Semaphores

Requirements for a mutual exclusion primitive:

- Must allow only one process into a critical section at a time.
- If several requests are made at once, must allow one process to proceed.
- Processes must be able to go on vacation *outside* critical sections.

Desirable properties for a mutual exclusion mechanism:

- Fair: if several processes are waiting, let each in eventually.
- Efficient: don't use of substantial amounts of resources when waiting. E.g. no busy waiting.
- Simple: should be easy to use. E.g. just bracket the critical sections.

Desirable properties of threads using the mechanism:

- Always lock before manipulating shared data.
- Always unlock after manipulating shared data.
- Do not lock again if you have already locked it.
- Do not unlock if you didn't lock it (usually; there are a few exceptions to this).
- Do not spend large amounts of time in critical sections.

Semaphores: A synchronization variable that takes on positive integer variables. Invented by Edsger Dijkstra in the mid-60's. A semaphore has two operations:

- **P** (dutch for "to test", proberen): an atomic operation that waits for semaphore to become positive, then decrements it by 1.
- **V** (dutch for "to increment", verhogen): an atomic operation that increments semaphore by 1.

Simple and elegant mechanism useful for mutual exclusion as well as other things.

```
class Semaphore
{
public:
    P(void);
    V(void);
private:
    int value;   // semaphore value
    List *queue; // queue of waiters
    int tas;     // test and set integer
};
Semaphore::P(void)
{
    IntStatus oldLevel = interrupt->SetLevel(IntOff);
    while (TAS(tas) != 0);    // Do nothing
    if (value > 0) {
        value = value - 1;
        tas = 0;
        (void)interrupt->SetLevel(oldLevel);
        return;
    }
    queue->Append((void *)currentThread);
    tas = 0;
    (void)interrupt->SetLevel(oldLevel);
    currentThread->Sleep();
}

Semaphore::V(void)
{
    IntStatus oldLevel = interrupt->SetLevel(IntOff);
    while (TAS(tas != 0));    // do nothing
    Thread t* = (Thread *)queue->Remove
    if (t == NULL)
        value += 1;
    else                 // schedule it
        scheduler->ReadyToRun(t);
    tas = 0;
    (void)interrupt->SetLevel(oldLevel);
}
```