Freie Universität Berlin

Prof. Dr. Margarita Esponda

Nichtsequentielle Programmierung, SoeSe 2017

Übungsblatt 5

TutorIn: Lilli Walter
Tutorium 6

Boyan Hristov, Sergelen Gongor

20. Juni 2017

Link zum Git Repository: https://github.com/BoyanH/FU-Berlin-ALP4/tree/master/Solutions/Homework5

Aufgabe 1

Zu zeigen: Await-Bedingung für $P_i \equiv B \equiv inD \leq afterF$

$$\begin{aligned} wp(inD \leftarrow inD + 1, PCI) =& wp(inD \leftarrow inD + 1, (inD \leq afterF + 1) \wedge (inF \leq afterD)) = \\ =& (inD + 1 \leq afterF + 1) \wedge (inF \leq afterD) \qquad \text{(Zuweisungsregel)} \\ =& (inD \leq afterF) \wedge (inF \leq afterD) \end{aligned}$$

Da es $P \wedge INV \wedge B \Rightarrow wp(S, Q \wedge INV)$ gelten muss ist hier $B = inD \leq afterF$

□

Zu zeigen: Await-Bedingung für $C_i \equiv B \equiv inF < afterD$

$$\begin{aligned} wp(inF \leftarrow inF + 1, PCI) =& wp(inF \leftarrow inF + 1, (inD \leq afterF + 1) \wedge (inF \leq afterD)) = \\ =& (inD \leq afterF + 1) \wedge (inF + 1 \leq afterD) = \qquad \text{(Zuweisungsregel)} \\ =& (inD \leq afterF + 1) \wedge (inF < afterD) = \end{aligned}$$

Da es $P \wedge INV \wedge B \Rightarrow wp(S, Q \wedge INV)$ gelten muss ist hier $B = inF < afterD$

□

Aufgabe 2

Leider war das Pseudo-Code, das in der Vorlesungsfolien stand, nicht richtig. Wir haben es repariert und danach die Bedingung für die Lese verändert -> diese können frei reinkommen ohne warten nur wenn es keine wartende Schreiber gibt und auch nur dann können sie weitere Leser reinlassen.

```java
package fu.alp4;

public class Writer extends IDataUser {

    public void run() {
        while (true) {
            try {
                // take a random rest to simulate different scenarios
                randomNap(5000, 10000);

                E.acquire();

                /**
                 * If there are currently some other writers or readers, wait
                 * for an available writer spot to be freed from a reader
                 * In te given time, release E, but remember to acquire it before
                 * incrementing nw++ for synchronization
                 */
                if (nw > 0 || nr > 0) {
                    dw++;
                    System.out.println("Waits to start writing! Stop letting further readers!");
                    E.release();
                    W.acquire();
                    E.acquire();
                }

                nw++;
                System.out.printf("Started writing; nr: %s; nw: %s; dr: %s; dw: %s\n", nr, nw, dr, dw);
                E.release();

                randomNap(2000, 4000);

                E.acquire();
                nw--;

                if (dr > 0 && dw == 0) {
                    dr--;
                    R.release();
                } else if (dw > 0) {
                    /**
                     * Deferred writers have higher priority, because we thought it's important to
                     * let writers as soon as possible so readers get the latest and greatest ^^
                     *
                     * E.g if both writers and readers are waiting, let the writer in
                     */

                    dw--;
                    W.release();
                }
                System.out.printf("Finished writing; nr: %s; nw: %s; dr: %s; dw: %s\n", nr, nw, dr, dw);
                E.release();


            } catch (InterruptedException e) {
                e.printStackTrace();
            }
        }
    }
}
```

```java
package fu.alp4;
```

```
public class Reader extends IDataUser {

    public void run() {
        while (true) {
            // take a random rest to simulate different scenarios
            randomNap(500, 2000);
            try {
                E.acquire();
                // skip waiting only if there are no deferred or non-deferred writers!
                // if a writer is waiting, he has the priority
                if (nw > 0 || dw > 0) {
                    dr++;
                    E.release();
                    R.acquire();
                    E.acquire();
                }

                nr++;
                // again, waiting writers have priority, don't let further readers in this
    case
                if (dr > 0 && dw == 0) {
                    dr--;
                    R.release();
                }

                System.out.printf("Started reading; nr: %s; nw: %s; dr: %s; dw: %s\n", nr,
    nw, dr, dw);
                E.release();

                // read
                randomNap(500, 2000);


                E.acquire();
                nr--;
                if (nr == 0 && dw > 0) {
                    dw--;
                    W.release();
                }

                System.out.printf("Finished reading; nr: %s; nw: %s; dr: %s; dw: %s\n", nr,
     nw, dr, dw);
                E.release();

            } catch (InterruptedException e) {
                e.printStackTrace();
            }
        }
    }
}
```

```
package fu.alp4;

public class Main {

    public static void main(String[] args) {

        for (int i = 0; i < 5; i++) {
            if (i < 4) {
                new Reader().start();
            } else {
                new Writer().start();
            }
        }
    }
}
```

Aufgabe 3

a) $\sum_{n=1}^{n} C_{ij} + A_j = E_j$ wobei C ist die Belegungsmatrix, A der Ressourcenrestvektor, E der Ressourcenvektor (Aus Vorlesungsfolien). Also an dem Beispiel $\sum_{n=1}^{n} C_{ij} + R_j = E_j$ wobei R der Ressourcenrestvektor ist.

$$\Rightarrow R_j = E_j - \sum_{n=1}^{n} C_{ij}$$
$$\Rightarrow R = [(3-(1+1)), (15-(1+3+6)), (12-(1+5+3+1)), (11-(1+4+2+4))] =$$
$$= [1, 5, 2, 0]$$

Der Ressourcenrestvektor bzw. die noch vorhande Ressourcen R = [1, 5, 2, 0]

b) Das System befindet sich in einem sicheren Zustand, weil es eine Scheduling-Reihenfolge gibt, die nicht zu Deadlock führt. Solche Reihenfolge ist z.B:

$$T_4 \Rightarrow R = [1, 11, 5, 2]$$
$$\rightarrow T_5 \Rightarrow R = [1, 11, 6, 6]$$
$$\rightarrow T_1 \Rightarrow R = [1, 11, 7, 7]$$
$$\rightarrow T_2 \Rightarrow R = [2, 12, 6, 6]$$
$$\rightarrow T_3 \Rightarrow R = [3, 15, 12, 11]$$

$R = E \Rightarrow$ sicheren Zustand

c) Da R = [1,5,2,0] wird nach dem Teilanforderung von Thread T_2 R = [1,2,0,0]. Das ist kein sicheren Zustand, da alle Threads danach (inklusive T_2 mit seinem neuen Restanforderung von [0,3,3,0]) mindestens eins von den letzten zwei Ressourcen brauchen, es gibt aber keine mehr vorhanden. Deswegen gibt es auch keine Scheduling-Reihenfolge, die nicht zu einem Deadlock führt.

$\Rightarrow$ soll nicht bedient werden.