ALP4 - Nichtsequentielle Programmierung 5

Tobias Kranz (414 71 30)
Johannes Rohloff ()

22. Mai 2013

Aufgabe 1

a) Prüfen auf Zyklen ist mittels Tiefensuche möglich.

b) Die Laufzeit beträgt, wenn der Graph als Adjazensliste gespeichert wird, $\theta(\#\text{Knoten} + \#\text{Kanten})$ (lineare Laufzeit) bzw. $\theta(\#\text{Knoten}^2)$ (quadratische Laufzeit) sofern er als Adjazensmatrix vorliegt.

Aufgabe 2

a)

```
#include <pthread.h>

//Loesung analog zu der in Go

//Kompiliert so nicht da keine main(), sonst keine Errors

unsigned int nR = 0;
unsigned int nW = 0;

unsigned int rCnt = 0;
unsigned int wCnt = 0;

pthread_mutex_t mutex = PTHREAD_MUTEX_INITIALIZER;

pthread_cond_t okR = PTHREAD_COND_INITIALIZER;
pthread_cond_t okW = PTHREAD_COND_INITIALIZER;

void readerIn(){
        pthread_mutex_lock(&mutex);
```

```
        if (nW > 0 || wCnt>0) {
                rCnt++;
                pthread_cond_wait(&okR, &mutex);
                rCnt--;
        }
        nR++;

        pthread_mutex_unlock(&mutex);
}

void readerOut(){
        pthread_mutex_lock(&mutex);
        nR--;
        if (nR == 0) {
                pthread_cond_signal(&okW);
        }
        pthread_mutex_unlock(&mutex);
}

void writerIn(){
        pthread_mutex_lock(&mutex);
        if (nR > 0 || nW > 0) {
                wCnt++;
                pthread_cond_wait(&okW, &mutex);
                wCnt--;
        }
        nW = 1;
        pthread_mutex_unlock(&mutex);
}

void writerOut(){
        pthread_mutex_lock(&mutex);
        nW = 0;

        if (rCnt>0) {
                pthread_cond_signal(&okR);
        } else {
                pthread_cond_signal(&okW);
        }
        pthread_mutex_unlock(&mutex);
}
```

b)

```
package ReaderWriterProblem
```

```
import . "sync"

//Loesung moeglichst dicht an [MAURER] S.137 f. wobei wie auch auf S.135 von
//signal-and-continue ausgegangen (also das mutex erst nach dem signal freigege

//'go build LS-Monitor.go' ohne Fehler

type Imp struct {
        nR, nW, rCnt, wCnt uint
        mutex              Mutex
        okR, okW           *Cond
}

func New() *Imp {
        x := new(Imp)
        x.okR = NewCond(&x.mutex)
        x.okW = NewCond(&x.mutex)
        return x
}

//Schreiber mit Prioritaet
func (x *Imp) ReaderIn() {
        x.mutex.Lock()
        //Awaited() um auf blockierte Reader/Writer zu pruefen gibts hier nicht
        //daher zaehlen rCnt bzw. wCnt mit
        if x.nW > 0 || x.wCnt > 0 {
                x.rCnt++
                x.okR.Wait()
                x.rCnt--
        }
        x.nR++
        x.mutex.Unlock()
}

func (x *Imp) ReaderOut() {
        x.mutex.Lock()
        x.nR--
        if x.nR == 0 {
                x.okW.Signal()
        }
        x.mutex.Unlock()
}

func (x *Imp) WriterIn() {
        x.mutex.Lock()
        if x.nR > 0 || x.nW > 0 {
                x.wCnt++
```

```
                x.okW.Wait()
                x.wCnt--
        }
        x.nW = 1
        x.mutex.Unlock()
}

//Priorisiert Reader
func (x *Imp) WriterOut() {
        x.mutex.Lock()
        x.nW = 0
        if x.rCnt > 0 {
                x.okR.Signal()
        } else {
                x.okW.Signal()
        }
        x.mutex.Unlock()
}
```