

Zu Aufgabe 14

Als gegeben werden folgende Klassen mit den entsprechenden Methoden vorausgesetzt:

mutex_t mit lock(), unlock()

zum Schutz von kritischen Abschnitten auf Maschinenebene (vgl. Aufg. 9);

queue_t mit enqueue(), dequeue()

zum Einfügen bzw. Streichen eines Threads aus einer Warteschlange (Rückgabewert von dequeue() ist ein Zeiger auf einen TCB);

tcb_t mit

block() zum Blockieren des Threads, für dessen TCB diese Methode aufgerufen wird

release() zum Freisetzen des Threads, für dessen TCB diese Methode aufgerufen wird

get_current_thread() zum Ermitteln des aktuellen Threads (Rückgabewert: Zeiger auf TCB).

```
class sem_t {
private:
    int         _count;
    queue_t *   _queue;
    mutex_t *   _mutex;

public:
    sem_t (int n);
    ~sem_t();
    void P();
    void V();
};
```

```
/* Constructor */
sem_t::sem_t (int n) {
    _count = n;
    _queue = new queue_t();
    _mutex = new mutex_t(0); // 0 means unlocked
}
```

```
/* Destructor */
sem_t::~sem_t() {
    delete _queue;
    delete _mutex;
}
```

```
void sem_t::P() {
    _mutex->lock();
    _count--;
    if (_count >= 0)
        _mutex->unlock();
    else {
        tcb_t *tcb = get_current_thread();
        _queue->enqueue (tcb);
        _mutex->unlock(); // this piece of code
        tcb->block(); // should be atomic
    }
}
```

```
void sem_t::V() {
    _mutex->lock();
    _count++;
    if (_count > 0)
        _mutex->unlock();
    else {
        tcb_t *tcb = _queue->dequeue();
        _mutex->unlock();
        tcb->release();
    }
}
```

```
sem_t s(4); // Example initialization
```