Multiprocessing

Multiple CPUs, cores, or hyperthreads

Multiprogramming

Multiple jobs or processes

Multithreading

Multiple threads per process.

Properties of synchronization primitives

* for locks
  - ✓ Atomic, Correctness
  - ✓ Performance
  - ✓ Fairness
  - ✓ Minimize use of locks as much as possible
  - ✓ Hierarchical usage of locks.

Read-writer pattern

Scenario: There might be possible no: of readers and writer for a shared resource.

Only one writer is allowed at a time in critical section but there could be multiple readers. Example: Editing a table's particular row. Writer will be only accessing it. There could be multiple readers for entire table.

Timer: Application wants to know time.

Application uses multiple readers to read the time but kernel is the only one who writes the time .

Implement read-writer lock using semaphore

```
sid writer = semcreate(1); sid mutex = semcreate
int reader_cnt = 0;                          (1);
// WRITERS
   lock_writer() {              → we need to wait
         wait (writer);            for writer semaphore.


}
// I'm writting
   unlock_writer() {  →   leave the critical
        signal (writer);     section .


}

// Readers
   lock_readers () {
           wait (mutex)
            reader_cnt ++;
             if (reader_cnt == 1) {
                    wait (writer) }
            signal (mutex)
}
// I'm reading
   unlock_reader()        {
           wait (mutex);
        reader_cnt --;
        if (reader-cnt == 0) {
            signal (writer);
        signal (mutex);
}
```

→ when we see there is no reader left anybody could go i.e reader and writer

Conditional variables

* Conditional variables are variables that represent certain conditions. Each conditional variables represent one condition.

* wait or cond-wait :-

→ when a thread calls cond-wait :
The caller is (always) put into the queue of that condition variable.
(In semaphore if count > 0 we don't block, we use the resource, in this always the caller blocks)

* signal or cond-signal :-

↳ If queue has threads waiting then one of them is released

↳ otherwise signal is lost.

* Works with monitor.

* If there are waiting threads, one of them will be released and as a result there

will are 2 threads executing within the monitor

- ✓ One of them is caller that triggers release and the other is one being released.
- ✓ Thre should be only one thread executing using mutex.

* Let a thread efficiently wait for a change to shared state that is protected by a lock.

* cond_wait()

↳ caller is put into the CV queue

↳ {a member of queue is released which changes the shared resource}

↳ releases CV mutex.

* cond_signal()

↳ if there are threads waiting in CV queue, one of them is released.

↳ otherwise, CV signal is lost.