2